Copley Motion Library

Generated by Doxygen 1.8.13

Contents

Chapter 1

Copley Motion Library

1.1 Introduction

The Copley Motion Library is a collection of C++ objects which are intended to simplify the development of CANopen and EtherCAT based products. These libraries allow low level access to the CANopen and EtherCAT networks while also providing high level methods for easy development of network based motion control applications.

1.1.1 Supported platforms

The Copley Motion Library is designed to be highly platform independent. The only requirements of a platform are:

- C++ must be supported. The Copley Motion Library makes heavy use of the object oriented features of the C++ language. Porting it to standard C would require significant effort.
- Multi-tasking. The library uses multiple threads of execution. For this reason, some type of multi-tasking operating system is required. A real time operating system is desirable, but not necessary.
- Network hardware. Some sort of CAN hardware must be available for CANopen support. An Ethernet port must be available for EtherCAT support.

The majority of the library code should be easily portable across platforms, even to those systems that do not have a very complete C++ implementation. In particular, the standard C and C++ libraries have been avoided to the extent possible.

1.1.2 Multi-tasking support

The C++ language does not define any standard method for multi-tasking, so a generic multi-tasking layer has been defined for use by this library. Three classes have been defined in the header file CML_Threads.h which define the multi-tasking interface. These classes are;

- Thread: An independent thread of execution.
- Mutex: A mechanism used to provide mutually exclusive access to a variable or system resource.
- Semaphore: A mechanism used to control access to a resource pool. This object allows threads to pend on it with timeouts.

When porting the library to a new environment, these three objects will need to be implemented for that environment. The libraries currently support multi-tasking under posix compatible operating systems (most Unix varieties including Linux), and MS Windows.

1.1.3 CAN hardware support

The Copley Motion Libraries have been designed to use a generic interface to the low level CAN network hardware. This makes porting the libraries to new CAN hardware very straight forward. All CAN hardware access is routed through the CanInterface object defined in the header file CML_Can.h. This object defines generic methods to open, close, read from, and write to the CAN network. Adding support for new CAN interface hardware is as simple as implementing a new CanInterface object derived from this class.

1.1.4 EtherCAT hardware support

When communicating over an EtherCAT network, CML uses the standard operating system API calls to access the Ethernet hardware. No special device drivers other then those normally used to access the Ethernet hardware are required.

Note that EtherCAT requires a dedicated Ethernet port. It is generally not possible to share a single Ethernet port for both a EtherCAT and general Ethernet communications at the same time.

1.2 CANopen basics

CAN (Control Area Network) is a serial network originally developed for use in the automotive industry. The physical layer of CAN consists of a two wire differentially driven bus, typically terminated with 120 Ohm resisters on each end. The maximum bit rate supported by CAN is 1,000,000 bits/second for up to 25 meters. Lower bit rates may be used for longer network lengths.

Communication over the CAN bus takes the form of network packets. Each packet consists of an identifier, some control bits, and zero to eight bytes of data. Each packet is sent with CRC information which allows the CAN controllers on the network to identify and re-send incorrectly formatted packets.

The identifier sent with each CAN packet identifies the type of packet being sent, as well as the priority of the packet. If two or more devices on the network attempt to transmit packets at the same time, the one sending the higher priority packet will succeed. The device sending the lower priority packet will detect the network collision and automatically back off the network and re-try the transmission later. The fact that the higher priority packet is transmitted when a network collision occurs allows very high network bandwidth utilization. Other network technologies (such as Ethernet) would require both transmitting devices to abort their transmissions if a network collision was detected.

CANopen is a high level protocol used to communicate over a CAN network. It allows blocks of data larger then the eight byte CAN limit to be transmitted as a single entity. Additionally, CANopen defines a standard framework for device operation which simplifies communication between dissimilar devices.

In most cases a CANopen network consists of one master device, and multiple slave devices (also called network nodes). Each of the nodes on the network has a 7-bit node ID number associated with it in the range 1 to 127 (the node ID 0 is reserved, and should not be used). The Copley Motion Libraries are designed to run on the CANopen network master.

1.2.1 CANopen Object Dictionary

One of the central concepts defined by the CANopen protocol is the notion of an object dictionary. This is essentially a collection of parameters on each device which define it's configuration and status. Most communication over the CANopen network consists of uploading data to, and downloading data from the object dictionaries of the various devices.

Each entry in the object dictionary is accessed using a 16-bit index value. Most entries in a device's object dictionary are simple atomic types (16-bit integers, 32-bit integers, strings, etc), however entries may be defined as complex types (records or arrays). In this case, the individual elements in the record or array are accessed using an additional 8-bit sub-index. It is not possible for these objects to be of complex types, so nesting of structures is not allowed.

1.2.2 SDO

The majority of traffic over the CANopen network consists of the network master reading and writing to the object dictionaries of the slave devices (nodes) on the network. The primary mechanism through which this is accomplished is the Service Data Object (SDO).

Each node on the CANopen network is required to implement at least one SDO. The SDO is essentially a channel that can be opened between a network master and a slave for the purpose of reading from and writing to the slave's object dictionary.

SDO transfers are always initiated by the master, and always confirmed by the slave. The synchronous nature of the SDO makes error detection very straight forward. However, since every transfer using an SDO takes at least two CAN messages (one from the master to the slave, one from the slave to the master) they can be somewhat slow for the transfer of real time information.

For example, for the network master to update an object which holds an eight byte long value, six CAN messages will be required:

- 1.The master sends a message to the slave indicating it's intentions to update the object. In this message it sends the object's index and sub-index values. It also passes the size (in bytes) of the data that will be transferred.
- 2.The slave responds to the master indicating that it is ready to receive the data.
- 3.The master sends the first 7 bytes of data. SDO transfers use one byte of the CAN message data for header information, so the largest amount of data that can be passed in any single message is 7 bytes.
- 4.The slave responds indicating that it received the data and is ready for more.
- 5.The master sends the remaining byte of data.
- 6.The slave responds indicating success.

1.2.3 PDO

A second method of accessing the data in the node's object dictionaries is defined by CANopen. This method is called the Process Data Object (PDO) and is primarily used to transfer frequently changing real-time data.

Unlike an SDO, a PDO can be initiated by either the network master, or the slave device. In fact, unlike SDO transfers, PDO transfers do not really follow a master - slave model. Any device on the network can initiate a PDO transfer, and a PDO message can be received and processed by multiple devices on the network.

Every PDO message consists of exactly one CAN message. Unlike SDO transfers, there is no confirmation with PDO transfers. This has the benefit of making much better use of the CAN network bandwidth, but since there is no response to a PDO, some other mechanism must be found to determine if the message was received successfully.

For example, updating the value of an 8-byte long variable in a device's object dictionary using a PDO can be accomplished in one CAN message, unlike the 6 messages that it required using SDO transfers.

- 1.The master sends a CAN message containing the 8-bytes of data. No additional header information is passed in the message, and no response is sent.

Of course, since all 8 bytes of the CAN message data were used to hold the object's value, there is no place the transmit the index / sub-index of the object being updated. This information must be implied based on the CAN message ID associated with the PDO. To create this association, a PDO must be mapped to one or more objects in a device's object dictionary before it can be used. This is accomplished by using SDO transfers, and effectively tells the device which object(s) a particular PDO transfer will access.

The result is that sending a PDO like in the above example requires a bit of setup using SDO transfers. It's therefore not useful to use a PDO transfer to update an object's value just once, since this could be more efficiently done using a single SDO transfer. However, if a particular object needs to be updated repeatedly, then the overhead of mapping a PDO to it makes more sense.

Aside from efficiency issues, PDO transfers have some other useful features. SDO transfers are inherently one master to one slave. PDO transfers can be used to broadcast a message to multiple other devices on the network. This is useful for synchronizing the start of a multi-axis move, for example.

Additionally, a slave device can be configured to transmit the value of one (or more) of the objects in it's object dictionary either at a set frequency, or when some internal event occurs. This would also be accomplished using PDO transfers.

1.2.4 Network management

Most communication over the CANopen network consists of reading and writing values to a device's object dictionary using either SDO or PDO transfers. There are however several other message types which are required by the CANopen protocol.

Network management messages are used to control the state of the devices on the CANopen network. Every CANopen device implements a simple state machine which is controlled through the use of these messages. The following states are defined:

- Pre-operational: Every node enters this state after power-up or reset. In this state, the device is not functional, but will communicate over the CANopen network. PDO transfers are not allowed in pre-operational state, but SDO transfers may be used.

- Operational: This is the normal operating state for all devices. SDO and PDO transfers are both allowed.
- Stopped: No communication is allowed in this state except for network management messages. Neither SDO nor PDO transfers may be used.

One use of network management messages is to control these state changes on the network devices. The following network management messages are sent by the network manager to control these state changes. Each of these messages can be either sent to a single node (by node ID), or broadcast to all the nodes on the network.

- Reset. This message causes the receiving node(s) to perform a soft reset and come up in pre-operation state.
- Reset communications. Causes the receiving node(s) to reset their CANopen network to it's power-on state, and enter pre-operational state. This is not a full device reset, just a reset of the CANopen interface.
- Pre-operational. This message causes the receiving node(s) to enter pre-operational state. No reset is performed.
- Start. Causes the node(s) to enter operational state.
- Stop. Causes the node(s) to enter stopped state.

In addition to controlling the device's state machines, network management messages can be used to monitor the operation of the nodes on the CANopen network. There are two protocols to perform this task; heartbeat and node guarding.

The heartbeat protocol is very simple, the network master configures the node to transmit a heartbeat message at some interval. The heartbeat message is then sent by the node at the specified frequency. The only information passed in the heartbeat message is the current state of the node (i.e. pre-operational, operational or stopped). The network manager can monitor these messages, and if anything happens to the node (or the network connection to the node) it can detect this by the lack of heartbeat messages.

Node guarding is similar. The network manager configures the node to expect node guarding messages at some interval. The network manager then sends a message to the configured node at that frequency, and the node responds with a node guarding message. This allows both the network manager and slave device to identify a network failure if the guarding messages stop.

1.2.5 SYNC messages

Another type of message defined by the CANopen protocol is the SYNC message. This is a message that one device is configured to transmit at some interval, and that all other devices on the network receive. It can be used for device synchronization, and PDO transfers can be configured to be sent in response to the SYNC event.

Every CANopen network should have one (and only one) device which is configured to produce SYNC messages (the SYNC producer). Other devices on the network which receive the SYNC messages are SYNC consumers.

Copley Controls amplifiers are able to be configured as both SYNC producers and SYNC consumers. The default configuration for every amplifier after reset is as a SYNC consumer. The Copley Motion Library Amp class constructor will configure one of the amplifiers as a SYNC producer by default.

1.2.6 Emergency messages

Emergency messages are sent by CANopen devices when some error condition is detected. They contain information about the type of error condition as well as manufacturer specific information not defined by the CANopen spec.

1.3 EtherCAT basics

The EtherCAT network is a high performance field bus based on the Ethernet physical layer. There are several different high level protocols that can be implemented on top of EtherCAT, but the most common protocol, and the one supported by CML, is the CANopen over EtherCAT (CoE) protocol. When using this protocol, the EtherCAT network acts very much like a high speed CANopen network supporting an object dictionary, SDO and PDO access just like in CANopen.

One significant difference between the EtherCAT network and CANopen network is that in the CANopen network the nodes are able to transmit their status updates any time they like. In the EtherCAT network the network master needs to poll the status of the nodes.

1.4 Architectural overview

The Copley Motion Libraries are made up of a large number of classes, but several are of primary importance and will be used for every program. This section gives a brief overview of the most important of these classes. Detailed documentation of each of the provided classes is provided later in the manual.

1.4.1 CanInterface

This class has already been mentioned above. It is one of the few classes that is highly platform dependent. The CanInterface class is used to abstract the CAN network interface hardware available on the system.

Typically, one of the first objects created in a program using the Copley Motion Libraries is a CanInterface object. This will in turn be passed to the CanOpen object.

1.4.2 EtherCatHardware

This class is the generic representation of the Ethernet hardware interface used by the operating system. Like the CanInterface object, the exact details of this class's implementation will be different depending on the operating system being used.

1.4.3 Network class

The Network class represents an EtherCAT or CANopen network.

1.4.4 CanOpen class

The CanOpen class is derived from the Network class and represents the CANopen network. The Open method of this class must be called before the class can be used. This method takes a reference to a CanInterface object as it's only parameter. The CanOpen class will then attempt to open the CAN interface (by calling CanInterface::Open method), and if that is successful it will start a new thread which will be responsible for reading messages from the CAN interface.

It is possible to have more then one CanOpen object in a system. Each should be connected to a distinct CAN network, and each should therefore be passed a distinct CanInterface object.

The primary responsibility of the CanOpen object is to listen for messages on the CAN network. To perform this task, the CanOpen object creates a separate high priority thread which constantly monitors the CAN network for new messages. Every time a new message is received, the read thread searches for a Receiver object associated with the message's CAN ID. If such an object is found, it's handler method is called to process the message.

1.4.5 EtherCAT class

The EtherCAT class is derived from the Network class and represents the EtherCAT network. The Open method of this class must be called before the class can be used. This method takes a reference to a EtherCatHardware object as it's only parameter. That object will be used to perform the low level communications with the Ethernet hardware.

1.4.6 Receiver class

Receiver objects are used to listen for messages on the CAN network. Each Receiver object has a CAN message ID associated with it. When a message is received which has this ID value, the CanOpen object will search for a Receiver object with that ID. If no such Receiver is found, the message will be ignored. If a Receiver object with a matching ID is found, then the virtual Receiver::NewFrame() method will be called.

1.4.7 Node class

The Node class represents a node (slave device) on the CANopen network. This class is passed a node ID value, and a reference to a CanOpen object during construction. The CanOpen object identifies which network the Node is connected to.

The Node class defines various methods to read and write standard objects from the node's object dictionary. The objects that are built into the Node class are those that are defined in the CANopen communication specification (DS301).

The Node class also includes logic which allows it to transmit and monitor the node guarding (or heartbeat) protocols for the device on the network. If the device being monitored stops responding, or changes state, the virtual method HandleStateChange() will be called. By default this does nothing, but it may be extended to perform any necessary action to handle the condition.

1.4.8 Amp class

The Amp inherits from the Node class, and is used to represent a Copley Controls amplifier on the CANopen network. This class includes numerous methods used to get and set various amplifier parameters. The Amp class also includes several high level methods used to make point to point moves, home the amplifier, and stream complex PVT style profiles down to the amplifier.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

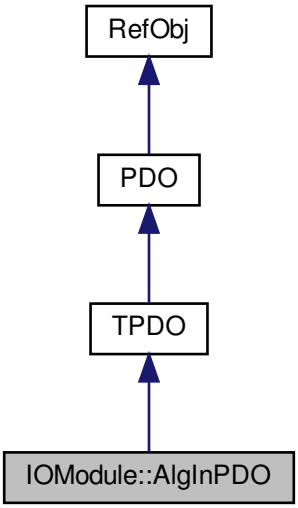
Chapter 5

Class Documentation

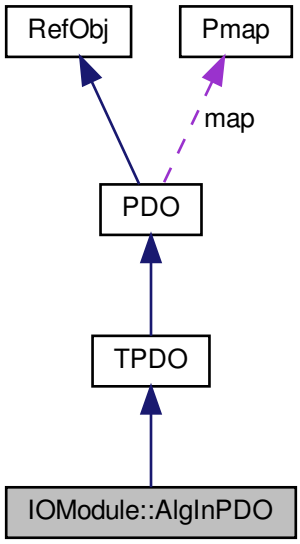
5.1 IOModule::AlgInPDO Class Reference

Transmit PDO for mapping analog inputs.

Inheritance diagram for IOModule::AlgInPDO:



Collaboration diagram for IOModule::AlgInPDO:



Public Member Functions

- const Error * Init (class IOModule *io, uint32 cobID, uint8 ct, uint8 id[ ], IOMODULE_EVENTS event)

  Initialize a analog input PDO object.
- bool GetInVal (uint8 id, int16 &value)

  Read the specified input from the PDO's cached data.
- void Received (void)

  New transmit PDO received.

Additional Inherited Members

5.1.1 Detailed Description

Transmit PDO for mapping analog inputs.

This class represents the standard transmit PDO which can be used to map up to 4 16-bit analog inputs.

5.1.2 Member Function Documentation

5.1.2.1 GetInVal()

```
bool GetInVal (
            uint8 id,
            int16 & value )
```

Read the specified input from the PDO's cached data.

The value returned will be the last value received via PDO for this input bank.

Parameters

id	The input ID to be checked.
value	The input value will be returned here. If the input is not mapped to this PDO, then this will not be changed.

Returns

true if the value was returned, false if the input isn't mapped to this PDO.

5.1.2.2 Init()

```
const Error * Init (
            class IOModule * io,
            uint32 cobID,
            uint8 ct,
            uint8 id[],
            IOMODULE_EVENTS event )
```

Initialize a analog input PDO object.

Parameters

io	Pointer to the I/O module to which this PDO is assigned.
cobID	The CAN ID for this PDO message.
ct	The number of inputs to be mapped (1 to 4)
id	An array of ct input ID numbers. These will be mapped (in order) to the PDO.
event	The event bit to post when a PDO message is received.

Returns

A pointer to an error object, or NULL on success

5.1.2.3 Received()

```
void Received (
            void  )  [virtual]
```

New transmit PDO received.

This method is called by the CANopen reader thread when a new PDO message is received. It causes this PDO object to post it's event to the IOModule object's event map. This will cause any waiting threads to wake up.

Reimplemented from TPDO.

The documentation for this class was generated from the following files:

- CML_IO.h
- IOmodule.cpp

5.2 AlgoPhaseInit Struct Reference

Configuration structure used to set up algorithmic phase init.

Public Member Functions

- AlgoPhaseInit (void)
  Default constructor.

Public Attributes

- uint16 phaseInitCurrent
  Maximum Current to use with algorithmic phase initialization. (0.01 amp units)
- uint16 phaseInitTime
  Algorithmic phase initialization timeout. (milliseconds)
- uint16 phaseInitConfig
  Algorithmic Phase Initialization config.

5.2.1 Detailed Description

Configuration structure used to set up algorithmic phase init.

These settings may be up/download from the amplifier using the functions Amp::SetAlgoPhaseInit and Amp::GetAlgo↩ PhaseInit.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AlgoPhaseInit()

```
AlgoPhaseInit (
            void  )  [inline]
```

Default constructor.

Initializes all structure elements to zero.

5.2.3 Member Data Documentation

5.2.3.1 phaseInitConfig

```
uint16 phaseInitConfig
```

Algorithmic Phase Initialization config.

Bit mapped as follows (Bit 0 If clear, use algorithmic phase initialization. If set force the phase angle to 0. bits 1 - 15 reserved

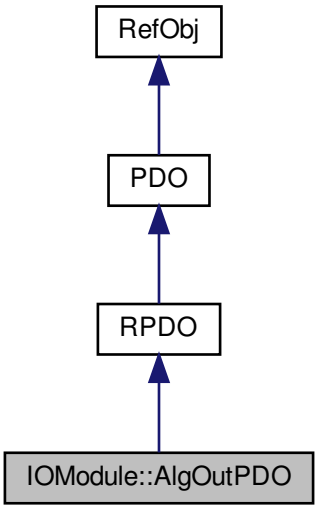
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

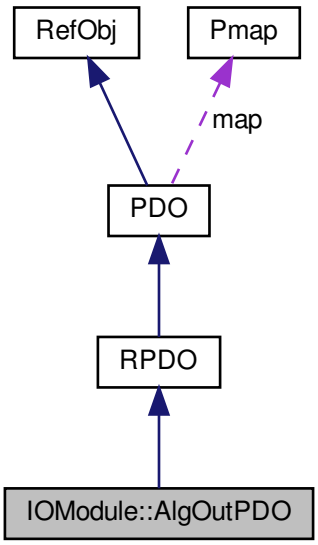
5.3 IOModule::AlgOutPDO Class Reference

Receive PDO for mapping analog outputs.

Inheritance diagram for IOModule::AlgOutPDO:



Collaboration diagram for IOModule::AlgOutPDO:



Public Member Functions

- const Error * Init (class IOModule *io, uint32 cobID, uint8 ct, uint8 id[ ])

  Initialize an analog output PDO object.
- bool Update (uint8 id, int16 value)

  Update the locally stored value of one of the 16-bit analog outputs associated with this PDO.
- const Error * Transmit (void)

  Transmit this PDO.

Additional Inherited Members

5.3.1 Detailed Description

Receive PDO for mapping analog outputs.

This class represents the standard receive PDO which can be used to transmit up to 4 16-bit analog outputs.

5.3.2 Member Function Documentation

5.3.2.1 Init()

```
const Error * Init (
            class IOModule * io,
            uint32 cobID,
            uint8 ct,
            uint8 id[] )
```

Initialize an analog output PDO object.

Parameters

io	Pointer to the I/O module to which this PDO is assigned.
cobID	The CAN ID for this PDO message.
ct	The number of outputs to be mapped (1 to 4)
id	An array of ct output ID numbers. These will be mapped (in order) to the PDO.

Returns

A pointer to an error object, or NULL on success

5.3.2.2 Transmit()

```
const Error * Transmit (
            void  )
```

Transmit this PDO.

Returns

A pointer to an error object, or NULL on success

5.3.2.3 Update()

```
bool Update (
            uint8 id,
            int16 value )
```

Update the locally stored value of one of the 16-bit analog outputs associated with this PDO.

Parameters

id	The output block ID to be updatad.
value	The new value for the output block.

Returns

true if the value was updated, false if the block isn't mapped to this PDO.

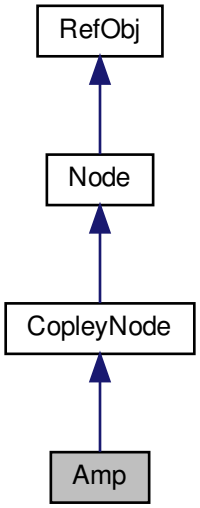
The documentation for this class was generated from the following files:

- CML_IO.h
- IOmodule.cpp

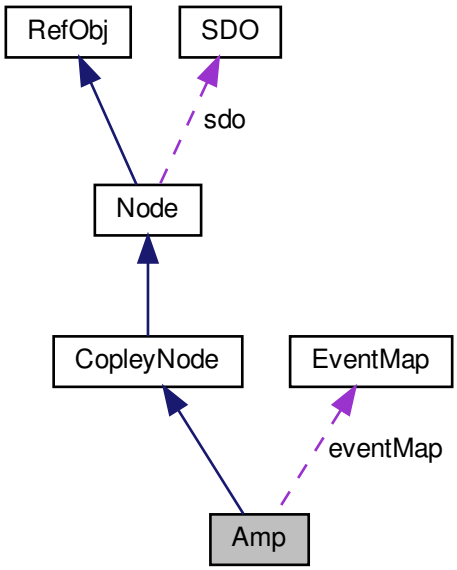
5.4 Amp Class Reference

Copley Controls amplifier object.

Inheritance diagram for Amp:



Collaboration diagram for Amp:



Public Member Functions

- virtual const Error * UpdateEvents (uint16 stat, uint32 events, uint16 inputs)

  Update the amplifier's event map based on the status information received by a status PDO.
- void PvtStatusUpdate (uint32 status)

  This is an internal function which should not be called by user code.
- virtual uint32 GetNetworkRef (void)

  Return a reference ID to the network that this node is attached to.
- virtual NodeState GetState (void)

  Returns the present state of this node.
- const Error * GetStatusWord (uint16 &value)

  Get the current value of the drive's status word.
- virtual const Error * SetControlWord (uint16 value)

  Set the amplifier's control word.
- const Error * GetControlWord (uint16 &value)

  Returns the present value of the CANopen device profile control word.

Amplifier initialization

- Amp ()

  Default constructor.
- Amp (Network &net, int16 nodeID)

  Construct and initialize an amplifier object using defaults for all amp settings.
- Amp (Network &net, int16 nodeID, AmpSettings &settings)

  Construct and initialize an amplifier object.
- virtual ~Amp ()

  Amp object destructor.
- const Error * Init (Network &net, int16 nodeID)

  Initialize the amplifier object using all default settings.
- const Error * Init (Network &net, int16 nodeID, AmpSettings &settings)

  Initialize the amplifier object with custom amp settings.
- const Error * ReInit (void)

  Re-initialize an amplifier.
- const Error * Reset (void)

  Reset the amplifier object.
- const Error * InitSubAxis (Amp &primary, int axis=2)

  Initialize an Amp object for use with a secondary axis of a multi-axis EtherCAT amplifier.

Amplifier modes & status info

- const Error * GetAmpName (char *name)

  Get the amplifier name stored in the amplifiers flash.
- const Error * SetAmpName (char *name)

  Set the amplifier name stored in the amplifiers flash.
- const Error * SetAmpMode (AMP_MODE mode)

  Set the amplifier mode of operation.
- const Error * GetAmpMode (AMP_MODE &mode)

  Get the currently active amplifier mode of operation.
- const Error * Disable (bool wait=true)

  Disable the amplifier.
- const Error * Enable (bool wait=true)

Enable the amplifier.

- bool IsHardwareEnabled (void)

  Return true if the amplifier's PWM outputs are currently enabled.
- bool IsSoftwareEnabled (void)

  Return true if the amplifier is being enabled by software.
- bool IsReferenced (void)

  Return true if the amplifier has been successfully referenced (homed).
- const Error * CheckStateForMove (void)

  Check the amplifier's state to make sure a move can be started.
- const Error * ClearFaults (void)

  Clear amplifier faults.
- const Error * GetFaults (AMP_FAULT &value)

  Get any active amplifier faults.
- const Error * SetFaultMask (AMP_FAULT &value)

  Set the amplifier's fault mask.
- const Error * GetFaultMask (AMP_FAULT &value)

  Get the current value of the amplifier's fault mask.
- const Error * GetEventStatus (EVENT_STATUS &stat)

  Get the amplifier's 'event status' register.
- const Error * GetEventSticky (EVENT_STATUS &stat)

  Get the amplifier's 'sticky' event status register.
- const Error * GetEventLatch (EVENT_STATUS &stat)

  Get the amplifier's latched event status register.
- const Error * SetPwmMode (AMP_PWM_MODE mode)

  Set the PWM output mode configuration for the amplifier.
- const Error * GetPwmMode (AMP_PWM_MODE &mode)

  Get the current PWM output mode configuration from the amplifier.
- const Error * SetPhaseMode (AMP_PHASE_MODE mode)

  Set the phasing mode configuration for the amplifier.
- const Error * GetPhaseMode (AMP_PHASE_MODE &mode)

  Get the current phasing mode configuration from the amplifier.

Motor & Amplifier state information

- const Error * GetPositionActual (uunit &value)

  Get the actual position used by the servo loop.
- const Error * SetPositionActual (uunit value)

  Set the actual position.
- const Error * GetPositionMotor (uunit &value)

  Get the actual motor position.
- const Error * SetPositionMotor (uunit value)

  Set the actual motor position.
- const Error * GetPositionLoad (uunit &value)

  Get the load encoder position.
- const Error * SetPositionLoad (uunit value)

  Set the load encoder position.
- const Error * GetPositionCommand (uunit &value)

  Get the instantaneous commanded position.
- const Error * GetPositionError (uunit &value)

  Get the position error.
- const Error * GetVelocityActual (uunit &value)

  Get the actual motor velocity.
- const Error * GetVelocityLoad (uunit &value)

  Get the load encoder velocity.

- const Error * GetVelocityCommand (uunit &value)
  Get the commanded velocity.
- const Error * GetVelocityLimited (uunit &value)
  Get the limited velocity.
- const Error * GetCurrentActual (int16 &value)
  Get the actual motor current.
- const Error * GetCurrentCommand (int16 &value)
  Get the commanded motor current.
- const Error * GetCurrentLimited (int16 &value)
  Get the limited motor current.
- const Error * GetTrajectoryVel (uunit &value)
  Get the instantaneous commanded velocity passed out of the trajectory generator.
- const Error * GetTrajectoryAcc (uunit &value)
  Get the instantaneous commanded acceleration passed out of the trajectory generator.
- const Error * GetTrajectoryJrkAbort (uunit &value)
  Get the jerk value used during trajectory aborts.
- const Error * SetTrajectoryJrkAbort (uunit value)
  Set the jerk value used during trajectory aborts.
- const Error * GetPhaseAngle (int16 &value)
  Get the motor phase angle.
- const Error * GetHallState (int16 &value)
  Get the current digital hall sensor state.
- const Error * GetRefVoltage (int16 &value)
  Get the analog reference input voltage.
- const Error * GetHighVoltage (int16 &value)
  Get the high voltage bus voltage in units of 0.1 volts.
- const Error * GetAmpTemp (int16 &value)
  Get the current amplifier temperature (degrees C).
- const Error * GetAnalogEncoder (int16 &sin, int16 &cos)
  Get the raw voltage on the two analog encoder inputs (0.1 millivolt units).
- const Error * GetMotorCurrent (int16 &u, int16 &v)
  Get the actual current values read directly from the amplifier's current sensors.

Input & Output pin control

- const Error * SetIoConfig (AmpIoCfg &cfg)
  Configure the amplifier's programmable I/O pins using the values passed in the config structure.
- const Error * GetIoConfig (AmpIoCfg &cfg)
  Read the amplifier's programmable I/O pin configuration and return it in the passed config structure.
- const Error * GetInputs (uint16 &value, bool viaSDO=false)
  Get the present value of the general purpose input pins.
- const Error * GetInputs32 (uint32 &value)
  32-bit version to Get the present value of the general purpose input pins.
- const Error * WaitInputEvent (Event &e, Timeout timeout, uint32 &match)
  Wait on the amplifier's general purpose input pins.
- const Error * WaitInputHigh (uint32 inputs, Timeout timeout=-1)
  Wait for any of the specified general purpose input pins to be set.
- const Error * WaitInputLow (uint32 inputs, Timeout timeout=-1)
  Wait for any of the specified general purpose input pins to be lowered.
- const Error * SetIoPullup (uint16 value)
  Set the current state of the input pin pull up/down resisters.
- const Error * GetIoPullup (uint16 &value)
  Get the current state of the input pin pull up/down resisters.
- const Error * SetInputConfig (int8 pin, INPUT_PIN_CONFIG cfg, uint16 axis=0)

Set the input pin configuration for the specified input pin.

- const Error * GetInputConfig (int8 pin, INPUT_PIN_CONFIG &cfg)

  Get the input pin configuration for the specified input pin.
- const Error * GetInputConfig (int8 pin, INPUT_PIN_CONFIG &cfg, uint16 &axis)

  Get the input pin configuration for the specified input pin.
- const Error * SetInputDebounce (int8 pin, int16 value)

  Set the input pin debounce time for the specified input pin.
- const Error * GetInputDebounce (int8 pin, int16 &value)

  Get the input pin debounce time for the specified input pin.
- const Error * SetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG cfg, uint32 mask1=0, uint32 mask2=0, uint16 axis=0)

  Set the output pin configuration for the specified pin.
- const Error * GetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG &cfg)

  Get the output configuration for the specified pin.
- const Error * GetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG &cfg, uint16 &axis)

  Get the output configuration for the specified pin.
- const Error * GetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG &cfg, uint32 &mask)

  Get the output pin configuration for the specified pin.
- const Error * GetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG &cfg, uint32 &mask, uint16 &axis)

  Get the output pin configuration for the specified pin.
- const Error * GetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG &cfg, uint32 &mask1, uint32 &mask2)

  Get the output pin configuration for the specified pin.
- const Error * GetOutputConfig (int8 pin, OUTPUT_PIN_CONFIG &cfg, uint32 &mask1, uint32 &mask2, uint16 &axis)

  Get the output pin configuration for the specified pin.
- const Error * SetOutputs (uint16 value)

  Update the state of the manual output pins.
- const Error * GetOutputs (uint16 &value)

  Get the present value of the output pin control register.
- const Error * SetIOOptions (int32 value)

  Set the IO Options.
- const Error * GetIOOPtions (int32 &value)

  Get the IO Options.
- const Error * SetIoPullup32 (int32 value)

  32 Bit version of SetIoPullup.
- const Error * GetIoPullup32 (int32 &value)

  32 Bit version of GetIoPullup.

Position capture

- const Error * SetPosCaptureCfg (POS_CAPTURE_CFG cfg)

  Set the position capture configuration.
- const Error * GetPosCaptureCfg (POS_CAPTURE_CFG &cfg)

  Read the current configuration of the position capture mechanism.
- const Error * GetPosCaptureStat (POS_CAPTURE_STAT &stat)

  Read the current status of the position capture mechanism.
- const Error * GetIndexCapture (int32 &value)

  Get the most recently captured encoder index position.
- const Error * GetHomeCapture (int32 &value)

  Get the most recently captured home sensor position.

General parameter setup.

- const Error * GetAmpConfig (AmpConfig &cfg)

Read the complete amplifier configuration from the amplifier and return it in the passed structure.

- const Error * SetAmpConfig (AmpConfig &cfg)

  Update an amplifier's configuration from the passed structure.
- const Error * SaveAmpConfig (AmpConfig &cfg)

  Upload the passed amplifier configuration to the amplifier's workign memory, and then copy that working memory to flash.
- const Error * SaveAmpConfig (void)

  Save all amplifier parameters to internal flash memory.
- const Error * LoadFromFile (const char *name, int &line)

  Load the specified amplifier data file.
- const Error * LoadCCDFromFile (const char *name, Network &net)

  Checks the network type and calls the correct function to write the ccd file to the amplifier.
- const Error * GetCanNetworkConfig (CanNetworkConfig &cfg)

  Get the current CANopen network configuration programmed into the amplifier.
- const Error * SetCanNetworkConfig (CanNetworkConfig &cfg)

  Set the CANopen node ID and bit rate configuration.
- const Error * GetNetworkOptions (NetworkOptions &cfg)

  Set the Network Options configuration.
- const Error * SetNetworkOptions (NetworkOptions &cfg)

  Set the Network Options configuration.
- const Error * GetAmpInfo (AmpInfo &info)

  Read the Amplifier information parameters from the drive.
- const Error * GetMtrInfo (MtrInfo &info)

  Read the motor information structure from the amplifier.
- const Error * SetMtrInfo (MtrInfo &info)

  Update the amplifier's motor information.
- const Error * GetRegenConfig (RegenConfig &cfg)

  Upload the current configuration parameters for the power regeneration resister connected to the amplifier.
- const Error * SetRegenConfig (RegenConfig &cfg)

  Download a new configuration structure for the power regeneration resister.
- const Error * GetUstepConfig (UstepConfig &cfg)

  Upload the current configuration parameters for the stepper.
- const Error * SetUstepConfig (UstepConfig &cfg)

  Download a new configuration structure for the microstepper.
- const Error * GetAlgoPhaseInit (AlgoPhaseInit &cfg)

  Upload the current configuration parameters for the algorithmic phase init.
- const Error * SetAlgoPhaseInit (AlgoPhaseInit &cfg)

  Download a new configuration structure for the algorithmic phase init.
- const Error * GetCammingConfig (CammingConfig &cfg)

  Upload the current configuration parameters for the camming.
- const Error * SetCammingConfig (CammingConfig &cfg)

  Download a new configuration structure for Camming.
- const Error * GetGainScheduling (GainScheduling &cfg)

  Upload the current configuration parameters for the GainS cheduling.
- const Error * SetGainScheduling (GainScheduling &cfg)

  Download a new configuration structure for Gain Scheduling.
- const Error * GetDAConverterConfig (DAConfig &cfg)

  Get the configuration for the D/A converter.
- const Error * SetDAConverterConfig (DAConfig &cfg)

  Set the D/A converter configuration.
- const Error * GetSpecialFirmwareConfig (AmpConfig &cfg)

  Reads the amplifier's special firmware.
- const Error * SetSpecialFirmwareConfig (AmpConfig &cfg)

  Configure the amplifier's special firmware.
- const Error * GetEncoderErrorConfig (EncoderErrorConfig &cfg)

Read the amplifier's encoder error configuration.

- const Error * SetEncoderErrorConfig (EncoderErrorConfig &cfg)

  Configure the amplifier's encoder error configuration.

Low level access to CANopen objects.

- const Error * Download (int16 index, int16 sub, int32 size, byte *data)

  Download data to an object in this Amps object dictionary.
- const Error * Upload (int16 index, int16 sub, int32 &size, byte *data)

  Upload data from an object in this Amps object dictionary.
- const Error * Dnld32 (int16 index, int16 sub, uint32 data)

  Download data to an object in this Amps object dictionary.
- const Error * Dnld32 (int16 index, int16 sub, int32 data)

  Download data to an object in this Amps object dictionary.
- const Error * Upld32 (int16 index, int16 sub, uint32 &data)

  Upload data from an object in this Amps object dictionary.
- const Error * Upld32 (int16 index, int16 sub, int32 &data)

  Upload data from an object in this Amps object dictionary.
- const Error * Dnld16 (int16 index, int16 sub, uint16 data)

  Download data to an object in this Amps object dictionary.
- const Error * Dnld16 (int16 index, int16 sub, int16 data)

  Download data to an object in this Amps object dictionary.
- const Error * Upld16 (int16 index, int16 sub, uint16 &data)

  Upload data from an object in this Amps object dictionary.
- const Error * Upld16 (int16 index, int16 sub, int16 &data)

  Upload data from an object in this Amps object dictionary.
- const Error * Dnld8 (int16 index, int16 sub, uint8 data)

  Download data to an object in this Amps object dictionary.
- const Error * Dnld8 (int16 index, int16 sub, int8 data)

  Download data to an object in this Amps object dictionary.
- const Error * Upld8 (int16 index, int16 sub, uint8 &data)

  Upload data from an object in this Amps object dictionary.
- const Error * Upld8 (int16 index, int16 sub, int8 &data)

  Upload data from an object in this Amps object dictionary.
- const Error * DnldString (int16 index, int16 sub, char *data)

  Download data to an object in this Amps object dictionary.
- const Error * UpldString (int16 index, int16 sub, int32 &len, char *data)

  Upload data from an object in this Amps object dictionary.

Control loop setup

- const Error * GetPosLoopConfig (PosLoopConfig &cfg)

  Get the configuration values of the amplifiers position loop.
- const Error * SetPosLoopConfig (PosLoopConfig &cfg)

  Update the amplifier's position loop configuration.
- const Error * GetVelLoopConfig (VelLoopConfig &cfg)

  Get the configuration values of the amplifiers velocity loop.
- const Error * SetVelLoopConfig (VelLoopConfig &cfg)

  Update the amplifier's velocity loop configuration.
- const Error * GetCrntLoopConfig (CrntLoopConfig &cfg)

  Get the configuration values of the amplifiers current loop.
- const Error * SetCrntLoopConfig (CrntLoopConfig &cfg)

  Update the amplifier's current loop configuration.
- const Error * GetVloopOutputFilter (Filter &f)

Get the coefficients used in the velocity loop output filter.

- const Error * GetVloopOutputFilter (Filter &f)

  Set new coefficients for the velocity loop output filter.
- const Error * SetVloopOutputFilter (Filter &f)

  Get the coefficients used in the second velocity loop output filter.
- const Error * GetVloopOutputFilter2 (Filter &f)

  Set the coefficients used in the second velocity loop output filter.
- const Error * SetVloopOutputFilter2 (Filter &f)

  Get the coefficients used in the third velocity loop output filter.
- const Error * GetVloopOutputFilter3 (Filter &f)

  Set the coefficients used in the third velocity loop output filter.
- const Error * SetVloopOutputFilter3 (Filter &f)

  Get the coefficients used in the velocity loop command filter.
- const Error * GetVloopCommandFilter (Filter &f)

  Set new coefficients for the velocity loop command filter.
- const Error * SetVloopCommandFilter (Filter &f)

  Get the coefficients used in the current loop input filter.
- const Error * GetIloopCommandFilter (Filter &f)

  Set the coefficients used in the current loop input filter.
- const Error * SetIloopCommandFilter (Filter &f)

  Get the coefficients used in the second current loop input filter.
- const Error * GetIloopCommandFilter2 (Filter &f)

  Set the coefficients used in the second current loop input filter.
- const Error * SetIloopCommandFilter2 (Filter &f)

  Get the coefficients used in the velocity loop command filter.
- const Error * GetAnalogCommandFilter (Filter &f)

  Set new coefficients for the analog reference input filter.
- const Error * SetAnalogCommandFilter (Filter &f)

  Get the coefficients used in the input shaping filter.
- const Error * GetInputShapingFilter (InputShaper &f)

  Set new coefficients for the input shaping filter.
- const Error * SetInputShapingFilter (InputShaper &f)

  Get the servo loop configuration.
- const Error * GetServoLoopConfig (int32 &value)

  Set the servo loop configuration.
- const Error * SetServoLoopConfig (int32 value)

Position and velocity windows

- const Error * SetTrackingWindows (TrackingWindows &cfg)

  Update the amplifier's tracking window configuration.
- const Error * GetTrackingWindows (TrackingWindows &cfg)

  Get the configuration values of the amplifiers position & velocity tracking windows.
- const Error * SetPositionErrorWindow (uunit value)

  Set the position error window.
- const Error * GetPositionErrorWindow (uunit &value)

  Get the position error window.
- const Error * SetPositionWarnWindow (uunit value)

  Set the position warning window.
- const Error * GetPositionWarnWindow (uunit &value)

  Get the position warning window.
- const Error * SetSettlingWindow (uunit value)

  Set the position settling window.
- const Error * GetSettlingWindow (uunit &value)

  Get the position window value.

- const Error * SetSettlingTime (uint16 value)
  Set the position window time value (milliseconds).
- const Error * GetSettlingTime (uint16 &value)
  Get the position window time value (milliseconds).
- const Error * SetVelocityWarnWindow (uunit value)
  Set the velocity warning window.
- const Error * GetVelocityWarnWindow (uunit &value)
  Get the velocity warning window.
- const Error * SetVelocityWarnTime (uint16 value)
  Set the velocity warning window time value (milliseconds).
- const Error * GetVelocityWarnTime (uint16 &value)
  Get the position window time value (milliseconds).
- const Error * SetSoftLimits (SoftPosLimit &cfg)
  Set software limit switch settings.
- const Error * GetSoftLimits (SoftPosLimit &cfg)
  Upload the current software limit switch settings from the amplifier.

Homing mode.

- const Error * GoHome (void)
  Execute a home move.
- const Error * GoHome (HomeConfig &cfg)
  Execute a home move.
- const Error * SetHomeConfig (HomeConfig &cfg)
  Configure the amplifier's homing related parameters.
- const Error * GetHomeConfig (HomeConfig &cfg)
  Load a structure with all parameters related to homing the amplifier.
- const Error * SetHomeMethod (COPLEY_HOME_METHOD method, uint16 extended=0)
  Set the method used for homing the drive.
- const Error * GetHomeMethod (COPLEY_HOME_METHOD &method, uint16 *extended=0)
  Get the selected homing method.
- const Error * SetHomeOffset (uunit value)
  Set the home offset value.
- const Error * GetHomeOffset (uunit &value)
  Get the home offset value.
- const Error * SetHomeVelFast (uunit value)
  Set the home velocity used to move to a home switch.
- const Error * GetHomeVelFast (uunit &value)
  Get the home velocity used to move to a home switch.
- const Error * SetHomeVelSlow (uunit value)
  Set the home velocity used to find a switch edge.
- const Error * GetHomeVelSlow (uunit &value)
  Get the home velocity used to find a switch edge.
- const Error * SetHomeAccel (uunit value)
  Set the home acceleration.
- const Error * GetHomeAccel (uunit &value)
  Get the home acceleration.
- const Error * SetHomeCurrent (int16 value)
  Set the home current.
- const Error * GetHomeCurrent (int16 &value)
  Get the home current.
- const Error * SetHomeDelay (int16 value)
  Set the home delay.
- const Error * GetHomeDelay (int16 &value)

Get the home current.

- const Error * GetHomeAdjustment (uunit &value)

  Get the last home adjustment amount.

Quick stop support

- const Error * QuickStop (void)

  Perform a 'quick stop' on the axis.
- const Error * HaltMove (void)

  Halt the current move.
- const Error * SetQuickStopDec (uunit value)

  Set the quick stop deceleration value (i.e.
- const Error * GetQuickStopDec (uunit &value)

  Get the quick stop deceleration value.
- const Error * SetHaltMode (HALT_MODE mode)

  Set the halt mode.
- const Error * GetHaltMode (HALT_MODE &mode)

  Get the halt mode.
- const Error * SetQuickStop (QUICK_STOP_MODE mode)

  Set the quick stop mode.
- const Error * GetQuickStop (QUICK_STOP_MODE &mode)

  Get the quick stop mode.
- const Error * ClearHaltedMove (void)

  Clear out any old set-point after a move is halted.

Point to point move support (position profile mode)

- const Error * SetupMove (ProfileConfigTrap &cfg)

  Setup a point to point move, but do not start it.
- const Error * SetupMove (ProfileConfigScurve &cfg)

  Setup a point to point move, but do not start it.
- const Error * SetupMove (ProfileConfigVel &cfg)

  Setup a point to point move, but do not start it.
- const Error * DoMove (ProfileConfigTrap &cfg, bool relative=false)

  Perform a point to point move.
- const Error * DoMove (ProfileConfigScurve &cfg, bool relative=false)

  Perform a point to point move.
- const Error * DoMove (ProfileConfigVel &cfg)

  Perform a velocity profile move.
- const Error * DoMove (uunit pos, bool relative=false)

  Perform a point to point move.
- const Error * StartMove (bool relative=false)

  Start the move that's already been programmed.
- const Error * MoveAbs (uunit pos)

  Start an absolute point to point move to the specified position.
- const Error * MoveRel (uunit dist)

  Start a relative point to point move of the specified distance.
- const Error * SetProfileConfig (ProfileConfig &cfg)

  Configure the amplifier's parameters related to point-to-point moves.
- const Error * GetProfileConfig (ProfileConfig &cfg)

  Load a structure with all parameters related to point-to-point moves.
- const Error * SetProfileType (PROFILE_TYPE type)

  Set the motion profile type.
- const Error * GetProfileType (PROFILE_TYPE &type)

Get the currently selected motion profile type.

- const Error * SetTargetPos (uunit value)

  Set the profile target position (i.e.
- const Error * GetTargetPos (uunit &value)

  Get the profile target position.
- const Error * SetProfileVel (uunit value)

  Set the profile velocity value (i.e.
- const Error * GetProfileVel (uunit &value)

  Get the profile velocity value.
- const Error * SetProfileAcc (uunit value)

  Set the profile acceleration value (i.e.
- const Error * GetProfileAcc (uunit &value)

  Get the profile acceleration value.
- const Error * SetProfileDec (uunit value)

  Set the profile deceleration value (i.e.
- const Error * GetProfileDec (uunit &value)

  Get the profile deceleration value.
- const Error * SetProfileJerk (uunit value)

  Set the jerk limit used with S-curve profiles.
- const Error * GetProfileJerk (uunit &value)

  Get the currently programmed jerk limit for S-curve profiles.

Profile velocity mode support

- const Error * SetTargetVel (uunit value)

  Set the target velocity used in profile velocity mode.
- const Error * GetTargetVel (uunit &value)

  Get the target velocity used in profile velocity mode.

Profile torque mode support

- const Error * SetTorqueTarget (int16 value)

  Set the amplifier target torque value.
- const Error * GetTorqueTarget (int16 &value)

  Get the current target torque value.
- const Error * GetTorqueDemand (int16 &value)

  Get the torque demand value.
- const Error * GetTorqueActual (int16 &value)

  Get the actual torque being applied by the motor at the moment.
- const Error * SetTorqueSlope (int32 value)

  Set the rate of change of torque for use in profile torque mode (AMPMODE_CAN_TORQUE).
- const Error * GetTorqueSlope (int32 &value)

  Get the rate of change of torque for use in profile torque mode (AMPMODE_CAN_TORQUE).
- const Error * SetTorqueRated (int32 value)

  Set the motor rated torque parameter.
- const Error * GetTorqueRated (int32 &value)

  Get the motor rated torque parameter.

PVT (interpolated position) trajectory support

- const Error * SendTrajectory (Trajectory &trj, bool start=true)

  Upload a PVT move trajectory to the amplifier and optionally start the move.
- const Error * StartPVT (void)

Start a PVT move that has already been uploaded.

- const Error * GetPvtBuffFree (int16 &n)

  Get the number of free positions in the PVT segment buffer.
- const Error * GetPvtSegID (uint16 &id)

  Get the segment ID that the amplifier expects for the next PVT segment.
- const Error * GetPvtBuffStat (uint32 &stat)

  Get the amplifier's PVT buffer status word.
- const Error * GetPvtSegPos (uunit &pos)

  Get the starting position of the PVT segment currently active in the amplifier.

Amplifier event processing

- const Error * WaitEvent (Event &e, Timeout timeout=-1)

  Wait for an amplifier event condition.
- const Error * WaitEvent (Event &e, Timeout timeout, AMP_EVENT &match)

  Wait for an amplifier event condition.
- const Error * WaitMoveDone (Timeout timeout=-1)

  Wait for the currently running move to finish, or for an error to occur.
- const Error * WaitHomeDone (Timeout timeout=-1)

  Wait for the currently running homing move to finish, or for an error to occur.
- const Error * ClearNodeGuardEvent (void)

  This function attempts to clear a node guarding event condition.
- const Error * GetEventMask (AMP_EVENT &e)

  Get the current state of the amplifier's event mask.
- const Error * GetErrorStatus (bool noComm=false)

  Return an error object identifying the amplifiers status.

Unit conversion functions.

If unit conversions are enabled in CML_Settings.h, then these functions handle the details of converting from user position, velocity, acceleration & jerk units to the internal units used by the amplifier.

- virtual const Error * SetCountsPerUnit (uunit cts)

  Configure the user programmable units.
- virtual const Error * GetCountsPerUnit (uunit &cts)

  Get the number of encoder counts / user distance unit.
- virtual const Error * SetCountsPerUnit (uunit load, uunit mtr)

  Configure the user programmable units for a dual encoder system.
- virtual const Error * GetCountsPerUnit (uunit &load, uunit &mtr)

  Get the number of encoder counts / user distance unit for both encoders in a dual encoder system.
- virtual int32 PosUser2Mtr (uunit pos)

  Convert a position from user position units to internal amplifier units.
- virtual int32 VelUser2Mtr (uunit vel)

  Convert a velocity from user units to internal amplifier units.
- virtual int32 AccUser2Mtr (uunit vel)

  Convert an acceleration from user units to internal amplifier units.
- virtual uunit PosMtr2User (int32 pos)

  Convert a position from internal amplifier units to user units.
- virtual uunit VelMtr2User (int32 vel)

  Convert a velocity from internal amplifier units to user units.
- virtual uunit AccMtr2User (int32 vel)

  Convert an acceleration from internal amplifier units to user units.
- virtual int32 PosUser2Load (uunit pos)

  Convert a position from user position units to internal amplifier units.

- virtual int32 VelUser2Load (uunit vel)
  Convert a velocity from user units to internal amplifier units.
- virtual int32 AccUser2Load (uunit acc)
  Convert an acceleration from user units to internal amplifier units.
- virtual int32 JrkUser2Load (uunit jrk)
  Convert a jerk value from user units to internal amplifier units.
- virtual uunit PosLoad2User (int32 pos)
  Convert a position from internal amplifier units to user units.
- virtual uunit VelLoad2User (int32 vel)
  Convert a velocity from internal amplifier units to user units.
- virtual uunit AccLoad2User (int32 acc)
  Convert an acceleration from internal amplifier units to user units.
- virtual uunit JrkLoad2User (int32 jrk)
  Convert a jerk value from internal amplifier units to user units.

Linkage access

Amplifier object may be attached to a Linkage object.

In this case, multi- axis moves may be easily performed through calls to the Linkage object holding the amplifiers.

- class Linkage * GetLinkage (void)
  Return a pointer to the linkage that this amplifier is attached to.
- uint32 GetLinkRef (void)
  Return a reference to the linkage that this amplifier is attached to.

Non standard modes of operation.

These functions are used when running in modes other then the standard CANopen position modes.

Note that at present the libraries offer only very limited support for these modes.

- const Error * GetFuncGenConfig (FuncGenConfig &cfg)
  Upload the current configuration of the amplifier's internal function generator.
- const Error * SetFuncGenConfig (FuncGenConfig &cfg)
  Configure the amplifier's internal function generator.
- const Error * GetAnalogRefConfig (AnalogRefConfig &cfg)
  Upload the amplifier's analog reference input configuration.
- const Error * SetAnalogRefConfig (AnalogRefConfig &cfg)
  Configure the amplifier's analog reference input.
- const Error * GetPwmInConfig (PwmInConfig &cfg)
  Upload the amplifier's PWM input pin configuration.
- const Error * SetPwmInConfig (PwmInConfig &cfg)
  Configure the amplifier's PWM input pins.
- const Error * SetCurrentProgrammed (int16 crnt)
  Set the programmed current value in 0.01 Amp units.
- const Error * GetCurrentProgrammed (int16 &crnt)
  Get the programmed current value.
- const Error * SetVelocityProgrammed (uunit vel)
  Set the programmed velocity value.
- const Error * GetVelocityProgrammed (uunit &vel)
  Get the programmed velocity value.
- const Error * SetMicrostepRate (int16 rate)
  Set the amplifier microstepping rate.
- const Error * GetMicrostepRate (int16 &rate)
  Get the amplifier microstepping rate.

Trace control functions

The amplifier supports an internal 'tracing' mechanism which allows certain internal variables to be sampled at a fixed rate and stored to internal memory.

This mechanism is used by the CME program to implement the oscilloscope display of internal amplifier information. The following methods are available to allow this amplifier feature to be used by CML programs.

- const Error * GetTraceStatus (AMP_TRACE_STATUS &stat, int16 &samp, int16 &sampMax)
  Get the current status of the amplifier's trace system.
- const Error * GetTraceRefPeriod (int32 &per)
  Get the 'reference period' used with the amplifiers trace mechanism.
- const Error * GetTracePeriod (int16 &per)
  Get the period of time between trace samples.
- const Error * SetTracePeriod (int16 per)
  Set the period of time between trace samples.
- const Error * GetTraceTrigger (AMP_TRACE_TRIGGER &type, uint8 &chan, int32 &level, int16 &delay)
  Get the current configuration of the amplifier's trace trigger.
- const Error * SetTraceTrigger (AMP_TRACE_TRIGGER type, uint8 chan=0, int32 level=0, int16 delay=0)
  Configure the amplifier's trace trigger.
- const Error * GetTraceMaxChannel (uint8 &max)
  Return the maximum number of trace channels supported by the amplifier.
- const Error * GetTraceChannel (uint8 ndx, AMP_TRACE_VAR &value)
  Get the amplifier variable current selected on one of the trace channels.
- const Error * SetTraceChannel (uint8 ndx, AMP_TRACE_VAR value)
  Select an amplifier trace variable to be sampled.
- const Error * GetTraceData (int32 *data, int32 &max)
  Upload any trace data captured in the amplifier.
- const Error * TraceStart (void)
  Start collecting trace data on the amplifier.
- const Error * TraceStop (void)
  Stop collecting trace data on the amplifier.

Protected Member Functions

- const Error * ClearEventLatch (EVENT_STATUS stat)
  Clear the latched version of the amplifier's event status register.
- virtual void HandleStateChange (NodeState from, NodeState to)
  Handle an amplifier state change.
- const Error * FormatPosInit (int32 pos, uint8 *data)
  Format a PVT segment to set the initial 32-bit position on a drive.
- const Error * FormatPvtSeg (int32 pos, int32 vel, uint8 time, uint8 *buff)
  Format a PVT trajectory segment.
- const Error * FormatPtSeg (int32 pos, uint8 time, uint8 *buff)
  Format a PT trajectory segment.
- const Error * SetPvtInitialPos (int32 pos, bool viaSDO=true)
  Set the initial position for a PVT trajectory.
- const Error * PvtBufferFlush (bool viaSDO=true)
  Flush the amplifier's PVT trajectory buffer.
- const Error * PvtBufferPop (uint16 n=1, bool viaSDO=true)
  Pop the N most recently sent segments off the amplifier's PVT trajectory buffer.
- const Error * PvtClearErrors (uint8 mask, bool viaSDO=true)

Clear the specified PVT buffer errors.

- const Error * PvtWriteBuff (uint8 *buff, bool viaSDO=false)

  Write to the PVT buffer on the amp.

- const Error * PvtWriteBuff (uint8 buff[ ][8], int ct, bool viaSDO=false)

  Write multiple PVT buffers to the drive.

Protected Attributes

- EventMap eventMap

  This is an event map that is used to track amplifier events and state changes.

Friends

- class **Linkage**

Additional Inherited Members

5.4.1 Detailed Description

Copley Controls amplifier object.

This object represents a Copley Controls amplifier on the network.

The Amp object can be used directly for fairly easy control of an amplifier on the CANopen network. The object provides easy to use methods for setting and getting amplifier parameter blocks. It also handles many of the details of both point to point moves and the transfer of complex PVT profiles.

In addition, the standard Amp object provides several virtual functions which can be used in derived classes to signal the derived object on state changes or emergency conditions.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Amp() [1/3]

```
Amp (
            void  )
```

Default constructor.

Init() must be called before the Amp object may be used.

5.4.2.2 Amp() [2/3]

```
Amp (
            Network & net,
            int16 nodeID )
```

Construct and initialize an amplifier object using defaults for all amp settings.

Parameters

net	Reference to the network object for this amp.
nodeID	a valid node ID for the amp. For CANopen, the node ID should range from 1 to 127. For EtherCAT, node ID's >= 0 identify the node by it's EtherCAT alias. Negative ID's identify the node by network position (-1 is the first node, -2 is the second, etc).

5.4.2.3 Amp() [3/3]

```
Amp (
            Network & net,
            int16 nodeID,
            AmpSettings & settings )
```

Construct and initialize an amplifier object.

Parameters

net	Reference to the Network for this amp.
nodeID	a valid node ID for the amp. For CANopen, the node ID should range from 1 to 127. For EtherCAT, node ID's >= 0 identify the node by it's EtherCAT alias. Negative ID's identify the node by network position (-1 is the first node, -2 is the second, etc).
settings	Amplifier settings to be used.

5.4.3 Member Function Documentation

5.4.3.1 AccLoad2User()

```
uunit AccLoad2User (
            int32 acc )  [virtual]
```

Convert an acceleration from internal amplifier units to user units.

Internal to the amplifier, all accelerations are stored in units of 10 encoder counts / second^2. If user units are not enabled in CML_Settings.h, then user units are the same as amplifier units, and this function has no effect.

If user units are enabled, then this function converts from amplifier units to user units (defined using Amp::SetCounts↩PerUnit).

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution. To convert motor encoder accelerations, use Amp::AccMtr2User. On single encoder systems either of these functions can be used.

Parameters

acc	The acceleration in units of 10 encoder counts / second^2

Returns

The acceleration in user units

5.4.3.2 AccMtr2User()

```
uunit AccMtr2User (
            int32 acc )  [virtual]
```

Convert an acceleration from internal amplifier units to user units.

This function converts using motor encoder units on a dual encoder system. Load encoder accelerations can be converted using Amp::AccLoad2User.

Parameters

acc	The acceleration in units of 10 encoder counts / second^2

Returns

The acceleration in user units

5.4.3.3 AccUser2Load()

```
int32 AccUser2Load (
            uunit acc )  [virtual]
```

Convert an acceleration from user units to internal amplifier units.

Internal to the amplifier, all accelerations are stored in units of 10 encoder counts / second / second. If user units are not enabled in CML_Settings.h, then user units are the same as amplifier units, and this function has no effect.

If user units are enabled, then this function converts from user units (defined using Amp::SetCountsPerUnit) to these internal amplifier units.

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution. To convert motor encoder accelerations, use Amp::AccUser2Mtr. On single encoder systems either of these functions can be used.

Parameters

acc	The acceleration in user units

Returns

The acceleration in 10 encoder counts / second ^ 2 units

5.4.3.4 AccUser2Mtr()

```
int32 AccUser2Mtr (
            uunit acc )  [virtual]
```

Convert an acceleration from user units to internal amplifier units.

This function converts using motor encoder units on a dual encoder system. Load encoder accelerations can be converted using Amp::AccUser2Load.

Parameters

acc	The acceleration in user units

Returns

The acceleration in 10 encoder counts / second ^ 2 units

5.4.3.5 CheckStateForMove()

```
const Error * CheckStateForMove (
            void  )
```

Check the amplifier's state to make sure a move can be started.

This function is used internally by the functions that start moves & homing. It looks at the current state of the amplifier and returns an appropriate error code if something is wrong that would cause problems during a move/home.

Returns

A pointer to an error object, or NULL for no error

5.4.3.6 ClearEventLatch()

```
const Error * ClearEventLatch (
            EVENT_STATUS stat )  [protected]
```

Clear the latched version of the amplifier's event status register.

This function is protected and generally only intended for internal use.

Parameters

stat	Identifies which bits to clear. Any bit set in this parameter will be cleared in the latched event status register.

Returns

A pointer to an error object, or NULL on success

5.4.3.7 ClearFaults()

```
const Error * ClearFaults (
            void  )
```

Clear amplifier faults.

This function can be used to clear any latching faults on the amplifier (see Amp::SetFaultMask for details on latching fault conditions). It also clears tracking error conditions.

Returns

A pointer to an error object, or NULL on success

5.4.3.8 ClearHaltedMove()

```
const Error * ClearHaltedMove (
            void  )
```

Clear out any old set-point after a move is halted.

Returns

A pointer to an error object, or NULL on success.

5.4.3.9 ClearNodeGuardEvent()

```
const Error * ClearNodeGuardEvent (
            void  )
```

This function attempts to clear a node guarding event condition.

Node guarding events occur when the amplifier fails to respond to it's heartbeat protocol for some reason. This could be caused by a network wiring problem, slow processing on the master controller (causing the amplifier guard message to be delayed or lost), or an amplifier error such as a reset or power down.

In any case, once a node guarding error is identified, the error condition must be cleared before any new moves may be performed.

This function attempts to clear the node guarding event condition, however if it determines that the amplifier has been reset then it fails and returns the error object AmpError::Reset. In this case, the amplifier object must be reinitialized before it can be used. The amp may be reinitialized by calling Amp::Init or Amp::ReInit.

If node guarding error become a problem, it may mean that the guard time is set too low. This can be adjusted when the amplifier object is initialized by the values in the AmpSettings object.

Returns

A pointer to an error object, or NULL on success.

5.4.3.10 Disable()

```
const Error * Disable (
            bool wait = true )
```

Disable the amplifier.

Note that if the brake delays are in use, then the amplifier may still be enabled when this function returns success. The outputs will actually be disabled after the amplifier finishes the braking procedure.

Parameters

wait	Wait for confirmation from the amplifier if true (default).

Returns

A pointer to an error object, or NULL on success.

5.4.3.11 Dnld16() [1/2]

```
const Error * Dnld16 (
            int16 index,
            int16 sub,
            uint16 data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.12 Dnld16() [2/2]

```
const Error * Dnld16 (
            int16 index,
            int16 sub,
            int16 data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.13 Dnld32() [1/2]

```
const Error * Dnld32 (
            int16 index,
            int16 sub,
            uint32 data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.14 Dnld32() [2/2]

```
const Error * Dnld32 (
            int16 index,
            int16 sub,
            int32 data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.15 Dnld8() [1/2]

```
const Error * Dnld8 (
            int16 index,
            int16 sub,
            uint8 data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.16 Dnld8() [2/2]

```
const Error * Dnld8 (
            int16 index,
            int16 sub,
            int8 data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.17 DnldString()

```
const Error * DnldString (
            int16 index,
            int16 sub,
            char * data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The value to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.18 DoMove() [1/4]

```
const Error * DoMove (
            ProfileConfigTrap & cfg,
            bool relative = false )
```

Perform a point to point move.

The move will use the trapezoidal profile mode, and all parameters will be programmed before the move is started.

This function can also be used to update a move that is alread in progress.

Parameters

cfg	A structure holding all the move configuration parameters.
relative	This will be a relative move if true, absolute if false (default)

Returns

A pointer to an error object, or NULL on success.

5.4.3.19 DoMove() [2/4]

```
const Error * DoMove (
            ProfileConfigScurve & cfg,
            bool relative = false )
```

Perform a point to point move.

The move will use the S-curve profile mode, and all parameters will be programmed before the move is started.

Parameters

cfg	A structure holding all the move configuration parameters.
relative	This will be a relative move if true, absolute if false (default)

Returns

A pointer to an error object, or NULL on success.

5.4.3.20 DoMove() [3/4]

```
const Error * DoMove (
            ProfileConfigVel & cfg )
```

Perform a velocity profile move.

All parameters will be programmed before the move is started.

This function can also be used to update a move that is alread in progress.

Parameters

cfg	A structure holding all the move configuration parameters.

Returns

A pointer to an error object, or NULL on success.

5.4.3.21 DoMove() [4/4]

```
const Error * DoMove (
            uunit pos,
            bool relative = false )
```

Perform a point to point move.

It's assumed that the drive is already configured with the properly trajectory parameters (velocity, acceleration, deceleration, profile type, etc).

This function sets the trajectories target position to the passed value, and manipulates the control word to start the move. It can be used for either absolute moves or relative moves

Parameters

pos	Position to move to (absolute) or distance to move (relative).
relative	True if this is a relative move, false for absolute.

Returns

A pointer to an error object, or NULL on success.

5.4.3.22 Download()

```
const Error * Download (
              int16 index,
              int16 sub,
              int32 size,
              byte * data )
```

Download data to an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
size	The number of bytes of data to be downloaded
data	A character array holding the data to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.23 Enable()

```
const Error * Enable (
              bool wait = true )
```

Enable the amplifier.

Parameters

wait	If true, the function won't return until a status message from the amp is received indicating that it successfully enabled.

Returns

A pointer to an error object, or NULL on success.

5.4.3.24 FormatPosInit()

```
const Error * FormatPosInit (
            int32 pos,
            uint8 * data )  [protected]
```

Format a PVT segment to set the initial 32-bit position on a drive.

Parameters

pos	The 32-bit initial position
data	Buffer of at least 8 bytes where formatted data will be stored.

Returns

An error object

5.4.3.25 FormatPtSeg()

```
const Error * FormatPtSeg (
            int32 pos,
            uint8 time,
            uint8 * buff )  [protected]
```

Format a PT trajectory segment.

The position and time information passed to the function are organized into the proper format and stored in the passed buffer. The buffer is assumed to be at least 8 bytes long.

Parameters

pos	Position at start of segment (encoder counts)
time	Time till next segment (milliseconds)
buff	Points to the buffer where the message will be stored.

Returns

An error object

5.4.3.26 FormatPvtSeg()

```
const Error * FormatPvtSeg (
            int32 pos,
            int32 vel,
            uint8 time,
            uint8 * buff )  [protected]
```

Format a PVT trajectory segment.

The position, velocity and time information passed to the function are organized into the proper format and stored in the passed buffer. The buffer is assumed to be at least 8 bytes long.

Parameters

pos	Position at start of segment (encoder counts)
vel	Velocity at start of segment (0.1 counts/sec)
time	Time till next segment (milliseconds)
buff	Points to the buffer where the message will be stored.

Returns

An error object

5.4.3.27 GetAlgoPhaseInit()

```
const Error * GetAlgoPhaseInit (
            AlgoPhaseInit & cfg )
```

Upload the current configuration parameters for the algorithmic phase init.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.28 GetAmpConfig()

```
const Error * GetAmpConfig (
            AmpConfig & cfg )
```

Read the complete amplifier configuration from the amplifier and return it in the passed structure.

This structure holds every amplifier parameter that can be stored to the amplifier's internal flash memory. The contents of the structure represent the complete amplifier configuration.

Parameters

cfg	The structure which will hold the uploaded configuration.

Returns

A pointer to an error object, or NULL on success

5.4.3.29 GetAmpInfo()

```
const Error * GetAmpInfo (
            AmpInfo & info )
```

Read the Amplifier information parameters from the drive.

These parameters describe the amplifiers capabilities. They are read only.

Parameters

info	A structure that will be filled with the amplifier info

Returns

A pointer to an error object, or NULL on success

5.4.3.30 GetAmpMode()

```
const Error * GetAmpMode (
            AMP_MODE & mode )
```

Get the currently active amplifier mode of operation.

Parameters

mode	The active mode of operation is returned here

Returns

A pointer to an error object, or NULL on success.

5.4.3.31 GetAmpName()

```
const Error * GetAmpName (
            char * name )
```

Get the amplifier name stored in the amplifiers flash.

Parameters

name	The name of the drive is returned here. This buffer should be at least 50 bytes long to avoid overflow.

Returns

A pointer to an error object, or NULL on success

5.4.3.32 GetAmpTemp()

```
const Error * GetAmpTemp (
            int16 & value )
```

Get the current amplifier temperature (degrees C).

Parameters

value	The value will be returned in this variable

Returns

A pointer to an error object, or NULL on success

5.4.3.33 GetAnalogCommandFilter()

```
const Error * GetAnalogCommandFilter (
            Filter & f )
```

Get the coefficients used in the velocity loop command filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.34 GetAnalogEncoder()

```
const Error * GetAnalogEncoder (
             int16 & sin,
             int16 & cos )
```

Get the raw voltage on the two analog encoder inputs (0.1 millivolt units).

If the amplifier has analog encoder inputs, then they will be read and returned.

Parameters

sin	The sine input of the analog encoder will be returned here.
cos	The cosine input of the analog encoder will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.35 GetAnalogRefConfig()

```
const Error * GetAnalogRefConfig (
            AnalogRefConfig & cfg )
```

Upload the amplifier's analog reference input configuration.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.36 GetCammingConfig()

```
const Error * GetCammingConfig (
            CammingConfig & cfg )
```

Upload the current configuration parameters for the camming.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.37 GetCanNetworkConfig()

```
const Error * GetCanNetworkConfig (
            CanNetworkConfig & cfg )
```

Get the current CANopen network configuration programmed into the amplifier.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.38 GetControlWord()

```
const Error * GetControlWord (
            uint16 & value )
```

Returns the present value of the CANopen device profile control word.

Parameters

value	Returns the control word value

Returns

A pointer to an error object, or NULL on success

5.4.3.39 GetCountsPerUnit() [1/2]

```
const Error * GetCountsPerUnit (
            uunit & cts )  [virtual]
```

Get the number of encoder counts / user distance unit.

This function is only available when user units are selected in CML_Settings.h.

This value defaults to 1.0 (i.e. user distance units are in encoder counts). It can be adjusted if some other distance unit is desired.

This value controls velocity, acceleration, and jerk units also. These units are always based on a time interval of seconds.

Parameters

cts	The count value will be returned here

Returns

A pointer to an error object, or NULL on success

5.4.3.40 GetCountsPerUnit() [2/2]

```
const Error * GetCountsPerUnit (
            uunit & load,
            uunit & mtr )  [virtual]
```

Get the number of encoder counts / user distance unit for both encoders in a dual encoder system.

This function is only available when user units are selected in CML_Settings.h.

These values default to 1.0 (i.e. user distance units are in encoder counts). It can be adjusted if some other distance unit is desired.

These values control velocity, acceleration, and jerk units also. These units are always based on a time interval of seconds.

Parameters

load	The load encoder scaling factor will be returned here
mtr	The motor encoder scaling factor will be returned here

Returns

A pointer to an error object, or NULL on success

5.4.3.41 GetCrntLoopConfig()

```
const Error * GetCrntLoopConfig (
            CrntLoopConfig & cfg )
```

Get the configuration values of the amplifiers current loop.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

5.4.3.42 GetCurrentActual()

```
const Error * GetCurrentActual (
            int16 & value )
```

Get the actual motor current.

This current is based on the amplifiers current sensors, and indicates the portion of current that is being used to generate torque in the motor.

The current is returned in units of 0.01 amps.

Parameters

value	A variable that will store the returned value.

Returns

A pointer to an error object, or NULL on success

5.4.3.43 GetCurrentCommand()

```
const Error * GetCurrentCommand (
            int16 & value )
```

Get the commanded motor current.

This current is the input to the current limiter. This value is also the output of the velocity loop when the motor is in either position or velocity control mode. When in current control mode, the commanded current is derived from the control source (analog input, PWM input, function generator, etc).

The current is returned in units of 0.01 amps.

Parameters

value	A variable that will store the returned value.

Returns

A pointer to an error object, or NULL on success

5.4.3.44 GetCurrentLimited()

```
const Error * GetCurrentLimited (
            int16 & value )
```

Get the limited motor current.

The commanded current (GetCurrentCommand) is passed to a current limiter. The output of the current limiter is the limited current which is passed as an input to the current loop.

The current is returned in units of 0.01 amps.

Parameters

value	A variable that will store the returned value.

Returns

A pointer to an error object, or NULL on success

5.4.3.45 GetCurrentProgrammed()

```
const Error * GetCurrentProgrammed (
            int16 & crnt )
```

Get the programmed current value.

This parameter is the current that the amplifier will attempt to maintain when set to the mode AMPMODE_PROG_CRNT.

Parameters

crnt	The current will be returned here (0.01 Amp units).

Returns

A pointer to an error object, or NULL on success

5.4.3.46 GetDAConverterConfig()

```
const Error * GetDAConverterConfig (
            DAConfig & cfg )
```

Get the configuration for the D/A converter.

Parameters

cfg	A structure where the configuration will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.47 GetEncoderErrorConfig()

```
const Error * GetEncoderErrorConfig (
            EncoderErrorConfig & cfg )
```

Read the amplifier's encoder error configuration.

Parameters

cfg	A structure that holds the encoder error configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.48 GetErrorStatus()

```
const Error * GetErrorStatus (
            bool noComm = false )
```

Return an error object identifying the amplifiers status.

This function causes the amplifier object to examine it's event mask and return an error object corresponding to the most serious error present.

Parameters

noComm	If true, then no CAN message communications will be performed by this function. This is useful if the function is being called from a CAN message handler which can't perform SDO communications. If false (default), then the amplifier may be queried for more detailed error information.

Returns

A pointer to an error object, or NULL if no errors are present

5.4.3.49 GetEventLatch()

```
const Error * GetEventLatch (
            EVENT_STATUS & stat )
```

Get the amplifier's latched event status register.

This register is a copy of the normal event status register in which bits are latched, i.e. they are set bit not cleared. Bits in this register are only cleared in response to a Amp::ClearEventLatch function call (which is protected by the Amp object). This register is primarily used internally by the Amp object to detect reset conditions on the amplifier.

Parameters

stat	The register status is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.50 GetEventMask()

```
const Error * GetEventMask (
            AMP_EVENT & e )
```

Get the current state of the amplifier's event mask.

The event mask is a bit-mapped variable identifies many interesting elements of the amplifiers state. The contents of this variable are built up from several different amplifier status words which are constantly updated over the CANopen network.

When the event mask is read using this function, no new messages are passed over the network. The current value of the event mask is simply returned. Any time the amplifier's state changes, it sends a message over the CANopen network which is used to update this mask.

It is also possible to wait on a particular value for this mask. See Amp::WaitEvent for details.

Parameters

e	The amplifier's event mask is returned here

Returns

A pointer to an error object, or NULL on success.

5.4.3.51 GetEventStatus()

```
const Error * GetEventStatus (
            EVENT_STATUS & stat )
```

Get the amplifier's 'event status' register.

This is the main register used internal to the amplifier to describe it's current state.

Parameters

stat	The register status is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.52 GetEventSticky()

```
const Error * GetEventSticky (
            EVENT_STATUS & stat )
```

Get the amplifier's 'sticky' event status register.

This register is a copy of the amplifiers event status register in which bits are set normally, but only cleared when the register is read (i.e. the bits are 'sticky'). It's useful for checking for transitory events which might me missed by reading the standard event status register.

Parameters

stat	The register status is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.53 GetFaultMask()

```
const Error * GetFaultMask (
            AMP_FAULT & value )
```

Get the current value of the amplifier's fault mask.

This mask identifies which error conditions will be treated as latching faults by the amplifier.

Parameters

value	The fault mask will be returned here

Returns

A pointer to an error object, or NULL on success

5.4.3.54 GetFaults()

```
const Error * GetFaults (
            AMP_FAULT & value )
```

Get any active amplifier faults.

Parameters

value	A bit mask identifying the active faults will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.55 GetFuncGenConfig()

```
const Error * GetFuncGenConfig (
            FuncGenConfig & cfg )
```

Upload the current configuration of the amplifier's internal function generator.

Parameters

cfg	A structure where the configuration will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.56 GetGainScheduling()

```
const Error * GetGainScheduling (
            GainScheduling & cfg )
```

Upload the current configuration parameters for the GainS cheduling.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.57 GetHallState()

```
const Error * GetHallState (
            int16 & value )
```

Get the current digital hall sensor state.

The hall state is the value of the hall sensors after any adjustments have been made to them based on the hallWiring parameter of the MtrInfo structure.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.58 GetHaltMode()

```
const Error * GetHaltMode (
            HALT_MODE & mode )
```

Get the halt mode.

This mode defines what happens when a halt command is issued to the amplifier.

Parameters

mode	The mode will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.59 GetHighVoltage()

```
const Error * GetHighVoltage (
            int16 & value )
```

Get the high voltage bus voltage in units of 0.1 volts.

Parameters

value	The value will be returned in this variable

Returns

A pointer to an error object, or NULL on success

5.4.3.60 GetHomeAccel()

```
const Error * GetHomeAccel (
            uunit & value )
```

Get the home acceleration.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.61 GetHomeAdjustment()

```
const Error * GetHomeAdjustment (
            uunit & value )
```

Get the last home adjustment amount.

The value returned is distance that the home position was adjusted by on the last successful home opperation.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.62 GetHomeCapture()

```
const Error * GetHomeCapture (
            int32 & value )
```

Get the most recently captured home sensor position.

Parameters

value	The captured position is returned here in units of encoder counts.

Returns

A pointer to an error object, or NULL on success

5.4.3.63 GetHomeConfig()

```
const Error * GetHomeConfig (
            HomeConfig & cfg )
```

Load a structure with all parameters related to homing the amplifier.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.64 GetHomeCurrent()

```
const Error * GetHomeCurrent (
            int16 & value )
```

Get the home current.

The homing current is returned in 0.01 Amp units (i.e. a value of 123 would be 1.23 Amps).

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.65 GetHomeDelay()

```
const Error * GetHomeDelay (
            int16 & value )
```

Get the home current.

The homing delay is returned in units of milliseconds.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.66 GetHomeMethod()

```
const Error * GetHomeMethod (
            COPLEY_HOME_METHOD & method,
            uint16 * extended = 0 )
```

Get the selected homing method.

Parameters

method	The home method will be returned here.
extended	If this pointer is non-null, then the extended homing method value will be returned here. If this pointer is null, then it will be ignored.

Returns

A pointer to an error object, or NULL on success

5.4.3.67 GetHomeOffset()

```
const Error * GetHomeOffset (
            uunit & value )
```

Get the home offset value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.68 GetHomeVelFast()

```
const Error * GetHomeVelFast (
            uunit & value )
```

Get the home velocity used to move to a home switch.

This velocity is used for any home moves that may be made at a high velocity without effecting the quality of the home sensor detection.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.69 GetHomeVelSlow()

```
const Error * GetHomeVelSlow (
            uunit & value )
```

Get the home velocity used to find a switch edge.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.70 GetIloopCommandFilter()

```
const Error * GetIloopCommandFilter (
            Filter & f )
```

Get the coefficients used in the current loop input filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.71 GetIloopCommandFilter2()

```
const Error * GetIloopCommandFilter2 (
            Filter & f )
```

Get the coefficients used in the second current loop input filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.72 GetIndexCapture()

```
const Error * GetIndexCapture (
            int32 & value )
```

Get the most recently captured encoder index position.

Parameters

value	The captured position is returned here in units of encoder counts.

Returns

A pointer to an error object, or NULL on success

5.4.3.73 GetInputConfig() [1/2]

```
const Error * GetInputConfig (
            int8 pin,
            INPUT_PIN_CONFIG & cfg )
```

Get the input pin configuration for the specified input pin.

Each of the amplifier input pins can be configured to perform some function. This function configures the specified input to perform the specified function.

Parameters

pin	The input pin to check. Input pins are numbered starting from 0. Check the amplifier datasheet for the number of input pins available.
cfg	The input pin function will be returned in this variable.

Returns

A pointer to an error object, or NULL on success

5.4.3.74 GetInputConfig() [2/2]

```
const Error * GetInputConfig (
            int8 pin,
            INPUT_PIN_CONFIG & cfg,
            uint16 & axis )
```

Get the input pin configuration for the specified input pin.

Each of the amplifier input pins can be configured to perform some function. This function configures the specified input to perform the specified function.

Parameters

pin	The input pin to check. Input pins are numbered starting from 0. Check the amplifier datasheet for the number of input pins available.
cfg	The input pin function will be returned in this variable.
axis	The axis number that this input pin is configured for

Returns

A pointer to an error object, or NULL on success

5.4.3.75 GetInputDebounce()

```
const Error * GetInputDebounce (
            int8 pin,
            int16 & value )
```

Get the input pin debounce time for the specified input pin.

Parameters

pin	The input pin to configure. Input pins are numbered starting from 0. Check the amplifier datasheet for the number of input pins available.
value	The pins debounce time (milliseconds) is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.76 GetInputs()

```
const Error * GetInputs (
            uint16 & value,
            bool viaSDO = false )
```

Get the present value of the general purpose input pins.

The input pin values are returned one per bit. The value of input pin 1 will be returned in bit 0 (1 if high, 0 if low), pin 2 will be in bit 1, etc.

Parameters

value	variable that will store the returned value
viaSDO	If true, an SDO will be used to read the input pins. If false (default), the most recent input value received from the amplifier via PDO will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.77 GetInputs32()

```
const Error * GetInputs32 (
            uint32 & value )
```

32-bit version to Get the present value of the general purpose input pins.

Note that only the lower 16 input pins are PDO mapped, so this function will always use an SDO access to read the full set of inputs. If the input of interest is one of the lower 16 pins, then the method Amp::GetInputs() is generally a much faster way to read the pin's state.

The input pin values are returned one per bit. The value of input pin 1 will be returned in bit 0 (1 if high, 0 if low), pin 2 will be in bit 1, etc.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.78 GetInputShapingFilter()

```
const Error * GetInputShapingFilter (
            InputShaper & f )
```

Get the coefficients used in the input shaping filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.79 GetIoConfig()

```
const Error * GetIoConfig (
            AmpIoCfg & cfg )
```

Read the amplifier's programmable I/O pin configuration and return it in the passed config structure.

Parameters

cfg	A structure that holds the configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.80 GetIOOPtions()

```
const Error * GetIOOPtions (
            int32 & value )
```

Get the IO Options.

See the Amp::SetIOOptions method for description.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.81 GetIoPullup()

```
const Error * GetIoPullup (
            uint16 & value )
```

Get the current state of the input pin pull up/down resisters.

Pull up/down resisters control how an undriven input pin will be interpreted by the amplifier. Depending on the model of amplifier being controlled, there may be zero or more groups of pull up/down resisters attached to some the input pins.

Each bit of this register is used to control one group of pull up/down resisters. Bit 0 controls group 0, etc.

Please refer to the amplifier data sheet for details on the number of groups of pull up/down resisters, and which input pins are included in each group.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.82 GetIoPullup32()

```
const Error * GetIoPullup32 (
            int32 & value )
```

32 Bit version of GetIoPullup.

This is useful on drives that support more then 16 pull up/down resistors. Please see Amp::GetIoPullup() for more details.

Parameters

value	variable that will store the returned value.

Returns

A pointer to an error object, or NULL on success

5.4.3.83 GetLinkage()

```
Linkage * GetLinkage (
            void  )
```

Return a pointer to the linkage that this amplifier is attached to.

WARNING - this function is dangerous as the pointer is not locked. Use Amp::GetLinkRef instead!

Returns

The linkage pointer, or NULL if the Amp is not attached to any linkage object.

5.4.3.84 GetLinkRef()

```
uint32 GetLinkRef (
            void  )
```

Return a reference to the linkage that this amplifier is attached to.

Returns

The linkage reference, or 0 if the Amp is not attached to any linkage object.

5.4.3.85 GetMicrostepRate()

```
const Error * GetMicrostepRate (
            int16 & rate )
```

Get the amplifier microstepping rate.

This parameter is only used in the diagnostic microstepping mode (AMPMODE_DIAG_USTEP).

Parameters

rate	The microstepping rate will be returned here (degrees / second).

Returns

A pointer to an error object, or NULL on success

5.4.3.86 GetMotorCurrent()

```
const Error * GetMotorCurrent (
            int16 & u,
            int16 & v )
```

Get the actual current values read directly from the amplifier's current sensors.

Note that if the motor wiring is being swapped in software, the U and V reading will be swapped.

Parameters

u	The U winding current will be returned here.
v	The V winding current will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.87 GetMtrInfo()

```
const Error * GetMtrInfo (
            MtrInfo & info )
```

Read the motor information structure from the amplifier.

Parameters

info	A structure that will be filled with the motor info

Returns

A pointer to an error object, or NULL on success

5.4.3.88 GetNetworkOptions()

```
const Error * GetNetworkOptions (
            NetworkOptions & cfg )
```

Set the Network Options configuration.

Parameters

cfg	Reference to the NetworkOptions structure where the configuration data will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.89 GetNetworkRef()

```
uint32 GetNetworkRef (
            void  )  [virtual]
```

Return a reference ID to the network that this node is attached to.

Returns

The reference ID or 0 if the node isn't attached to any network.

Reimplemented from Node.

5.4.3.90 GetOutputConfig() [1/6]

```
const Error * GetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG & cfg )
```

Get the output configuration for the specified pin.

Parameters

pin	The output pin to check.
cfg	The pin configuration value will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.91 GetOutputConfig() [2/6]

```
const Error * GetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG & cfg,
            uint16 & axis )
```

Get the output configuration for the specified pin.

Parameters

pin	The output pin to check.
cfg	The pin configuration value will be returned here.
axis	On multi-axis drives the axis number that this output pin is configured for will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.92 GetOutputConfig() [3/6]

```
const Error * GetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG & cfg,
            uint32 & mask )
```

Get the output pin configuration for the specified pin.

Parameters

pin	The output pin to check.
cfg	The pin configuration value will be returned here.
mask	The pin's status bit selection mask will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.93 GetOutputConfig() [4/6]

```
const Error * GetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG & cfg,
            uint32 & mask,
            uint16 & axis )
```

Get the output pin configuration for the specified pin.

Parameters

pin	The output pin to check.
cfg	The pin configuration value will be returned here.
mask	The pin's status bit selection mask will be returned here.
axis	On multi-axis drives the axis number that this output pin is configured for will be returned here

Returns

A pointer to an error object, or NULL on success

5.4.3.94 GetOutputConfig() [5/6]

```
const Error * GetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG & cfg,
            uint32 & param1,
            uint32 & param2 )
```

Get the output pin configuration for the specified pin.

This function supports output pin configurations that require two 32-bit parameters.

Parameters

pin	The output pin to check.
cfg	The pin configuration value will be returned here.
param1	The pin's first 32-bit parameter will be returned here
param2	The pin's second 32-bit parameter will be returned here

Returns

A pointer to an error object, or NULL on success

5.4.3.95 GetOutputConfig() [6/6]

```
const Error * GetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG & cfg,
            uint32 & param1,
            uint32 & param2,
            uint16 & axis )
```

Get the output pin configuration for the specified pin.

This function supports output pin configurations that require two 32-bit parameters.

Parameters

pin	The output pin to check.
cfg	The pin configuration value will be returned here.
param1	The pin's first 32-bit parameter will be returned here
param2	The pin's second 32-bit parameter will be returned here
axis	On multi-axis drives the axis number that this pin is configured for will be returned here

Returns

A pointer to an error object, or NULL on success

5.4.3.96 GetOutputs()

```
const Error * GetOutputs (
            uint16 & value )
```

Get the present value of the output pin control register.

This register shows the current state of all digital output pins. For each pin, the corresponding bit in the register will be 1 if the pin is active, and 0 if the pin is inactive. Bit 0 follows output pin 0, bit 1 follows pin 1, etc.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.97 GetPhaseAngle()

```
const Error * GetPhaseAngle (
            int16 & value )
```

Get the motor phase angle.

The phase angle describes the motor's electrical position with respect to it's windings. It's an internal parameter used by the amplifier to commutate a brushless motor.

The angle is returned in degrees.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.98 GetPhaseMode()

```
const Error * GetPhaseMode (
            AMP_PHASE_MODE & mode )
```

Get the current phasing mode configuration from the amplifier.

Parameters

mode	The mode information will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.99 GetPosCaptureCfg()

```
const Error * GetPosCaptureCfg (
            POS_CAPTURE_CFG & cfg )
```

Read the current configuration of the position capture mechanism.

Parameters

cfg	The position capture configuration value is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.100 GetPosCaptureStat()

```
const Error * GetPosCaptureStat (
            POS_CAPTURE_STAT & stat )
```

Read the current status of the position capture mechanism.

Parameters

stat	The position capture status value is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.101 GetPositionActual()

```
const Error * GetPositionActual (
            uunit & value )
```

Get the actual position used by the servo loop.

For dual encoder systems, this will be the load encoder position. To get the motor encoder position on such a system, use Amp::GetPositionMotor.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.102 GetPositionCommand()

```
const Error * GetPositionCommand (
            uunit & value )
```

Get the instantaneous commanded position.

This position is the command input to the servo loop. The commanded position is calculated by the trajectory generator and updated every servo cycle.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.103 GetPositionError()

```
const Error * GetPositionError (
            uunit & value )
```

Get the position error.

This is the difference between the instantaneous commanded position and the actual position.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.104 GetPositionErrorWindow()

```
const Error * GetPositionErrorWindow (
            uunit & value )
```

Get the position error window.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The position error window value will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.105 GetPositionLoad()

```
const Error * GetPositionLoad (
            uunit & value )
```

Get the load encoder position.

For single encoder systems, this value is NOT USED.

For dual encoder systems, this function returns the load encoder position and is identical to the value returned by Amp::GetPositionActual. This is also the passive load position when the load encoder is configured to be in passive mode.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.106 GetPositionMotor()

```
const Error * GetPositionMotor (
            uunit & value )
```

Get the actual motor position.

For single encoder systems, this value is identical to the value returned by Amp::GetPositionActual.

For dual encoder systems, this function returns the actual motor position and Amp::GetPositionActual may be used to get the load encoder position.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.107 GetPositionWarnWindow()

```
const Error * GetPositionWarnWindow (
            uunit & value )
```

Get the position warning window.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.108 GetPosLoopConfig()

```
const Error * GetPosLoopConfig (
            PosLoopConfig & cfg )
```

Get the configuration values of the amplifiers position loop.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

5.4.3.109 GetProfileAcc()

```
const Error * GetProfileAcc (
            uunit & value )
```

Get the profile acceleration value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.110 GetProfileConfig()

```
const Error * GetProfileConfig (
            ProfileConfig & cfg )
```

Load a structure with all parameters related to point-to-point moves.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.111 GetProfileDec()

```
const Error * GetProfileDec (
            uunit & value )
```

Get the profile deceleration value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.112 GetProfileJerk()

```
const Error * GetProfileJerk (
            uunit & value )
```

Get the currently programmed jerk limit for S-curve profiles.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The location in which the jerk value will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.113 GetProfileType()

```
const Error * GetProfileType (
            PROFILE_TYPE & type )
```

Get the currently selected motion profile type.

This profile type is used for point to point moves in which the amplifier calculates it's own trajectory.

Parameters

type	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.114 GetProfileVel()

```
const Error * GetProfileVel (
            uunit & value )
```

Get the profile velocity value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.115 GetPvtBuffFree()

```
const Error * GetPvtBuffFree (
            int16 & n )
```

Get the number of free positions in the PVT segment buffer.

Parameters

n	A reference used to return the number of free positions

Returns

A pointer to an error object, or NULL on success

5.4.3.116 GetPvtBuffStat()

```
const Error * GetPvtBuffStat (
            uint32 & stat )
```

Get the amplifier's PVT buffer status word.

Parameters

stat	A reference to a variable where the status will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.117 GetPvtSegID()

```
const Error * GetPvtSegID (
            uint16 & id )
```

Get the segment ID that the amplifier expects for the next PVT segment.

This starts at zero when the amp is reset, and is increased for every segment received.

Parameters

id	A reference to the variable where the ID will be written

Returns

A pointer to an error object, or NULL on success

5.4.3.118 GetPvtSegPos()

```
const Error * GetPvtSegPos (
            uunit & pos )
```

Get the starting position of the PVT segment currently active in the amplifier.

When running in PVT mode, this allows an approximation of the amplifier position to be retrieved without adding any additional overhead to the CANopen network.

The position returned by this function is only valid when running in PVT mode.

Parameters

pos	The position is returned here.

Returns

A pointer to an error object, or NULL on success.

5.4.3.119 GetPwmInConfig()

```
const Error * GetPwmInConfig (
            PwmInConfig & cfg )
```

Upload the amplifier's PWM input pin configuration.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.120 GetPwmMode()

```
const Error * GetPwmMode (
            AMP_PWM_MODE & mode )
```

Get the current PWM output mode configuration from the amplifier.

Parameters

mode	The mode information will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.121 GetQuickStop()

```
const Error * GetQuickStop (
            QUICK_STOP_MODE & mode )
```

Get the quick stop mode.

This mode defines what happens when a quick stop command is issued to the amplifier.

Parameters

mode	The mode will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.122 GetQuickStopDec()

```
const Error * GetQuickStopDec (
            uunit & value )
```

Get the quick stop deceleration value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.123 GetRefVoltage()

```
const Error * GetRefVoltage (
            int16 & value )
```

Get the analog reference input voltage.

If the amplifier has an analog reference input, it's value will be returned in millivolts.

Parameters

value	The value will be returned in this variable

Returns

A pointer to an error object, or NULL on success

5.4.3.124 GetRegenConfig()

```
const Error * GetRegenConfig (
            RegenConfig & cfg )
```

Upload the current configuration parameters for the power regeneration resister connected to the amplifier.

Note that not all amplifiers support a regen resister. Please see the amplifier datasheet to determine if this feature is available for the amplifier being used.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.125 GetServoLoopConfig()

```
const Error * GetServoLoopConfig (
            int32 & value )
```

Get the servo loop configuration.

This parameter allows various parts of the drive servo loops to be enabled/disabled.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.126 GetSettlingTime()

```
const Error * GetSettlingTime (
            uint16 & value )
```

Get the position window time value (milliseconds).

This timeout is used in conjunction with the position window value to identify when a motor has come to rest at the desired position.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.127 GetSettlingWindow()

```
const Error * GetSettlingWindow (
            uunit & value )
```

Get the position window value.

This window, along with the position window time value, is used to identify when the motor has come to rest at the desired position.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.128 GetSoftLimits()

```
const Error * GetSoftLimits (
            SoftPosLimit & cfg )
```

Upload the current software limit switch settings from the amplifier.

Parameters

cfg	The limit switch settings will be returned in this structure.

Returns

A pointer to an error object, or NULL on success.

5.4.3.129 GetSpecialFirmwareConfig()

```
const Error * GetSpecialFirmwareConfig (
            AmpConfig & cfg )
```

Reads the amplifier's special firmware.

These will use serial binary commands since there are no corresponding CAN objects.

Parameters

cfg	A structure that holds the configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.130 GetState()

```
NodeState GetState (
            void  )  [virtual]
```

Returns the present state of this node.

Note that this requires node guarding or heartbeats to be enabled.

Returns

The present node state.

Reimplemented from Node.

5.4.3.131 GetStatusWord()

```
const Error * GetStatusWord (
            uint16 & value )
```

Get the current value of the drive's status word.

The drive status word indicates the drives current state in it's internal state machine. The drives state in turn identifies if the drive is enabled / disabled, whether a fault is present on the drive, whether it is in motion, etc.

This status word is part of the CANopen device profile (DSP-402). It's used internally by the amplifier object.

Parameters

value	Returns the current status word value

Returns

A pointer to an error object, or NULL on success

5.4.3.132 GetTargetPos()

```
const Error * GetTargetPos (
            uunit & value )
```

Get the profile target position.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.133 GetTargetVel()

```
const Error * GetTargetVel (
            uunit & value )
```

Get the target velocity used in profile velocity mode.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.134 GetTorqueActual()

```
const Error * GetTorqueActual (
            int16 & value )
```

Get the actual torque being applied by the motor at the moment.

Parameters

value	The torque value is returned here. This is specified in thousandths of the motor rated torque (see Amp::SetTorqueTarget more more info).

Returns

A pointer to an error object, or NULL on success

5.4.3.135 GetTorqueDemand()

```
const Error * GetTorqueDemand (
            int16 & value )
```

Get the torque demand value.

This is the torque that the amplifier is attempting to apply at the moment.

Parameters

value	The torque value is returned here. This is specified in thousandths of the motor rated torque (see Amp::SetTorqueTarget more more info).

Returns

A pointer to an error object, or NULL on success

5.4.3.136 GetTorqueRated()

```
const Error * GetTorqueRated (
            int32 & value )
```

Get the motor rated torque parameter.

The motor's rated torque is the amount of torque that the motor can continuously output without damage.

Parameters

value	The motor's rated torque in 0.001 Nm units. For linear motors the units are 0.001 N.

Returns

A pointer to an error object, or NULL on success

5.4.3.137 GetTorqueSlope()

```
const Error * GetTorqueSlope (
            int32 & value )
```

Get the rate of change of torque for use in profile torque mode (AMPMODE_CAN_TORQUE).

Parameters

value	The rate of change specified in thousandths of the total rated torque per second.

Returns

A pointer to an error object, or NULL on success

5.4.3.138 GetTorqueTarget()

```
const Error * GetTorqueTarget (
            int16 & value )
```

Get the current target torque value.

This parameter is used in profile torque mode (AMPMODE_CAN_TORQUE) to specify the torque that should be applied by the motor.

Parameters

value	The torque value is returned here. This is specified in thousandths of the motor rated torque (see Amp::SetTorqueTarget more more info).

Returns

A pointer to an error object, or NULL on success

5.4.3.139 GetTraceChannel()

```
const Error * GetTraceChannel (
            uint8 ndx,
            AMP_TRACE_VAR & value )
```

Get the amplifier variable current selected on one of the trace channels.

Parameters

ndx	The trace channel to get
value	The trace variable assigned to this channel will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.140 GetTraceData()

```
const Error * GetTraceData (
            int32 * data,
            int32 & max )
```

Upload any trace data captured in the amplifier.

Trace data should only be uploaded when the traces are stopped. Uploading data while it is currently being collected in the amplifier can cause corrupt data to be uploaded.

The trace data is returned as an array of 32-bit integer values. If there are N currently active trace channels, and M samples of data have been collected, then a total of N x M integer values will be returned. In this case, the samples for channel n ($0 <= n < N$) will be located at postion $n + m*N$ for $0 <= m < M$.

Parameters

data	An array where the trace data will be returned.
max	On entry to this call, this parameter must hold the maximum number of 32-bit integer values to upload. On successful return this parameter will be filled with the total number of integers uploaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.141 GetTraceMaxChannel()

```
const Error * GetTraceMaxChannel (
              uint8 & max )
```

Return the maximum number of trace channels supported by the amplifier.

Parameters

max	The number of channels is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.142 GetTracePeriod()

```
const Error * GetTracePeriod (
              int16 & per )
```

Get the period of time between trace samples.

When the trace system is running, the amplifier will sample and store it's internal variables this often.

Note that this parameter specifies time in units of the amplifier's 'reference period'. See Amp::GetTraceRefPeriod for more information.

Parameters

per	The trace period is returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.143 GetTraceRefPeriod()

```
const Error * GetTraceRefPeriod (
            int32 & per )
```

Get the 'reference period' used with the amplifiers trace mechanism.

The amplifier internally samples it's trace channels at integer multiples of this time.

For example, if the amplifier's reference period is 100,000 nanoseconds, then setting the trace period to 12 would indicate that the amplifier should sample it's internal variables every 1.2 milliseconds.

Parameters

per	The reference period is returned here in units of nanoseconds.

Returns

A pointer to an error object, or NULL on success

5.4.3.144 GetTraceStatus()

```
const Error * GetTraceStatus (
            AMP_TRACE_STATUS & stat,
            int16 & samp,
            int16 & sampMax )
```

Get the current status of the amplifier's trace system.

Parameters

stat	Information on whether the trace is currently running is returned in this parameter.
samp	The total number of trace samples collected is returned here.
sampMax	The maximum number of trace samples that will fit in the internal buffer is returned here. This value will change depending on how many trace channels are active and which variables are selected.

Returns

A pointer to an error object, or NULL on success

5.4.3.145 GetTraceTrigger()

```
const Error * GetTraceTrigger (
            AMP_TRACE_TRIGGER & type,
            uint8 & chan,
            int32 & level,
            int16 & delay )
```

Get the current configuration of the amplifier's trace trigger.

See Amp::SetTraceTrigger for more information about the trigger.

Parameters

type	The type of trigger to be used.
chan	Which trace channel to trigger off of.
level	The trigger level
delay	The delay between the occurance of the trigger and the start of data collection.

Returns

A pointer to an error object, or NULL on success

5.4.3.146 GetTrackingWindows()

```
const Error * GetTrackingWindows (
            TrackingWindows & cfg )
```

Get the configuration values of the amplifiers position & velocity tracking windows.

This function allows all tracking window paramters to be read from the amplifier as a group.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

5.4.3.147 GetTrajectoryAcc()

```
const Error * GetTrajectoryAcc (
            uunit & value )
```

Get the instantaneous commanded acceleration passed out of the trajectory generator.

This acceleration is used by the position loop to calculate it's acceleration feed forward term.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.148 GetTrajectoryJrkAbort()

```
const Error * GetTrajectoryJrkAbort (
            uunit & value )
```

Get the jerk value used during trajectory aborts.

If this is 0, then the abort will be calculated without any jerk limits.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.149 GetTrajectoryVel()

```
const Error * GetTrajectoryVel (
            uunit & value )
```

Get the instantaneous commanded velocity passed out of the trajectory generator.

This velocity is used by the position loop to calculate it's velocity feed forward term.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.150 GetUstepConfig()

```
const Error * GetUstepConfig (
            UstepConfig & cfg )
```

Upload the current configuration parameters for the stepper.

Parameters

cfg	A structure where the configuration parameters will be returned.

Returns

A pointer to an error object, or NULL on success

5.4.3.151 GetVelLoopConfig()

```
const Error * GetVelLoopConfig (
            VelLoopConfig & cfg )
```

Get the configuration values of the amplifiers velocity loop.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

viDrain only in some firmware versions

5.4.3.152 GetVelocityActual()

```
const Error * GetVelocityActual (
            uunit & value )
```

Get the actual motor velocity.

The motor velocity is estimated by the amplifier based on the change in position seen at the encoder. For dual encoder systems, the load encoder velocity can be queried using the function Amp::GetVelocityLoad.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.153 GetVelocityCommand()

```
const Error * GetVelocityCommand (
            uunit & value )
```

Get the commanded velocity.

The commanded velocity is the velocity value that is passed to the velocity limiter, and from there to the velocity control loop. If the amplifier is in position mode (i.e. the position loop is active), then this velocity is the output of the position control loop.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.154 GetVelocityLimited()

```
const Error * GetVelocityLimited (
            uunit & value )
```

Get the limited velocity.

This velocity is the result of applying the velocity limiter to the commanded velocity (see GetVelocityCommand).

When the velocity loop is being driven by the position loop, the velocity limiter consists of a maximum velocity value only. When some other source is driving the velocity loop, the limiter also includes a maximum acceleration and deceleration value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.155 GetVelocityLoad()

```
const Error * GetVelocityLoad (
            uunit & value )
```

Get the load encoder velocity.

The load velocity is estimated by the amplifier based on the change in position seen at the load encoder. For dual encoder systems, the motor encoder velocity can be queried using the function Amp::GetVelocityActual.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.156 GetVelocityProgrammed()

```
const Error * GetVelocityProgrammed (
            uunit & vel )
```

Get the programmed velocity value.

This parameter is the velocity that the amplifier will attempt to maintain when set to the mode AMPMODE_PROG_VEL.

Parameters

vel	The velocity will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.157 GetVelocityWarnTime()

```
const Error * GetVelocityWarnTime (
            uint16 & value )
```

Get the position window time value (milliseconds).

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.158 GetVelocityWarnWindow()

```
const Error * GetVelocityWarnWindow (
            uunit & value )
```

Get the velocity warning window.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	variable that will store the returned value

Returns

A pointer to an error object, or NULL on success

5.4.3.159 GetVloopCommandFilter()

```
const Error * GetVloopCommandFilter (
            Filter & f )
```

Get the coefficients used in the velocity loop command filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.160 GetVloopOutputFilter()

```
const Error * GetVloopOutputFilter (
            Filter & f )
```

Get the coefficients used in the velocity loop output filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.161 GetVloopOutputFilter2()

```
const Error * GetVloopOutputFilter2 (
            Filter & f )
```

Get the coefficients used in the second velocity loop output filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.162 GetVloopOutputFilter3()

```
const Error * GetVloopOutputFilter3 (
            Filter & f )
```

Get the coefficients used in the third velocity loop output filter.

Parameters

f	A structure where the filter coefficients will be returned

Returns

A pointer to an error object, or NULL on success

5.4.3.163 GoHome() [1/2]

```
const Error * GoHome (
            void  )
```

Execute a home move.

The various homing parameters (method, velocity, etc) are assumed to have already be configured.

Returns

A pointer to an error object, or NULL on success.

5.4.3.164 GoHome() [2/2]

```
const Error * GoHome (
            HomeConfig & cfg )
```

Execute a home move.

The various homing parameters are passed in the HomeConfig structure.

This function simply programs all the homing parameters passed in the structure, then calls Amp::GoHome().

Parameters

cfg	The homing configuration parameter structure.

Returns

A pointer to an error object, or NULL on success.

5.4.3.165 HaltMove()

```
const Error * HaltMove (
            void  )
```

Halt the current move.

The exact type of halt can be programmed using the Amp::SetHaltMode function.

Note that the halt function is only available when running in one of the standard CAN amplifier modes. If doing low level velocity or current control, then moves must be stopped externally.

Returns

A pointer to an error object, or NULL on success.

5.4.3.166 HandleStateChange()

```
void HandleStateChange (
            NodeState from,
            NodeState to )  [protected], [virtual]
```

Handle an amplifier state change.

This method wakes up any task waiting on the move done semaphore in the event of a guard error. If this feature is desired, this method should be called from any class that over rides this method.

Parameters

from	Previous state of the Amp before the change
to	New state of the Amp

Reimplemented from Node.

5.4.3.167 Init() [1/2]

```
const Error * Init (
            Network & net,
            int16 nodeID )  [virtual]
```

Initialize the amplifier object using all default settings.

Parameters

net	Reference to the Network for this amp.
nodeID	a valid node ID for the amp. For CANopen, the node ID should range from 1 to 127. For EtherCAT, node ID's >= 0 identify the node by it's EtherCAT alias. Negative ID's identify the node by network position (-1 is the first node, -2 is the second, etc).

Returns

A pointer to an error object, or NULL on success.

Reimplemented from Node.

5.4.3.168 Init() [2/2]

```
const Error * Init (
            Network & net,
            int16 nodeID,
            AmpSettings & settings )
```

Initialize the amplifier object with custom amp settings.

Parameters

net	Reference to the Network for this amp.
nodeID	a valid node ID for the amp
settings	Amplifier settings to be used.

Returns

A pointer to an error object, or NULL on success.

5.4.3.169 InitSubAxis()

```
const Error * InitSubAxis (
            Amp & primary,
            int axis = 2 )
```

Initialize an Amp object for use with a secondary axis of a multi-axis EtherCAT amplifier.

For Copley EtherCAT multi-axis drives (such as the AE2, BE2, etc), the drive is configured as a single node on the EtherCAT network with multiple axes of motion residing at that node. To control such a drive using CML, use a seperate Amp object for each axis.

The first axis of such a drive is the primary axis and should be initialized using the normal Amp::Init() function call just like a single axis amp. For any additional axes, use the Amp::InitSubAxis() call and pass in a reference to the primary Amp object.

Note that multi-axis CANopen drives are normally configured as multiple distinct nodes on the CANopen network. Do not use this method to initialize nodes on a multi-axis CANopen drive, instead use the Amp::Init() method for each Amp object.

Parameters

primary	Reference to the Amp object created for the primary (first) axis of the multi-axis EtherCAT drive.
axis	The axis number of the axis to be initialized. The axis number passed should be greater then or equal to two. Axis one is the primary axis and should be initialized using the Amp::Init() method.

Returns

A pointer to an error object, or NULL on success.

5.4.3.170 IsHardwareEnabled()

```
bool IsHardwareEnabled (
            void  )
```

Return true if the amplifier's PWM outputs are currently enabled.

Returns

true if the amplifier's PWM outputs are currently enabled.

5.4.3.171 IsReferenced()

```
bool IsReferenced (
            void  )
```

Return true if the amplifier has been successfully referenced (homed).

When an amplifier is first powered up (or after a reset) it does not know the absolute position of the motor. Once the home routine has been successfully executed, the encoder zero location is known and the amplifier is considered referenced.

Once an amplifier has been referenced, it will not loose reference until it is reset, or until a new home routine is executed. During the execution of a home routine, the amplifier is considered to be unreferenced. If the home routine is completed successfully, the amplifier will then be referenced again.

Returns

true if the amplifier has been referenced. Return false if the amplifier has not been referenced if an error occurs reading this information from the amplifier.

5.4.3.172 IsSoftwareEnabled()

```
bool IsSoftwareEnabled (
            void  )
```

Return true if the amplifier is being enabled by software.

The amplifier outputs may still be disabled if this is true due to an error condition, etc.

Returns

true if the amplifier is enabled by software.

5.4.3.173 JrkLoad2User()

```
uunit JrkLoad2User (
            int32 jrk )  [virtual]
```

Convert a jerk value from internal amplifier units to user units.

Internal to the amplifier, all jerk values are stored in units of 100 encoder counts / second^3. If user units are not enabled in CML_Settings.h, then user units are the same as amplifier units, and this function has no effect.

If user units are enabled, then this function converts from amplifier units to user units (defined using Amp::SetCounts↩PerUnit).

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution.

Parameters

jrk	The jerk value in units of 100 encoder counts / second^3

Returns

The jerk in user units

5.4.3.174 JrkUser2Load()

```
int32 JrkUser2Load (
            uunit jrk )  [virtual]
```

Convert a jerk value from user units to internal amplifier units.

Internal to the amplifier, all jerk values are stored in units of 100 encoder counts / second ^ 3. If user units are not enabled in CML_Settings.h, then user units are the same as amplifier units, and this function has no effect.

If user units are enabled, then this function converts from user units (defined using Amp::SetCountsPerUnit) to these internal amplifier units.

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution.

Parameters

jrk	The jerk in user units

Returns

The jerk in 100 encoder counts / second ^ 3 units

5.4.3.175 LoadCCDFromFile()

```
const Error * LoadCCDFromFile (
            const char * name,
            Network & net )
```

Checks the network type and calls the correct function to write the ccd file to the amplifier.

Parameters

name	Name of the file to be passed.
net	Reference to the network object.

Returns

The most recent error code returned by this node.

5.4.3.176 LoadFromFile()

```
const Error * LoadFromFile (
            const char * name,
            int & line )
```

Load the specified amplifier data file.

This function presently supports loading *.ccx files created by the CME-2 program, version 3.1 and later.

Parameters

name	The name (and optionally path) of the file to load
line	If not NULL, the last line number read from the file is returned here. This is useful for finding file format errors.

Returns

A pointer to an error object, or NULL on success.

5.4.3.177 MoveAbs()

```
const Error* MoveAbs (
            uunit pos )  [inline]
```

Start an absolute point to point move to the specified position.

This is identical to calling Amp::DoMove(pos)

Parameters

pos	The position to move to

Returns

A pointer to an error object, or NULL on success

5.4.3.178 MoveRel()

```
const Error* MoveRel (
             uunit dist )  [inline]
```

Start a relative point to point move of the specified distance.

This is identical to calling Amp::DoMove(dist, true)

Parameters

dist	The distance to move

Returns

A pointer to an error object, or NULL on success

5.4.3.179 PosLoad2User()

```
uunit PosLoad2User (
             int32 pos )  [virtual]
```

Convert a position from internal amplifier units to user units.

Internal to the amplifier, all positions are stored in units of encoder counts. If user units are not enabled in CML_Settings.h, then user units are also in encoder counts and this function has no effect.

If user units are enabled, then this function converts from amplifier units to user units (defined using Amp::SetCountsPerUnit).

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution. To convert motor encoder positions, use Amp::PosMtr2User. On single encoder systems either of these functions can be used.

Parameters

pos	The position in encoder counts

Returns

The position in user units

5.4.3.180 PosMtr2User()

```
uunit PosMtr2User (
            int32 pos )  [virtual]
```

Convert a position from internal amplifier units to user units.

This function converts using motor encoder units on a dual encoder system. Load encoder positions can be converted using Amp::PosLoad2User.

Parameters

pos	The position in encoder counts

Returns

The position in user units

5.4.3.181 PosUser2Load()

```
int32 PosUser2Load (
            uunit pos )  [virtual]
```

Convert a position from user position units to internal amplifier units.

Internal to the amplifier, all positions are stored in units of encoder counts. If user units are not enabled in CML_Settings.h, then user units are also in encoder counts and this function has no effect.

If user units are enabled at compile time, then this function converts from user units (defined using Amp::SetCountsPerUnit) to these internal amplifier units.

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution. To convert motor encoder positions, use Amp::PosUser2Mtr. On single encoder systems either of these functions can be used.

Parameters

pos	The position in user units

Returns

The position in encoder counts

5.4.3.182 PosUser2Mtr()

```
int32 PosUser2Mtr (
            uunit pos )  [virtual]
```

Convert a position from user position units to internal amplifier units.

This function converts using motor encoder units on a dual encoder system. Load encoder positions can be converted using Amp::PosUser2Load.

Parameters

pos	The position in user units

Returns

The position in encoder counts

5.4.3.183 PvtBufferFlush()

```
const Error * PvtBufferFlush (
            bool viaSDO = true )  [protected]
```

Flush the amplifier's PVT trajectory buffer.

Flushing the buffer in this way will cause any running profile to be aborted.

Parameters

viaSDO	If true, use a SDO to download the message. If false, use a PDO. default is true.

Returns

An error object.

5.4.3.184 PvtBufferPop()

```
const Error * PvtBufferPop (
            uint16 n = 1,
            bool viaSDO = true )  [protected]
```

Pop the N most recently sent segments off the amplifier's PVT trajectory buffer.

If there are less then N segments on the buffer, then the buffer is cleared. Any profile running on the amplifier will continue to run (is not aborted) unless a buffer underflow occurs.

Parameters

n	The number of segments to pop off the buffer. Defaults to 1.
viaSDO	If true, use a SDO to download the message. If false, use a PDO. default is true.

Returns

An error object.

5.4.3.185 PvtClearErrors()

```
const Error * PvtClearErrors (
            uint8 mask,
            bool viaSDO = true )  [protected]
```

Clear the specified PVT buffer errors.

Parameters

mask	A bit mask representing which PVT buffer errors to clear.
viaSDO	If true, use a SDO to download the message. If false, use a PDO. default is true.

Returns

An error object.

5.4.3.186 PvtStatusUpdate()

```
void PvtStatusUpdate (
            uint32 status )
```

This is an internal function which should not be called by user code.

This function is used to handle the details of streaming out a trajectory that's too big to fit in the amplifier's PVT buffer all at once.

5.4.3.187 PvtWriteBuff() [1/2]

```
const Error * PvtWriteBuff (
            uint8 * buff,
            bool viaSDO = false )  [protected]
```

Write to the PVT buffer on the amp.

We use a PDO to do this on CANopen, but use an SDO access on EtherCAT.

Parameters

buff	Points to a buffer of formatted data to be sent. The buffer length must be at least 8 bytes.
viaSDO	Only used on CANopen. If true, download using an SDO.

Returns

A pointer to an error object, or NULL on success.

5.4.3.188 PvtWriteBuff() [2/2]

```
const Error * PvtWriteBuff (
            uint8 buff[][8],
            int ct,
            bool viaSDO = false )  [protected]
```

Write multiple PVT buffers to the drive.

Parameters

buff	Points to a buffer of formatted data to be sent
ct	Number of buffers to write;
viaSDO	Only used on CANopen. If true, download using an SDO.

Returns

A pointer to an error object, or NULL on success.

5.4.3.189 QuickStop()

```
const Error * QuickStop (
            void  )
```

Perform a 'quick stop' on the axis.

The exact meaning of a quick stop can be programmed using the Amp::SetQuickStop function. Regardless of the type of quick stop being performed, the amplifier will always end up disabled at the end of the quick stop. If disabling the amplifier is not desirable, then the Amp::HaltMove function should be used instead.

Note that the quick stop function is only available when running in one of the standard CAN amplifier modes. If doing low level velocity or current control, then moves must be stopped externally.

Returns

A pointer to an error object, or NULL on success.

5.4.3.190 ReInit()

```
const Error * ReInit (
            void  )
```

Re-initialize an amplifier.

This function simply calls Amp::Init using the same parameters that were initially passed.

Returns

A pointer to an error object, or NULL on success.

5.4.3.191 Reset()

```
const Error * Reset (
            void  )
```

Reset the amplifier object.

This function should be used for Amp objects instead of Node::ResetNode(). It resets the amplifier and re-initializes the amplifier object.

Returns

A pointer to an error object, or NULL on success.

5.4.3.192 SaveAmpConfig() [1/2]

```
const Error * SaveAmpConfig (
            AmpConfig & cfg )
```

Upload the passed amplifier configuration to the amplifier's workign memory, and then copy that working memory to flash.

Parameters

cfg	The structure which holds the new configuration.

Returns

A pointer to an error object, or NULL on success

5.4.3.193 SaveAmpConfig() [2/2]

```
const Error * SaveAmpConfig (
            void  )
```

Save all amplifier parameters to internal flash memory.

Flash memory is a type of non-volatile RAM which allows amplifier parameters to be saved between power cycles. When this function is called, any amplifier parameters that may be stored to flash will be copied from their working (RAM) locations to the stored (flash) locations.

For a list of those amplifier parameters which may be saved to flash memory, see the AmpConfig structure. Every member of that structure represents an amplifier parameter that may be saved to flash.

Returns

A pointer to an error object, or NULL on success

5.4.3.194 SendTrajectory()

```
const Error * SendTrajectory (
            Trajectory & trj,
            bool start = true )
```

Upload a PVT move trajectory to the amplifier and optionally start the move.

Parameters

trj	Reference to the trajectory that will be feed to the amp. A local pointer to this trajectory will be stored if the entire profile will not fit in the amplifiers on-board buffer. This pointer will be kept until the entire profile has been uploaded to the amp. It is therefore important to ensure that the trajectory object will remain valid (i.e. not be deallocated) until the amplifier object has called the Trajectory.Finish() method on it.
start	If true (the default), the profile will be started by this call. If false, the profile will be uploaded, but not started. Use true if this is a single axis move, false if this is part of a multi-axis move which needs to be synchronized.

Returns

An error object.

Clear the PVT segment cache

Make sure the trajectory object is ready to go

5.4.3.195 SetAlgoPhaseInit()

```
const Error * SetAlgoPhaseInit (
            AlgoPhaseInit & cfg )
```

Download a new configuration structure for the algorithmic phase init.

Parameters

cfg	A structure containing the configuration parameters to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.196 SetAmpConfig()

```
const Error * SetAmpConfig (
            AmpConfig & cfg )
```

Update an amplifier's configuration from the passed structure.

The AmpConfig structure holds all amplifier parameters that may be stored in the amplifier's non-volatile flash memory. This function may be used to update all of these parameters in a single call.

Note that this function updates the copies of these variables in working RAM, not directly in the amplifier flash memory. To copy these parameters to non-volatile memory, call Amp::SaveAmpConfig after updating them.

Parameters

cfg	The structure which holds the new configuration.

Returns

A pointer to an error object, or NULL on success

5.4.3.197 SetAmpMode()

```
const Error * SetAmpMode (
            AMP_MODE mode )
```

Set the amplifier mode of operation.

The mode of operation determines the top level control loop that will be controled (position, velocity, or current), and the source of that control (CANopen network, digital input pins, etc).

Parameters

mode	The mode of operation to be set

Returns

A pointer to an error object, or NULL on success.

5.4.3.198 SetAmpName()

```
const Error * SetAmpName (
            char * name )
```

Set the amplifier name stored in the amplifiers flash.

Parameters

name	The name of the drive to be set

Returns

A pointer to an error object, or NULL on success

5.4.3.199 SetAnalogCommandFilter()

```
const Error * SetAnalogCommandFilter (
            Filter & f )
```

Set new coefficients for the analog reference input filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.200 SetAnalogRefConfig()

```
const Error * SetAnalogRefConfig (
            AnalogRefConfig & cfg )
```

Configure the amplifier's analog reference input.

Note that some amplifier models do not support the analog reference.

Parameters

cfg	A structure holding the configuration to download

Returns

A pointer to an error object, or NULL on success

5.4.3.201 SetCammingConfig()

```
const Error * SetCammingConfig (
            CammingConfig & cfg )
```

Download a new configuration structure for Camming.

Parameters

cfg	A structure containing the configuration parameters to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.202 SetCanNetworkConfig()

```
const Error * SetCanNetworkConfig (
            CanNetworkConfig & cfg )
```

Set the CANopen node ID and bit rate configuration.

Note that the amplifier only uses this parameter at startup or after a reset.

Parameters

cfg	Structure holding the configuration to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.203 SetControlWord()

```
const Error * SetControlWord (
            uint16 value )  [virtual]
```

Set the amplifier's control word.

The control word is part of the CANopen device profile (DSP-402). It's used to enable/disable the amplifier, start moves, etc. This function is used internally by the Amp object.

Parameters

value	The control word value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.204 SetCountsPerUnit() [1/2]

```
const Error * SetCountsPerUnit (
            uunit cts )  [virtual]
```

Configure the user programmable units.

Unit conversions may be enabled or disabled at compile time through a setting in CML_Settings.h. If this feature is disabled, then all position, velocity, acceleration & jerk values are passed as 32-bit integers in the amplifier's native units:

- Position: Encoder counts
- Velocity: 0.1 encoder counts / second (i.e. 100 would be 10 counts/sec)
- Acceleration: 10 counts / second ^ 2 (i.e. 100 would be 1000 counts/sec^2)

- Jerk: 100 counts / second^3 (i.e. 100 would be 10000 counts/sec^3).

If unit conversions are enabled in CML_Settings.h, then these values are passed as double precision floating point values, and this function may be used to set a scaling factor for these units. The scaling factor is passed to this function as a number of encoder counts / user distance unit.

Velocity units are always equal to distance units / second. Likewise, acceleration and jerk units are distance units / second^2 and distance units / second^3.

For example, if the motor in question has a 1 micron encoder, then user units of meters, meters/sec, meters/sec^2, etc can be selected by passing a value of 1,000,000 to this function (i.e. the number of microns/meter).

When user units are enabled at compile time, the amplifier defaults to units of encoder counts, encoder counts / second, etc.

Parameters

cts	The number of encoder counts / user distance unit.

Returns

A pointer to an error object, or NULL on success. Note that if user units are disabled in CML_Settings.h, then this function will return an error.

5.4.3.205 SetCountsPerUnit() [2/2]

```
const Error * SetCountsPerUnit (
            uunit load,
            uunit mtr )  [virtual]
```

Configure the user programmable units for a dual encoder system.

This method provides the same feature as the single encoder version however it takes two scaling parameters; a load encoder scaler and a motor encoder scaler. These two values are used to scale amplifier parameters based on which encoder they refer to.

Parameters

load	The load encoder scaling factor. This gives the number of load encoder counts / user position unit.
mtr	The motor encoder scaling factor. This gives the number of motor encoder counts / user position unit.

Returns

A pointer to an error object, or NULL on success.

5.4.3.206 SetCrntLoopConfig()

```
const Error * SetCrntLoopConfig (
            CrntLoopConfig & cfg )
```

Update the amplifier's current loop configuration.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

5.4.3.207 SetCurrentProgrammed()

```
const Error * SetCurrentProgrammed (
            int16 crnt )
```

Set the programmed current value in 0.01 Amp units.

This parameter is only used when running in the mode AMPMODE_PROG_CRNT. The value programmed through this variable is the current that the amplifier will attempt to output.

Parameters

crnt	The current to output (0.01 Amp units).

Returns

A pointer to an error object, or NULL on success

5.4.3.208 SetDAConverterConfig()

```
const Error * SetDAConverterConfig (
            DAConfig & cfg )
```

Set the D/A converter configuration.

Parameters

cfg	A structure holding the D/A converter configuration.

Returns

A pointer to an error object, or NULL on success

5.4.3.209 SetEncoderErrorConfig()

```
const Error * SetEncoderErrorConfig (
            EncoderErrorConfig & cfg )
```

Configure the amplifier's encoder error configuration.

Parameters

cfg	A structure that holds the encoder error configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.210 SetFaultMask()

```
const Error * SetFaultMask (
            AMP_FAULT & value )
```

Set the amplifier's fault mask.

The fault mask identifies which conditions will be treated as latching faults by the amplifier. If such a condition occurs, the amplifier's output will be disabled immediately, and will not be enabled until the fault condition is cleared.

Parameters

value	A bit mask identifying which fault conditions to latch.

Returns

A pointer to an error object, or NULL on success

5.4.3.211 SetFuncGenConfig()

```
const Error * SetFuncGenConfig (
            FuncGenConfig & cfg )
```

Configure the amplifier's internal function generator.

Parameters

cfg	A structure holding the configuration to download

Returns

A pointer to an error object, or NULL on success

5.4.3.212 SetGainScheduling()

```
const Error * SetGainScheduling (
            GainScheduling & cfg )
```

Download a new configuration structure for Gain Scheduling.

Parameters

cfg	A structure containing the configuration parameters to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.213 SetHaltMode()

```
const Error * SetHaltMode (
            HALT_MODE mode )
```

Set the halt mode.

When the amplifier's halt command is issued (Amp::HaltMove) the amplifier will attempt to stop the move in progress using the method defined by it's halt mode.

Parameters

mode	The mode to set

Returns

A pointer to an error object, or NULL on success

5.4.3.214 SetHomeAccel()

```
const Error * SetHomeAccel (
            uunit value )
```

Set the home acceleration.

This acceleration value will be used for all moves that are part of the home routine.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.215 SetHomeConfig()

```
const Error * SetHomeConfig (
            HomeConfig & cfg )
```

Configure the amplifier's homing related parameters.

The passed structure contains all parameters related to performing a home routine.

Parameters

cfg	A structure holding the configuration parameters.

Returns

A pointer to an error object, or NULL on success

5.4.3.216 SetHomeCurrent()

```
const Error * SetHomeCurrent (
            int16 value )
```

Set the home current.

The home current value is only used when homing to a hard stop. This parameter is specified in units of 0.01 Amps (i.e. a value of 123 would be 1.23 Amps).

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.217 SetHomeDelay()

```
const Error * SetHomeDelay (
            int16 value )
```

Set the home delay.

The home delay value is only used when homing to a hard stop. This parameter is specified in units of milliseconds.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.218 SetHomeMethod()

```
const Error * SetHomeMethod (
            COPLEY_HOME_METHOD method,
            uint16 extended = 0 )
```

Set the method used for homing the drive.

Parameters

method	The home method to set
extended	If the 'method' parameter is set to CHM_EXTENDED, then this value will be written to the extended homing parameter on the amplifier. For any other homing method this parameter is ignored.

Returns

A pointer to an error object, or NULL on success

5.4.3.219 SetHomeOffset()

```
const Error * SetHomeOffset (
            uunit value )
```

Set the home offset value.

This offset is the difference between the location of the homing sensor (as defined by the homing method), and the actual zero position. Once the home location has been found, the amplifier will use this offset to determine where the zero position location is.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.220 SetHomeVelFast()

```
const Error * SetHomeVelFast (
            uunit value )
```

Set the home velocity used to move to a home switch.

This velocity will be used for any move in the home routine that can be done at relatively high speed. A second slower velocity can also be programed for the parts of the home routine that are speed sensitive.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.221 SetHomeVelSlow()

```
const Error * SetHomeVelSlow (
            uunit value )
```

Set the home velocity used to find a switch edge.

This velocity will be used for any move in the home routine which is speed sensitive. This typically is a move in which the edge of a sensor is being searched for.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.222 SetIloopCommandFilter()

```
const Error * SetIloopCommandFilter (
            Filter & f )
```

Set the coefficients used in the current loop input filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.223 SetIloopCommandFilter2()

```
const Error * SetIloopCommandFilter2 (
            Filter & f )
```

Set the coefficients used in the second current loop input filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.224 SetInputConfig()

```
const Error * SetInputConfig (
            int8 pin,
            INPUT_PIN_CONFIG cfg,
            uint16 axis = 0 )
```

Set the input pin configuration for the specified input pin.

Each of the amplifier input pins can be configured to perform some function. This method configures the specified input to perform the specified function.

Parameters

pin	The input pin to configure. Input pins are numbered starting from 0. Check the amplifier datasheet for the number of input pins available.
cfg	The input pin function to be assigned to this pin.
axis	The axis to apply this input configuration to. This only applies to multi-axis drives. Default is 0 (first axis/axis A).

Returns

A pointer to an error object, or NULL on success

5.4.3.225 SetInputDebounce()

```
const Error * SetInputDebounce (
            int8 pin,
            int16 value )
```

Set the input pin debounce time for the specified input pin.

Each of the amplifier input pins can be configured ignore transient states that last less then the debounce time. This function configures the debounce time for a specific pin.

Parameters

pin	The input pin to configure. Input pins are numbered starting from 0. Check the amplifier datasheet for the number of input pins available.
value	The debounce time to use (milliseconds)

Returns

A pointer to an error object, or NULL on success

5.4.3.226 SetInputShapingFilter()

```
const Error * SetInputShapingFilter (
            InputShaper & f )
```

Set new coefficients for the input shaping filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.227 SetIoConfig()

```
const Error * SetIoConfig (
            AmpIoCfg & cfg )
```

Configure the amplifier's programmable I/O pins using the values passed in the config structure.

The inputCt and outputCt values of the config structure should indicate the total number of input & output pins to configure. If the amplifier has more pins then these values indicate, the configuration of the remaining amplifier pins will not be changed.

Parameters

cfg	A structure that holds the configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.228 SetIOOptions()

```
const Error * SetIOOptions (
            int32 value )
```

Set the IO Options.

This parameter is used to configure the optional features of the general purpose IO. Bits 0-3 describe whether several IO pins are used as a serial interface for expanded IO features. 0 is normal IO, 1 is AEM/APM development board LEDs and address switches. 2 is LEDs wired the same as the developers kit board, but using separate red & green LEDs for network status.

Parameters

value	The new value for the IOOptions configuration

Returns

A pointer to an error object, or NULL on success

5.4.3.229 SetIoPullup()

```
const Error * SetIoPullup (
            uint16 value )
```

Set the current state of the input pin pull up/down resisters.

Pull up/down resisters control how an undriven input pin will be interpreted by the amplifier. Depending on the model of amplifier being controlled, there may be zero or more groups of pull up/down resisters attached to some the input pins.

Each bit of this register is used to control one group of pull up/down resisters. Bit 0 controls group 0, etc.

Please refer to the amplifier data sheet for details on the number of groups of pull up/down resisters, and which input pins are included in each group.

Parameters

value	The new value to write to the pull up/down control register.

Returns

A pointer to an error object, or NULL on success

5.4.3.230 SetIoPullup32()

```
const Error * SetIoPullup32 (
            int32 value )
```

32 Bit version of SetIoPullup.

This is useful on drives that support more then 16 pull up/down resistors. Please see Amp::SetIoPullup() for more details.

Parameters

value	The new value to write to the pull up/down control register.

Returns

A pointer to an error object, or NULL on success

5.4.3.231 SetMicrostepRate()

```
const Error * SetMicrostepRate (
            int16 rate )
```

Set the amplifier microstepping rate.

This parameter is only used in the diagnostic microstepping mode (AMPMODE_DIAG_USTEP).

Parameters

rate	The microstepping rate in degrees / second

Returns

A pointer to an error object, or NULL on success

5.4.3.232 SetMtrInfo()

```
const Error * SetMtrInfo (
            MtrInfo & info )
```

Update the amplifier's motor information.

Parameters

info	A structure that contains the motor info to be downloaded.

Returns

A pointer to an error object, or NULL on success

5.4.3.233 SetNetworkOptions()

```
const Error * SetNetworkOptions (
            NetworkOptions & cfg )
```

Set the Network Options configuration.

Parameters

cfg	Reference to the NetworkOptions structure containing the configuration data to be written to the amp.

Returns

A pointer to an error object, or NULL on success

5.4.3.234 SetOutputConfig()

```
const Error * SetOutputConfig (
            int8 pin,
            OUTPUT_PIN_CONFIG cfg,
            uint32 param1 = 0,
            uint32 param2 = 0,
            uint16 axis = 0 )
```

Set the output pin configuration for the specified pin.

Each of the amplifier output pins can be configured to perform some function. These functions break down into several basic modes:

- manual mode: In this mode, the output pin will be controlled through the CANopen network using the Amp::SetOutputs function. Output pins can be configured to be either active high or active low in this mode.
- Status word tracking: In this mode, the output pin is configured to track one or more bits of one of the amplifier's internal status words. A 32-bit mask is also supplied which identifies which bits are to be tracked. If any of the selected bits are set in the status word, the output pin will go active.
- Position trigger. In this mode the output pin will be configured to go active based on the position of the motor. In some cases the output will go active between two programmed positions. In other cases the output will be triggered by crossing a position and will stay active for a programmed duration.

Parameters

pin	The output pin to configure. Output pins are numbered starting from 0. Check the amplifier datasheet for the number of output pins available.
cfg	The pin function to be assigned to this pin.
param1	A 32-bit parameter used in conjunction with the output pin configuration to define the pin behavior. For most simple output pin modes this parameter is a bitmask that selects bits in a status register that the output should track. If the output pin is being configured for manual mode, then the mask is not used and does not need to be specified.
param2	A second 32-bit parameter used in a few output pin configurations.
axis	On multi-axis drives this is used to configure which axis this output pin is configured for.

Returns

A pointer to an error object, or NULL on success

5.4.3.235 SetOutputs()

```
const Error * SetOutputs (
            uint16 value )
```

Update the state of the manual output pins.

The passed value will be written to the output pin control register. Any of the output pins that have been configured as manual outputs will be updated based on the value of this register.

Bit 0 controls output pin 0, bit 1 sets output pin 1, etc.

Note that only those output pins that have been configured as manual outputs are effected by this command. Output pins that are configured to perform some other function (such as tracking bits in the event status register) are not effected. See Amp::SetOutputConfig for details on configuring the amplifier output pins.

Also note that this pin controls the active/inactive state of the outputs, not the high/low state. Each output pin can be individually configured as active high or active low. Setting a bit in the register to 1 sets the corresponding output pin active. Active high/low configuration is set using Amp::SetOutputConfig.

Parameters

value	The new value to write to the output pin control register.

Returns

A pointer to an error object, or NULL on success

5.4.3.236 SetPhaseMode()

```
const Error * SetPhaseMode (
            AMP_PHASE_MODE mode )
```

Set the phasing mode configuration for the amplifier.

Parameters

mode	The phasing mode to set

Returns

A pointer to an error object, or NULL on success

5.4.3.237 SetPosCaptureCfg()

```
const Error * SetPosCaptureCfg (
            POS_CAPTURE_CFG cfg )
```

Set the position capture configuration.

The position capture mechanism in the amplifier allows the motor position to be captured by some event. The position can be captured by a transition on the encoder index signal, or by a transition on a general purpose input pin which has been configured as a 'home' input.

Parameters

cfg	The position capture configuration value is passed here.

Returns

A pointer to an error object, or NULL on success

5.4.3.238 SetPositionActual()

```
const Error * SetPositionActual (
            uunit value )
```

Set the actual position.

On dual encoder systems, this will set the load encoder position. Amp::SetPositionMotor may be used to set the motor position on such systems.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The actual position of the motor.

Returns

A pointer to an error object, or NULL on success

5.4.3.239 SetPositionErrorWindow()

```
const Error * SetPositionErrorWindow (
            uunit value )
```

Set the position error window.

If the absolute value of the motor's position error ever exceeds this value, then a tracking error will occur.

A tracking error causes the amplifier to abort any move in progress, and attempt to bring the motor to a stop using it's velocity loop. The commanded velocity input to the velocity loop will be driven to zero, subject to the velocity loop acceleration and deceleration limits.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The value to use for the position error window.

Returns

A pointer to an error object, or NULL on success

5.4.3.240 SetPositionLoad()

```
const Error * SetPositionLoad (
            uunit value )
```

Set the load encoder position.

On single encoder systems, this value is NOT USED.

For dual encoder systems, this will set the load encoder position. This is also the passive load position when the load encoder is configured to be in passive mode.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The actual position of the motor.

Returns

A pointer to an error object, or NULL on success

5.4.3.241 SetPositionMotor()

```
const Error * SetPositionMotor (
            uunit value )
```

Set the actual motor position.

On dual encoder systems, this will set the motor encoder position. Amp::SetPositionActual may be used to set the load position on such systems.

Parameters

value	The actual position of the motor.

Returns

A pointer to an error object, or NULL on success

5.4.3.242 SetPositionWarnWindow()

```
const Error * SetPositionWarnWindow (
            uunit value )
```

Set the position warning window.

If the absolute value of the position error ever exceeds this value, then a tracking warning will result. A tracking warning causes a bit in the drives status to be set.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.243 SetPosLoopConfig()

```
const Error * SetPosLoopConfig (
            PosLoopConfig & cfg )
```

Update the amplifier's position loop configuration.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

5.4.3.244 SetProfileAcc()

```
const Error * SetProfileAcc (
            uunit value )
```

Set the profile acceleration value (i.e.

the acceleration that the motor will normally attain when starting the move).

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The value to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.245 SetProfileConfig()

```
const Error * SetProfileConfig (
            ProfileConfig & cfg )
```

Configure the amplifier's parameters related to point-to-point moves.

Parameters

cfg	A structure holding the configuration parameters.

Returns

A pointer to an error object, or NULL on success

5.4.3.246 SetProfileDec()

```
const Error * SetProfileDec (
            uunit value )
```

Set the profile deceleration value (i.e.

the acceleration that the motor will normally attain when ending the move).

Note that S-curve profiles don't use a separate deceleration value. For S-curve moves, the value programmed in SetProfileAcc is also used for the deceleration segment at the end of the move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The value to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.247 SetProfileJerk()

```
const Error * SetProfileJerk (
            uunit value )
```

Set the jerk limit used with S-curve profiles.

Jerk is the rate of change of acceleration.

Note that this value is only used with S-curve profiles. Trapezoidal profiles do not limit jerk (i.e. they allow instantaneous changes in acceleration).

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The jerk value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.248 SetProfileType()

```
const Error * SetProfileType (
            PROFILE_TYPE type )
```

Set the motion profile type.

The motion profile type is only used when running in 'position profile' mode. In this mode, the drive performs point to point moves using it's internal trajectory generator.

The motion profile type defines the type of trajectory profile that the drive will generate.

Parameters

type	The profile type to use

Returns

A pointer to an error object, or NULL on success

5.4.3.249 SetProfileVel()

```
const Error * SetProfileVel (
            uunit value )
```

Set the profile velocity value (i.e.

the velocity that the motor will normally attain during the move).

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The value to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.250 SetPvtInitialPos()

```
const Error * SetPvtInitialPos (
            int32 pos,
            bool viaSDO = true )  [protected]
```

Set the initial position for a PVT trajectory.

This function sends a full 32-bit position value which can be used to start a PVT move beyond the 24-bit limit of normal segments. It's normally used at the beginning of a PVT trajectory when the starting position is greater then 24-bits and the commanded position at the time of the move's start is not obvious.

Parameters

pos	The 32-bit initial position
viaSDO	If true, use a SDO to download the message. If false, use a PDO. default is true.

Returns

An error object

5.4.3.251 SetPwmInConfig()

```
const Error * SetPwmInConfig (
            PwmInConfig & cfg )
```

Configure the amplifier's PWM input pins.

Note that these settings are only used when the amplifier is controlled by it's PWM (or pulse/direction) input pins, i.e. not in CANopen mode.

Parameters

cfg	A structure holding the configuration to download

Returns

A pointer to an error object, or NULL on success

5.4.3.252 SetPwmMode()

```
const Error * SetPwmMode (
            AMP_PWM_MODE mode )
```

Set the PWM output mode configuration for the amplifier.

Parameters

mode	The PWM output mode to set

Returns

A pointer to an error object, or NULL on success

5.4.3.253 SetQuickStop()

```
const Error * SetQuickStop (
            QUICK_STOP_MODE mode )
```

Set the quick stop mode.

When the amplifier's quick stop command is issued (Amp::QuickStop), the amplifier will attempt to stop the move in progress using the method defined by it's quick stop mode.

Parameters

mode	The mode to set

Returns

A pointer to an error object, or NULL on success

5.4.3.254 SetQuickStopDec()

```
const Error * SetQuickStopDec (
            uunit value )
```

Set the quick stop deceleration value (i.e.

the acceleration that the motor will use when doing a quick stop).

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The value to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.255 SetRegenConfig()

```
const Error * SetRegenConfig (
            RegenConfig & cfg )
```

Download a new configuration structure for the power regeneration resister.

Note that not all amplifiers support a regen resister. Please see the amplifier datasheet to determine if this feature is available for the amplifier being used.

Parameters

cfg	A structure containing the configuration parameters to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.256 SetServoLoopConfig()

```
const Error * SetServoLoopConfig (
            int32 value )
```

Set the servo loop configuration.

This parameter allows various parts of the drive servo loops to be enabled/disabled.

Parameters

value	The servo config value to be used.

Returns

A pointer to an error object, or NULL on success

5.4.3.257 SetSettlingTime()

```
const Error * SetSettlingTime (
            uint16 value )
```

Set the position window time value (milliseconds).

The drive will be considered to be settled in position after a move when it's absolute position error value has been within the position window for an amount of time greater then the position window time value.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.258 SetSettlingWindow()

```
const Error * SetSettlingWindow (
            uunit value )
```

Set the position settling window.

The drive will be considered to be settled in position after a move when it's absolute position error value has been within the position window for an amount of time greater then the position window time value.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.259 SetSoftLimits()

```
const Error * SetSoftLimits (
            SoftPosLimit & cfg )
```

Set software limit switch settings.

The amplifier's software limit settings consist of a positive and negative absolute position. Any time the motors actual position is greater then the positive limit, or less then the negative limit, a limit event occurs. Software limit events are treated by the amplifer in the same way that physical limit switches are, no current will be output in the direction of the limit switch, and any running trajectory will be aborted.

Software limit switches are not used until the amplifier has been homed. Also, if the positive software limit is set to a value greater then or equal to the negative software limit, then the limits are disabled.

Parameters

cfg	The limit switch settings to use

Returns

A pointer to an error object, or NULL on success.

5.4.3.260 SetSpecialFirmwareConfig()

```
const Error * SetSpecialFirmwareConfig (
            AmpConfig & cfg )
```

Configure the amplifier's special firmware.

These will use serial binary commands since there are no corresponding CAN objects.

Parameters

cfg	A structure that holds the configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.261 SetTargetPos()

```
const Error * SetTargetPos (
            uunit value )
```

Set the profile target position (i.e.

the position to which the motor should move).

For relative moves, this function sets the distance to move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The position to move to

Returns

A pointer to an error object, or NULL on success

5.4.3.262 SetTargetVel()

```
const Error * SetTargetVel (
            uunit value )
```

Set the target velocity used in profile velocity mode.

This parameter is only used when the amplifier is set to profile velocity mode (AMPMODE_CAN_VELOCITY). When in this mode, this parameter defines the target velocity for motion.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The new target velocity.

Returns

A pointer to an error object, or NULL on success

5.4.3.263 SetTorqueRated()

```
const Error * SetTorqueRated (
            int32 value )
```

Set the motor rated torque parameter.

The motor's rated torque is the amount of torque that the motor can continuously output without damage.

Parameters

value	The motor's rated torque in 0.001 Nm units. For linear motors the units are 0.001 N.

Returns

A pointer to an error object, or NULL on success

5.4.3.264 SetTorqueSlope()

```
const Error * SetTorqueSlope (
            int32 value )
```

Set the rate of change of torque for use in profile torque mode (AMPMODE_CAN_TORQUE).

Setting this parameter to zero will cause the rate of change to be unlimited.

Parameters

value	The rate of change specified in thousandths of the total rated torque per second. For example, setting to 1000 would specify a slope of the full rated torque of the motor every second.

Returns

A pointer to an error object, or NULL on success

5.4.3.265 SetTorqueTarget()

```
const Error * SetTorqueTarget (
            int16 value )
```

Set the amplifier target torque value.

This parameter is used in profile torque mode (AMPMODE_CAN_TORQUE) to specify the desired target torque value. The actual torque commanded by the amplifier will ramp up/down to this value based on the programmed torque slope (see Amp::SetTorqueSlope).

The units used for this object are based on the CANopen DS402 specification. An applications note is available on Copley's web site which gives more information on exactly how to convert between these torque units and the commanded current in the drive. http://www.copleycontrols.com/Motion/pdf/Current-Scaling.pdf

Parameters

value	The torque value to be set. This is specified in thousandths of the motor rated torque.

Returns

A pointer to an error object, or NULL on success

5.4.3.266 SetTraceChannel()

```
const Error * SetTraceChannel (
            uint8 ndx,
            AMP_TRACE_VAR value )
```

Select an amplifier trace variable to be sampled.

Parameters

ndx	The trace channel that the variable will be assigned to.
value	The trace variable to sample.

Returns

A pointer to an error object, or NULL on success

5.4.3.267 SetTracePeriod()

```
const Error * SetTracePeriod (
            int16 per )
```

Set the period of time between trace samples.

When the trace system is running, the amplifier will sample and store it's internal variables this often.

Note that this parameter specifies time in units of the amplifier's 'reference period'. See Amp::GetTraceRefPeriod for more information.

Parameters

per	The trace period to be set.

Returns

A pointer to an error object, or NULL on success

5.4.3.268 SetTraceTrigger()

```
const Error * SetTraceTrigger (
              AMP_TRACE_TRIGGER type,
              uint8 chan = 0,
              int32 level = 0,
              int16 delay = 0 )
```

Configure the amplifier's trace trigger.

The trigger acts something like the trigger on an oscilloscope. It allows some event to be specified which will cause the trace subsystem to start collecting data. Most trigger types watch one of the trace channels and constantly compare it's value to a level. The type of comparison made will depend on the type of trigger. For example, the trace can be triggered on the rising edge of a signal, on the falling edge, etc.

The trigger also allows a delay value to be specified. Trace data will start to be collected N trace periods after the trigger, where N is the delay value. The delay can also be negative, in which case the data will start to be collected before the trigger event.

Parameters

type	The trigger type.
chan	The trace channel to watch. This parameter defaults to 0 if not specified.
level	The trigger level. This parameter defaults to 0 if not specified.
delay	The trigger delay in trace sample periods. Defaults to 0 if not specified.

Returns

A pointer to an error object, or NULL on success

5.4.3.269 SetTrackingWindows()

```
const Error * SetTrackingWindows (
              TrackingWindows & cfg )
```

Update the amplifier's tracking window configuration.

This function allows all tracking window parameters to be configured with one function call.

Parameters

cfg	A structure that holds the configuration settings.

Returns

A pointer to an error object, or NULL on success

5.4.3.270 SetTrajectoryJrkAbort()

```
const Error * SetTrajectoryJrkAbort (
            uunit value )
```

Set the jerk value used during trajectory aborts.

If this is 0, then the abort will be calculated without any jerk limits.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	The jerk value to be used.

Returns

A pointer to an error object, or NULL on success

5.4.3.271 SetupMove() [1/3]

```
const Error * SetupMove (
            ProfileConfigTrap & cfg )
```

Setup a point to point move, but do not start it.

The move may be subsequently started using Amp::StartMove().

The move will use the trapezoidal profile mode, and all parameters will be programmed based on the values passed in the cfg structure.

Parameters

cfg	A structure holding all the move configuration parameters.

Returns

A pointer to an error object, or NULL on success.

5.4.3.272 SetupMove() [2/3]

```
const Error * SetupMove (
            ProfileConfigScurve & cfg )
```

Setup a point to point move, but do not start it.

The move may be subsequently started using Amp::StartMove().

The move will use the S-curve profile mode, and all parameters will be programmed based on the values passed in the cfg structure.

Parameters

cfg	A structure holding all the move configuration parameters.

Returns

A pointer to an error object, or NULL on success.

5.4.3.273 SetupMove() [3/3]

```
const Error * SetupMove (
            ProfileConfigVel & cfg )
```

Setup a point to point move, but do not start it.

The move may be subsequently started using Amp::StartMove().

The move will use the velocity profile mode, and all parameters will be programmed based on the values passed in the cfg structure.

Parameters

cfg	A structure holding all the move configuration parameters.

Returns

A pointer to an error object, or NULL on success.

5.4.3.274 SetUstepConfig()

```
const Error * SetUstepConfig (
            UstepConfig & cfg )
```

Download a new configuration structure for the microstepper.

Parameters

cfg	A structure containing the configuration parameters to set.

Returns

A pointer to an error object, or NULL on success

5.4.3.275 SetVelLoopConfig()

```
const Error * SetVelLoopConfig (
            VelLoopConfig & cfg )
```

Update the amplifier's velocity loop configuration.

Parameters

cfg	A structure that will be filled with the config info.

Returns

A pointer to an error object, or NULL on success

viDrain only in some firmware versions

5.4.3.276 SetVelocityProgrammed()

```
const Error * SetVelocityProgrammed (
            uunit vel )
```

Set the programmed velocity value.

This parameter is only used when running in the mode AMPMODE_PROG_VEL. The value programmed through this variable is the velocity that the amplifier will attempt to output.

Parameters

vel	The velocity to output.

Returns

A pointer to an error object, or NULL on success

5.4.3.277 SetVelocityWarnTime()

```
const Error * SetVelocityWarnTime (
            uint16 value )
```

Set the velocity warning window time value (milliseconds).

If the velocity error exceeds the velocity warning window, then a bit will be set in the amplifier status word. This bit will not be cleared until the velocity error has been within the warning window for at least this long.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.278 SetVelocityWarnWindow()

```
const Error * SetVelocityWarnWindow (
            uunit value )
```

Set the velocity warning window.

If the absolute value of the velocity error exceeds this value, then a velocity warning will result. A velocity warning causes a bit in the drives status to be set.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

Parameters

value	the value to set

Returns

A pointer to an error object, or NULL on success

5.4.3.279 SetVloopCommandFilter()

```
const Error * SetVloopCommandFilter (
            Filter & f )
```

Set new coefficients for the velocity loop command filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.280 SetVloopOutputFilter()

```
const Error * SetVloopOutputFilter (
            Filter & f )
```

Set new coefficients for the velocity loop output filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.281 SetVloopOutputFilter2()

```
const Error * SetVloopOutputFilter2 (
            Filter & f )
```

Set the coefficients used in the second velocity loop output filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.282 SetVloopOutputFilter3()

```
const Error * SetVloopOutputFilter3 (
            Filter & f )
```

Set the coefficients used in the third velocity loop output filter.

Parameters

f	A structure holding the filter coefficients

Returns

A pointer to an error object, or NULL on success

5.4.3.283 StartMove()

```
const Error * StartMove (
            bool relative = false )
```

Start the move that's already been programmed.

This function is primarily intended for internal use, and is called by DoMove and SendTrajectory. Note that the amplifier mode must have already been setup when this function is called. The mode should be either AMPMODE_CAN_PRO↩ FILE, or AMPMODE_CAN_PVT. This function is not used to start a homing move.

Parameters

relative	If true, start a relative move. If false, start an absolute move. Note that this is only used with point to point moves, interpolated moves should always set relative to false.

Returns

A pointer to an error object, or NULL on success.

5.4.3.284 StartPVT()

```
const Error * StartPVT (
            void  )
```

Start a PVT move that has already been uploaded.

Returns

A pointer to an error object, or NULL on success.

5.4.3.285 TraceStart()

```
const Error * TraceStart (
            void  )
```

Start collecting trace data on the amplifier.

The trace will automatically stop once the amplifier's internal trace buffer fills up.

Returns

A pointer to an error object, or NULL on success

5.4.3.286 TraceStop()

```
const Error * TraceStop (
            void  )
```

Stop collecting trace data on the amplifier.

Returns

A pointer to an error object, or NULL on success

5.4.3.287 UpdateEvents()

```
const Error * UpdateEvents (
            uint16 stat,
            uint32 events,
            uint16 inputs )  [virtual]
```

Update the amplifier's event map based on the status information received by a status PDO.

This function is intended for internal use and shouldn't generally be called by user code.

Parameters

stat	The CANopen status word
events	The Event status word
inputs	The current state of the first 16 input pins

Returns

Null on success, or an error object on failure

5.4.3.288 Upld16() [1/2]

```
const Error * Upld16 (
            int16 index,
            int16 sub,
            uint16 & data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.289 Upld16() [2/2]

```
const Error * Upld16 (
            int16 index,
            int16 sub,
            int16 & data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.290 Upld32() [1/2]

```
const Error * Upld32 (
            int16 index,
            int16 sub,
            uint32 & data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.291 Upld32() [2/2]

```
const Error * Upld32 (
            int16 index,
            int16 sub,
            int32 & data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.292 Upld8() [1/2]

```
const Error * Upld8 (
            int16 index,
            int16 sub,
            uint8 & data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.293 Upld8() [2/2]

```
const Error * Upld8 (
            int16 index,
            int16 sub,
            int8 & data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.294 UpldString()

```
const Error * UpldString (
            int16 index,
            int16 sub,
            int32 & len,
            char * data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
len	Holds the size of the buffer on entry, and the length of the downloaded data on return.
data	The uploaded data will be returned here.

Returns

A pointer to an error object, or NULL on success

5.4.3.295 Upload()

```
const Error * Upload (
            int16 index,
            int16 sub,
            int32 & size,
            byte * data )
```

Upload data from an object in this Amps object dictionary.

The object number is adjusted based on the axis number if necessary.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
size	On entry, this gives the maximum number of bytes of data to be uploaded. On successful return, it gives the actual number of bytes received.
data	A character array which will store the uploaded data.

Returns

A pointer to an error object, or NULL on success

5.4.3.296 VelLoad2User()

```
uunit VelLoad2User (
            int32 vel )  [virtual]
```

Convert a velocity from internal amplifier units to user units.

Internal to the amplifier, all velocities are stored in units of 0.1 encoder counts / second. If user units are not enabled in CML_Settings.h, then user units are the same as amplifier units, and this function has no effect.

If user units are enabled, then this function converts from amplifier units to user units (defined using Amp::SetCounts↩PerUnit).

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution. To convert motor encoder velocities, use Amp::VelMtr2User. On single encoder systems either of these functions can be used.

Parameters

vel	The velocity in 0.1 encoder counts / second

Returns

The velocity in user units

5.4.3.297 VelMtr2User()

```
uunit VelMtr2User (
            int32 vel )  [virtual]
```

Convert a velocity from internal amplifier units to user units.

This function converts using motor encoder units on a dual encoder system. Load encoder velcities can be converted using Amp::VelLoad2User.

Parameters

vel	The velocity in 0.1 encoder counts / second

Returns

The velocity in user units

5.4.3.298 VelUser2Load()

```
int32 VelUser2Load (
            uunit vel )  [virtual]
```

Convert a velocity from user units to internal amplifier units.

Internal to the amplifier, all velocities are stored in units of 0.1 encoder counts / second. If user units are not enabled in CML_Settings.h, then user units are the same as amplifier units, and this function has no effect.

If user units are enabled, then this function converts from user units (defined using Amp::SetCountsPerUnit) to these internal amplifier units.

For dual encoder systems the unit conversion used by this function is based on the load encoder resolution. To convert motor encoder velocities, use Amp::VelUser2Mtr. On single encoder systems either of these functions can be used.

Parameters

vel	The velocity in user units

Returns

The velocity in 0.1 encoder counts / second

5.4.3.299 VelUser2Mtr()

```
int32 VelUser2Mtr (
            uunit vel )  [virtual]
```

Convert a velocity from user units to internal amplifier units.

This function converts using motor encoder units on a dual encoder system. Load encoder velocities can be converted using Amp::VelUser2Load.

Parameters

vel	The velocity in user units

Returns

The velocity in 0.1 encoder counts / second

5.4.3.300 WaitEvent() [1/2]

```
const Error * WaitEvent (
            Event & e,
            Timeout timeout = -1 )
```

Wait for an amplifier event condition.

This function can be used to wait on any generic event associated with the amplifier.

Parameters

e	The event to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever (default).

Returns

A pointer to an error object, or NULL on success.

5.4.3.301 WaitEvent() [2/2]

```
const Error * WaitEvent (
            Event & eventObj,
            Timeout timeoutObj,
            AMP_EVENT & eventMatch )
```

Wait for an amplifier event condition.

This function can be used to wait on any generic event associated with the amplifier.

Parameters

eventObj	The event to wait on.
timeoutObj	The timeout for the wait (milliseconds). If < 0, then wait forever.
eventMatch	Returns the matching event condition.

Returns

A pointer to an error object, or NULL on success.

5.4.3.302 WaitHomeDone()

```
const Error * WaitHomeDone (
            Timeout timeout = -1 )
```

Wait for the currently running homing move to finish, or for an error to occur.

This is similar to the Amp::WaitMoveDone method, except it does some additional checks after the move finishes to ensure that the homing operation was successful. This function will fail immediately if the amp is not currently in homing mode.

Parameters

timeout	The maximum time to wait (milliseconds)

Returns

A pointer to an error object, or NULL on success.

5.4.3.303 WaitInputEvent()

```
const Error * WaitInputEvent (
            Event & e,
            Timeout timeout,
            uint32 & match )
```

Wait on the amplifier's general purpose input pins.

The amplifier object maintains an EventMap object which reflects the state of the amplifier's general purpose input pins. Each bit of this EventMap corresponds to one input pin; bit 0 for input 0, bit 1 for input 1, etc. The bit in the event map is set when the corresponding input pin is high, and cleared when the input pin is low.

This function provides a very flexible method for waiting on a particular state on the input pins. Event objects may be created to define a specific state of one or more pins, and these objects may be used in conjunction with this function to wait for that state to occur.

In addition to this function, two simpler functions are also provided. These functions (WaitInputHigh and WaitInput↩ Low) allow the user to wait on one or more input pins to go high or low respectively. Internally, these function call WaitInputEvent for their implementation.

Parameters

e	An Event object describing the input pin state to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever.
match	On success, the state of the input pins which caused the match to occur will be returned here.

Returns

A pointer to an error object, or NULL on success.

5.4.3.304 WaitInputHigh()

```
const Error * WaitInputHigh (
              uint32 inputs,
              Timeout timeout = -1 )
```

Wait for any of the specified general purpose input pins to be set.

The inputs parameter specifies which input(s) to wait on using a bit mask. Bit 0 should be set for input 0, bit 1 for input 1, etc. The function will return when any of the specified input pins goes high.

Parameters

inputs	Specifies which input pin(s) to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever. If not specified, the timeout defaults to -1

Returns

A pointer to an error object, or NULL on success.

5.4.3.305 WaitInputLow()

```
const Error * WaitInputLow (
              uint32 inputs,
              Timeout timeout = -1 )
```

Wait for any of the specified general purpose input pins to be lowered.

The inputs parameter specifies which input(s) to wait on using a bit mask. Bit 0 should be set for input 0, bit 1 for input 1, etc. The function will return when any of the specified input pins goes low.

Parameters

inputs	Specifies which input pin(s) to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever. If not specified, the timeout defaults to -1

Returns

A pointer to an error object, or NULL on success.

5.4.3.306 WaitMoveDone()

```
const Error * WaitMoveDone (
            Timeout timeout = -1 )
```

Wait for the currently running move to finish, or for an error to occur.

Parameters

timeout	The maximum time to wait (milliseconds)

Returns

A pointer to an error object, or NULL on success.

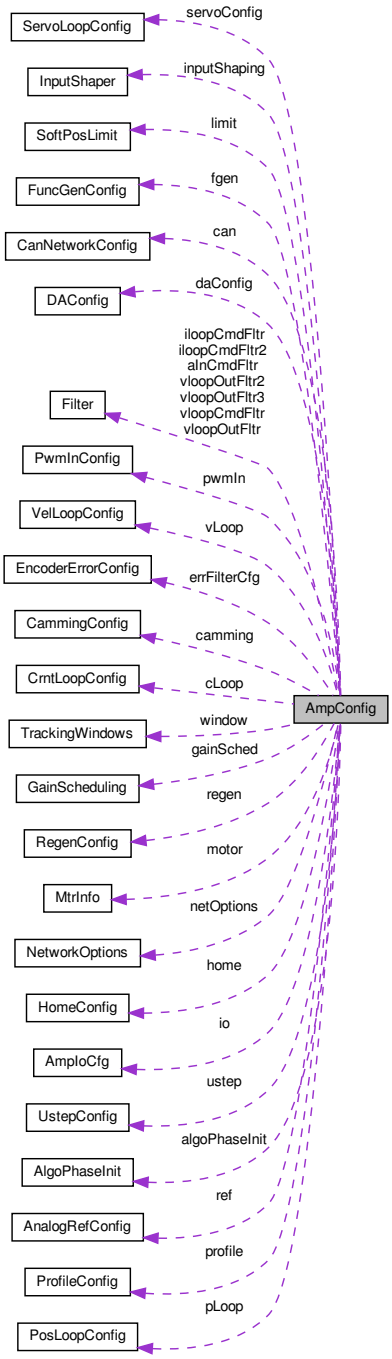
The documentation for this class was generated from the following files:

- CML_Amp.h
- Amp.cpp
- AmpFile.cpp
- AmpParam.cpp
- AmpPDO.cpp
- AmpPVT.cpp
- AmpStruct.cpp
- AmpUnits.cpp
- AmpVersion.cpp

5.5 AmpConfig Struct Reference

Amplifier configuration structure.

Collaboration diagram for AmpConfig:



Public Member Functions

- AmpConfig ()

  Default constructor. Simply sets all members to zero.

Public Attributes

- char name [COPLEY_MAX_STRING]

  Amplifier axis name.
- AMP_MODE controlMode

  Amplifier default mode of operation.
- AMP_PHASE_MODE phaseMode

  Amplifier phasing mode.
- AMP_PWM_MODE pwmMode

  PWM output mode.
- char CME_Config [COPLEY_MAX_STRING]

  String used by CME to save state information This string is reserved and should not be modified.
- AMP_FAULT faultMask

  Amplifier fault mask register.
- uunit progVel

  Programmed velocity value.
- int16 progCrnt

  Programmed current value (0.01 amp units).
- uint32 options

  Amplifier options.
- int16 stepRate

  Diagnostic microstepping rate.
- uint16 capCtrl

  Index capture configuration.
- uint32 trjJrkAbort

  Trajectory abort jerk.
- EVENT_STATUS limitBitMask

  One bit of a standard CANopen status word is user programmable using this setting.
- uint16 encoderOutCfg

  Some amplifier models provide a secondary encoder connected which can be configured as either an input or output.
- CanNetworkConfig can

  CANopen network configuration.
- PosLoopConfig pLoop

  Position loop configuration.
- VelLoopConfig vLoop

  Velocity loop configuration.
- CrntLoopConfig cLoop

  Current loop configuration.
- MtrInfo motor

  Motor information.
- TrackingWindows window

  Tracking window settings.
- SoftPosLimit limit

  Software position limits.
- AmpIoCfg io

  General purpose I/O pin configuration.
- HomeConfig home

Homing mode configuration.

- ProfileConfig profile

  Trajectory profile settings.
- AnalogRefConfig ref

  Analog reference input settings.
- PwmInConfig pwmIn

  PWM input configuration.
- FuncGenConfig fgen

  Internal function generator settings.
- RegenConfig regen

  Regeneration resister configuration.
- Filter vloopOutFltr

  Velocity loop output filter settings.
- Filter vloopCmdFltr

  Velocity loop command filter settings.
- Filter aInCmdFltr

  Analog command filter settings.
- InputShaper inputShaping

  Input shaping filter.
- UstepConfig ustep

  Stepper configuration.
- AlgoPhaseInit algoPhaseInit

  Algorithmic Phase Initialization.
- CammingConfig camming

  Camming Configuration.
- GainScheduling gainSched

  Gain Scheduling Configuration.
- NetworkOptions netOptions

  Can Bus network support.
- DAConfig daConfig

  D/A converter configuration.
- ServoLoopConfig servoConfig

  Configure various parts of the amps servo loops.
- EncoderErrorConfig errFilterCfg

  Encoder error filter configuration.

5.5.1 Detailed Description

Amplifier configuration structure.

This structure contains all user configurable parameters used by an amplifier which may be stored in non-volatile memory.

5.5.2 Member Data Documentation

5.5.2.1 capCtrl

```
uint16 capCtrl
```

Index capture configuration.

This parameter is not normally used in CANopen mode and is included here for completness only.

5.5.2.2 CME_Config

```
char CME_Config[COPLEY_MAX_STRING]
```

String used by CME to save state information This string is reserved and should not be modified.

5.5.2.3 encoderOutCfg

```
uint16 encoderOutCfg
```

Some amplifier models provide a secondary encoder connected which can be configured as either an input or output.

This parameter is used to configure this hardware.

5.5.2.4 limitBitMask

```
EVENT_STATUS limitBitMask
```

One bit of a standard CANopen status word is user programmable using this setting.

This feature is not used by CML and is only included here for completness.

5.5.2.5 options

```
uint32 options
```

Amplifier options.

This parameter is reserved for future use.

5.5.2.6 phaseMode

```
AMP_PHASE_MODE phaseMode
```

Amplifier phasing mode.

This parameter controls the type of commutation used by the amplifer.

5.5.2.7 progCrnt

```
int16 progCrnt
```

Programmed current value (0.01 amp units).

This parameter is only used in Amp mode AMPMODE_PROG_CRNT.

5.5.2.8 progVel

```
uunit progVel
```

Programmed velocitiy value.

This parameter is only used in Amp mode AMPMODE_PROG_VEL.

5.5.2.9 pwmMode

```
AMP_PWM_MODE pwmMode
```

PWM output mode.

This parameter can be used to configure the pwm output section of the amplifier.

5.5.2.10 stepRate

```
int16 stepRate
```

Diagnostic microstepping rate.

This parameter gives the microstep rate (degrees / second) for use in a special diagnostic microstepping mode. The parameter is not used in normal CANopen modes, and is only include here for completness.

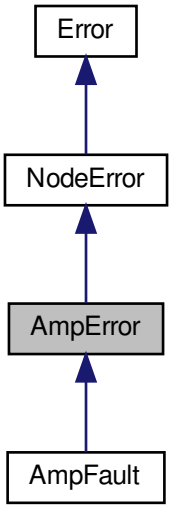
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

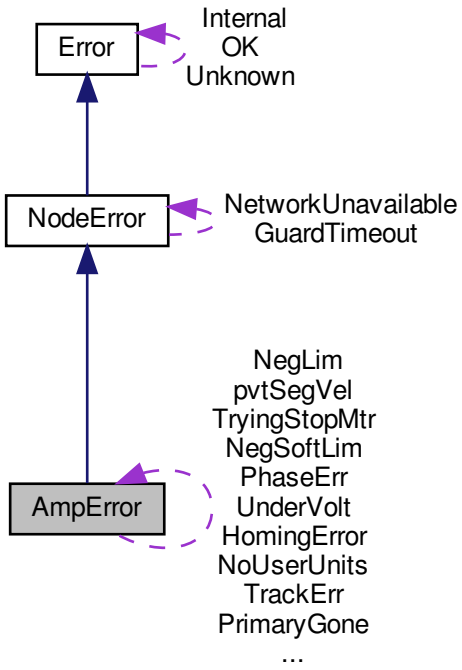
5.6 AmpError Class Reference

This class represents error conditions that can occur in the Copley Amplifier object.

Inheritance diagram for AmpError:



Collaboration diagram for AmpError:



Static Public Member Functions

- static const AmpError * DecodeStatus (EVENT_STATUS stat)

  Decode the passed event status word and return an appropriate error object.

Static Public Attributes

- static const AmpError Fault

  Latching fault is active.
- static const AmpError ShortCircuit

  Short circuit detected.
- static const AmpError AmpTemp

  Amplifier over temp.
- static const AmpError OverVolt

  Amplifier over voltage.
- static const AmpError UnderVolt

  Amplifier under voltage.
- static const AmpError MotorTemp

Motor over temp.

- static const AmpError EncoderPower

  Encoder power error.
- static const AmpError PhaseErr

  Motor phasing error.
- static const AmpError TrackErr

  Position tracking error.
- static const AmpError NodeState

  The drive's state is inappropriate for the requested operation.
- static const AmpError pvtSegPos

  PVT segment position value over 24-bits.
- static const AmpError pvtSegVel

  PVT segment velocity value too large.
- static const AmpError pvtBufferFull

  PVT trajectory buffer full.
- static const AmpError badDeviceID

  Unknown device identity.
- static const AmpError badHomeParam

  Bad parameter specified to home command.
- static const AmpError badMoveParam

  Bad parameter specified to move command.
- static const AmpError InMotion

  Amplifier is currently in motion.
- static const AmpError GuardError

  The amplifier's heartbeat message timed out.
- static const AmpError PosLim

  Positive limit switch is active.
- static const AmpError NegLim

  Negative limit switch is active.
- static const AmpError PosSoftLim

  Positive software limit is active.
- static const AmpError NegSoftLim

  Negative software limit is active.
- static const AmpError TrackWarn

  Position tracking warning.
- static const AmpError Unknown

  An error occurred, but went away before it could be decoded.
- static const AmpError Reset

  The amplifier has been reset.
- static const AmpError Disabled

  The amplifier is disabled.
- static const AmpError QuickStopMode

  The amplifier is doing a quick stop.
- static const AmpError NoUserUnits

  User units are not available (see CML_Settings.h)
- static const AmpError Abort

  Last trajectory aborted.

- static const AmpError pvtPosUnavail

  The PVT segment position is not available.
- static const AmpError VelWin

  Velocity tracking window exceeded.
- static const AmpError PhaseInit

  Amplifier is currently performing a phase initialization.
- static const AmpError NotHoming

  The amplifier is not currently configured for homing mode.
- static const AmpError HomingError

  The amplifier did not complete the homing method successfully.
- static const AmpError BadAxis

  Illegal axis number specified.
- static const AmpError PrimaryGone

  Primary axis Amp object no longer available.
- static const AmpError NotInit

  Amp object not properly initialized.
- static const AmpError TryingStopMtr

  The amplifier is trying to stop the motor.

Protected Member Functions

- AmpError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.6.1 Detailed Description

This class represents error conditions that can occur in the Copley Amplifier object.

5.6.2 Member Function Documentation

5.6.2.1 DecodeStatus()

```
const AmpError * DecodeStatus (
            EVENT_STATUS stat )  [static]
```

Decode the passed event status word and return an appropriate error object.

Parameters

stat	The amplifier event status register

Returns

A pointer to an error object, or NULL if there is no error.

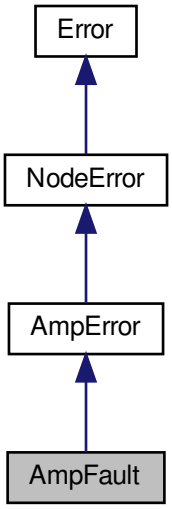
The documentation for this class was generated from the following files:

- CML_Amp.h
- Amp.cpp

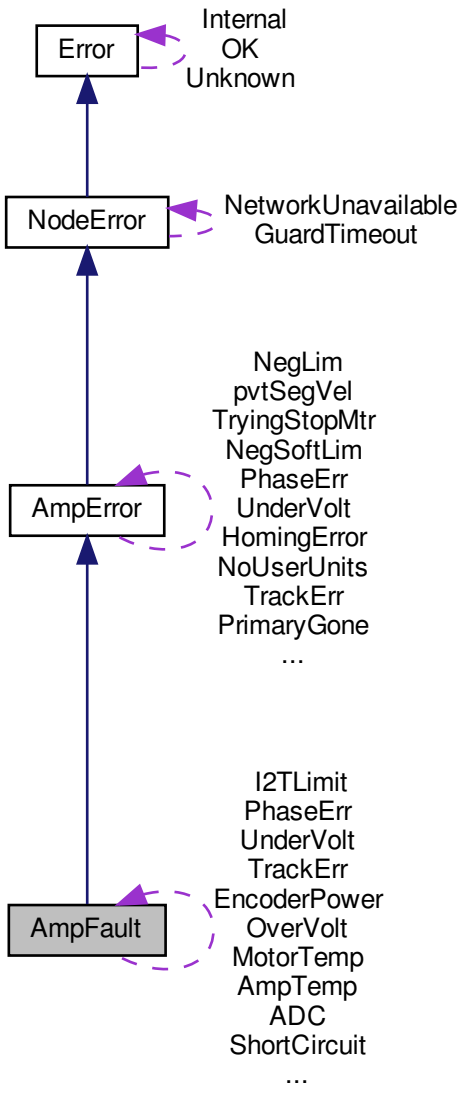
5.7 AmpFault Class Reference

This class represents latching amplifier fault conditions.

Inheritance diagram for AmpFault:



Collaboration diagram for AmpFault:



Static Public Member Functions

- static const AmpFault * DecodeFault (AMP_FAULT f)

  Return an appropriate fault object based on the amplifier fault mask.

Static Public Attributes

- static const AmpFault Memory

Fatal hardware error: the flash data is corrupt.

- static const AmpFault ADC

  Fatal hardware error: An A/D offset error has occurred.
- static const AmpFault ShortCircuit

  The amplifier detected a short circuit condition.
- static const AmpFault AmpTemp

  The amplifier is over temperature.
- static const AmpFault MotorTemp

  A motor temperature error was detected.
- static const AmpFault OverVolt

  The amplifier bus voltage is over the acceptable limit.
- static const AmpFault UnderVolt

  The amplifier bus voltage is below the acceptable limit.
- static const AmpFault EncoderPower

  Over current on the encoder power supply.
- static const AmpFault PhaseErr

  Amplifier phasing error.
- static const AmpFault TrackErr

  Tracking error, the position error is too large.
- static const AmpFault I2TLimit

  Current limited by I^2 algorithm.
- static const AmpFault Unknown

  Some unknown amplifier fault has occurred.

Protected Member Functions

- AmpFault (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.7.1 Detailed Description

This class represents latching amplifier fault conditions.

5.7.2 Member Function Documentation

5.7.2.1 DecodeFault()

```
const AmpFault * DecodeFault (
            AMP_FAULT fault )  [static]
```

Return an appropriate fault object based on the amplifier fault mask.

Parameters

fault	The amplifier fault mask.

Returns

A pointer to the fault object, NULL if there is no fault.

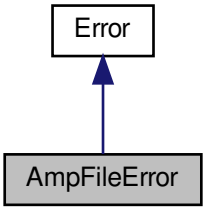
The documentation for this class was generated from the following files:

- CML_Amp.h
- Amp.cpp

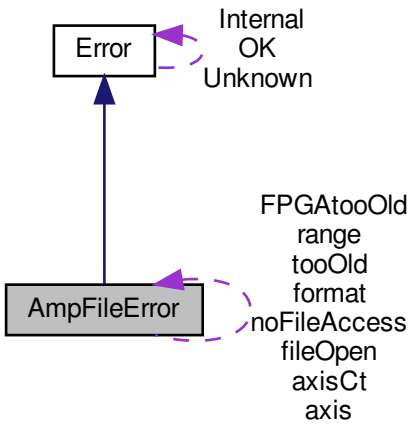
5.8 AmpFileError Class Reference

This class represents error conditions that can occur when loading amplifer data from a data file.

Inheritance diagram for AmpFileError:



Collaboration diagram for AmpFileError:



Static Public Attributes

- static const AmpFileError format

  Amplifier file format error.
- static const AmpFileError tooOld

  Amplifier file format is too old, use CME version 3.1 or later.
- static const AmpFileError noFileAccess

  File access was not enabled at compile time. See CML_Settings.h.
- static const AmpFileError fileOpen

  Error opening amplifier file.
- static const AmpFileError range

  A parameter in the amplifier file is out of range.
- static const AmpFileError axis

  Amplifier file is for multi axis, not supported.
- static const AmpFileError axisCt

  Amplifier requested an axis that is not present in the ccx file or vice versa.
- static const AmpFileError FPGAtooOld

  Amplifier firmware is too old, update to 2.72 or later.

Protected Member Functions

- AmpFileError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.8.1 Detailed Description

This class represents error conditions that can occur when loading amplifer data from a data file.

The documentation for this class was generated from the following file:

- CML_AmpStruct.h

5.9 AmpInfo Struct Reference

Amplifier characteristics data structure.

Public Attributes

- char model [COPLEY_MAX_STRING]
  Model number string.
- char mfgName [COPLEY_MAX_STRING]
  Name of the amplifier manufacturer.
- char mfgWeb [COPLEY_MAX_STRING]
  Web address of the manufacturer.
- char mfgInfo [COPLEY_MAX_STRING]
  Amplifier's manufacturing information string.
- char swVer [COPLEY_MAX_STRING]
  Software version number.
- uint16 swVerNum
  Version number represented as an integer.
- uint32 serial
  Serial number.
- uint32 modes
  Supported modes of operation (see DSP402 spec)
- uint16 crntPeak
  Peak current rating (10 milliamp units)
- uint16 crntCont
  Continuous current rating (10 milliamp units)
- uint16 crntTime
  Time at peak current (milliseconds)
- uint16 voltMax
  Max bus voltage (100 millivolt units)
- uint16 voltMin
  Min bus voltage (100 millivolt units)
- uint16 voltHyst
  Bus voltage hysteresis for over voltage shutdown (100 millivolt units)

- uint16 tempMax

  Max temperature (deg C)
- uint16 tempHyst

  Temperature hysteresis for over temp shutdown (deg C)
- uint16 pwmPeriod

  PWM period (10 nanosecond units)
- uint16 servoPeriod

  Servo period (multiples of PWM period)
- int16 crntScale

  Current scaling factor.
- int16 voltScale

  Voltage scaling factor.
- int16 refScale

  Reference scaling factor.
- int16 aencScale

  Analog encoder scaling factor.
- int16 type

  Amp type.
- int16 pwm_off

  PWM off time.
- int16 pwm_dbzero

  PWM deadband @ zero current.
- int16 pwm_dbcont

  PWM deadband @ continuous current.
- int16 regenPeak

  Internal regen resister peak limit.
- int16 regenCont

  Internal regen resister continuous limit.
- int16 regenTime

  Internal regen resister peak time.

5.9.1 Detailed Description

Amplifier characteristics data structure.

This structure is used to hold information about the amplifier such as it's model number, serial number, peak current rating, etc.

The amplifier characteristics defined in this structure can not be changed. They are defined by Copley Controls when the amplifier is designed and/or manufactured.

Use the Amp::GetAmpInfo method to retrieve this information from the amplifier.

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.10 AmploCfg Struct Reference

Programmable I/O pin configuration.

Public Member Functions

- AmploCfg (void)

  Default constructor for AmploCfg structure.

Public Attributes

- uint8 inputCt

  Number of programmable inputs available on this amplifier.
- uint8 outputCt

  Number of programmable outputs available on this amplifier.
- uint16 inPullUpCfg

  Input pin pull-up / pull-down resister configuration.
- int32 inPullUpCfg32

  Input pin pull-up / pull-down resister configuration.
- INPUT_PIN_CONFIG inCfg [COPLEY_MAX_INPUTS]

  Input pin configuration for each pin.
- int16 inDebounce [COPLEY_MAX_INPUTS]

  Input pin debounce time (milliseconds)
- OUTPUT_PIN_CONFIG outCfg [COPLEY_MAX_OUTPUTS]

  Output pin configuration for each output pin See Amp::SetOutputConfig for more information.
- uint32 outMask [COPLEY_MAX_OUTPUTS]

  Output pin configuration mask for each pin.
- uint32 outMask1 [COPLEY_MAX_OUTPUTS]

  Output pin configuration mask for each pin.
- int32 ioOptions

  IO Options, used to configure the optional features of the general purpose IO.

5.10.1 Detailed Description

Programmable I/O pin configuration.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 AmpIoCfg()

```
AmpIoCfg (
            void  )
```

Default constructor for AmpIoCfg structure.

This simply sets all member parameters to default values of zero.

5.10.3 Member Data Documentation

5.10.3.1 inCfg

```
INPUT_PIN_CONFIG inCfg[COPLEY_MAX_INPUTS]
```

Input pin configuration for each pin.

See Amp::SetInputConfig for more information.

5.10.3.2 inPullUpCfg

```
uint16 inPullUpCfg
```

Input pin pull-up / pull-down resister configuration.

See Amp::SetIoPullup for more information

5.10.3.3 inPullUpCfg32

```
int32 inPullUpCfg32
```

Input pin pull-up / pull-down resister configuration.

32 Bit version of paramter inPullUpCfg

5.10.3.4 inputCt

```
uint8 inputCt
```

Number of programmable inputs available on this amplifier.

This is a read only parameter, writes have no effect.

5.10.3.5 outMask

```
uint32 outMask[COPLEY_MAX_OUTPUTS]
```

Output pin configuration mask for each pin.

See Amp::SetOutputConfig for more information

5.10.3.6 outMask1

```
uint32 outMask1[COPLEY_MAX_OUTPUTS]
```

Output pin configuration mask for each pin.

See Amp::SetOutputConfig for more information

5.10.3.7 outputCt

```
uint8 outputCt
```

Number of programmable outputs available on this amplifier.

This is a read only parameter, writes have no effect.

The documentation for this struct was generated from the following files:

- CML_AmpStruct.h
- AmpStruct.cpp

5.11 AmpSettings Class Reference

Copley amplifier settings object.

Public Member Functions

- AmpSettings ()

  Create a settings object with all default values.

Public Attributes

- uint32 synchPeriod

  Synch object period in microseconds.
- uint32 synchID

  Synch object CAN message ID.
- bool synchUseFirstAmp

  Use first initialized amplifier as synch producer.
- bool synchProducer

  Synch producer (true/false) If true, this node will produce synch messages.
- uint32 timeStampID

  High resolution time stamp CAN ID.
- uint16 heartbeatPeriod

  The CANopen heartbeat protocol is one of two standard methods used to constantly watch for network or device problems.
- uint16 heartbeatTimeout

  Additional time to wait before generating a heartbeat error (milliseconds) If the heartbeat protocol is used, then this value, combined with the heartbeatTime will determine how long the network master waits for the node's heartbeat message before it generates a heartbeat error.
- uint16 guardTime

  Node guarding guard time (milliseconds)
- uint8 lifeFactor

  Node guarding life time factor.
- bool enableOnInit

  Enable amplifier at init time.
- AMP_MODE initialMode

  Initial mode of operation.
- bool resetOnInit

  Reset the amplifier on init.
- uint8 maxPvtSendCt

  Max PVT segments to send in response to a PVT status update.

5.11.1 Detailed Description

Copley amplifier settings object.

This object is passed to the Init() method of the Copley amp. It holds the various customizable settings used by the amplifier.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 AmpSettings()

```
AmpSettings (
            void  )
```

Create a settings object with all default values.

Default constructor for amplifier settings object.

The default values for each member of this class are defined below.

This constructor sets all the settings to the default values.

5.11.3 Member Data Documentation

5.11.3.1 enableOnInit

```
bool enableOnInit
```

Enable amplifier at init time.

If this is true, then the amplifier will be enabled at the end of a successful init(). If false, the amplifier will be disabled when init() returns.

Default: true

5.11.3.2 guardTime

```
uint16 guardTime
```

Node guarding guard time (milliseconds)

The CANopen node guarding protocol is a second method (the first being the heartbeat protocol) for devices on the network to watch for network problems. In this protocol, the master controller sends a request message out to the slave device at a specified interval. The slave device responds to this request with a message indicating it's state.

The main difference between this protocol and the heartbeat protocol is that both the slave node and the master are able to recognize network errors. With the heartbeat protocol only the network master is able to identify network problems.

Note that only one of these two protocols can be active in a node device at any time. If the heartbeat period is non-zero, then the heartbeat protocol will be used.

This parameter gives the node guarding period for use with this node. This is the period between node guarding request messages sent by the master controller.

Note that both this parameter, and the life time factor must be non-zero for node guarding to be used.

For EtherCAT networks, this parameter specifies the process data heartbeat timeout. This is similar to the CANopen node guarding timeout. If the drive is operational, and goes this long without receiving process data from the master, then it will identify a timeout condition.

Default 200 (ms)

5.11.3.3 heartbeatPeriod

```
uint16 heartbeatPeriod
```

The CANopen heartbeat protocol is one of two standard methods used to constantly watch for network or device problems.

When the heartbeat protocol is used, each device on the CANopen network transmits a 'heartbeat' message at a specified interval. The network master watches for these messages, and is able to detect a device error if it's heartbeat message is not received within the expected time.

This parameter configures the heartbeat period (milliseconds) that will be used by this amplifier to transmit it's heartbeat message.

If this parameter is set to zero, then the heartbeat protocol is disabled on this node.

This setting is not used on EtherCAT networks.

Default: zero (not used)

5.11.3.4 heartbeatTimeout

```
uint16 heartbeatTimeout
```

Additional time to wait before generating a heartbeat error (milliseconds) If the heartbeat protocol is used, then this value, combined with the heartbeatTime will determine how long the network master waits for the node's heartbeat message before it generates a heartbeat error.

Note that setting this to zero does not disable the heartbeat protocol. set the heartbeatPeriod value to zero to disable heartbeat.

This setting is not used on EtherCAT networks.

Default 200 (ms)

5.11.3.5 initialMode

```
AMP_MODE initialMode
```

Initial mode of operation.

This defines the mode of operation that the amplifier will be placed in when it is initialized.

Default: AMPMODE_CAN_HOMING

5.11.3.6 lifeFactor

```
uint8 lifeFactor
```

Node guarding life time factor.

When the node guarding protocol is used, this parameter is used by the slave device to determine how long to wait for a node guarding request from the master controller before signaling an error condition.

The life time factor is treated as a multiple of the guard time.

If this parameter and the node guard time are both non-zero, and the heartbeatTime is zero, then node guarding will be setup for the amplifier.

This setting is not used on EtherCAT networks.

Default 3 (multiples of the guard time)

5.11.3.7 maxPvtSendCt

```
uint8 maxPvtSendCt
```

Max PVT segments to send in response to a PVT status update.

This parameter may be used to limit the number of new PVT segments to send in response to a PVT status update. Normally, this parameter may be safely left at it's default setting.

Default 6

5.11.3.8 resetOnInit

```
bool resetOnInit
```

Reset the amplifier on init.

If true, the amplifier will be reset when it is initialized.

Note that resetting nodes on an EtherCAT network will make it impossible to access anything on the network while the reset is occuring. Resetting EtherCAT nodes is therefore not recommended.

Default: false

5.11.3.9 synchID

```
uint32 synchID
```

Synch object CAN message ID.

This is the message ID used for the synch message. Default is 0x00000080

5.11.3.10 synchPeriod

```
uint32 synchPeriod
```

Synch object period in microseconds.

The synch object is a message that is transmitted by one node on a CANopen network at a fixed interval. This message is used to synchronize the devices on the network.

Default is 10,000 (10 ms).

5.11.3.11 synchProducer

```
bool synchProducer
```

Synch producer (true/false) If true, this node will produce synch messages.

Default: false

Note: If the 'synchUseFirstAmp' setting of this object is true, then the passed value of this settings will not be used, and will be overwritten during initialization.

Note: There should be exactly one synch producer on every network.

5.11.3.12 synchUseFirstAmp

```
bool synchUseFirstAmp
```

Use first initialized amplifier as synch producer.

If this setting is true (default), then the first amplifier to be initialized will be set as the synch producer, and all other amplifier's will be setup as synch consumers. This causes the value of the 'synchProducer' setting to be updated during init to indicate whether the amp is producing synch messages or not.

By default, this setting is true

5.11.3.13 timeStampID

```
uint32 timeStampID
```

High resolution time stamp CAN ID.

The high resolution time stamp is a PDO that is generated by the synch producer and consumed by the other amplifiers on the network. It is used to synchronize the clocks of the amplifiers. This parameter defines the CAN ID that will be used for this message. Setting to zero will disable the time stamp message. Default 0x0180

The documentation for this class was generated from the following files:

- CML_Amp.h
- Amp.cpp

5.12 AnalogRefConfig Struct Reference

Analog input configuration.

Public Member Functions

- AnalogRefConfig (void)

  Default constructor. Simply sets all members to zero.

Public Attributes

- int16 calibration

  Calibration offset.
- int16 offset

  Offset in millivolts.
- int16 deadband

  Deadband in millivolts.
- int32 scale

  Scaling factor.

5.12.1 Detailed Description

Analog input configuration.

These parameters are used when the amplifier is being driven from it's analog reference input pin. Note that not all amplifier have an analog input reference, and that this is not a standard CANopen mode of operation.

5.12.2 Member Data Documentation

5.12.2.1 calibration

```
int16 calibration
```

Calibration offset.

This offset is set at the factory and should normally not be modified. Units are millivolts

5.12.2.2 deadband

```
int16 deadband
```

Deadband in millivolts.

The analog input will be treated as zero when it's absolute value is less then this.

5.12.2.3 scale

```
int32 scale
```

Scaling factor.

Units are dependent on the mode of operation: 0.01 Amp when driving current. 0.1 Encoder counts/second when driving velocity

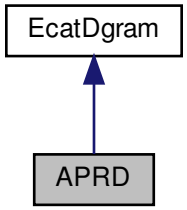
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

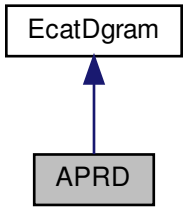
5.13 APRD Struct Reference

Read by position in network (aka Auto Increment Physical Read) The read is performed on the node who's position matches the passed address.

Inheritance diagram for APRD:



Collaboration diagram for APRD:



Additional Inherited Members

5.13.1 Detailed Description

Read by position in network (aka Auto Increment Physical Read) The read is performed on the node who's position matches the passed address.

Pass 0 for the node closest to the master, 1 for the next node, etc. In the datagram sent out, the ADP field will hold -N for a passed address of N. Each node increments the ADP value as it passes, and the node that receives a zero responds to the read.

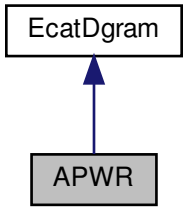
The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

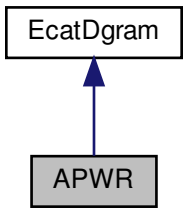
5.14 APWR Struct Reference

Write by position in network (Auto Increment Physical Write) Like the APRD datagram, but a write version.

Inheritance diagram for APWR:



Collaboration diagram for APWR:



Additional Inherited Members

5.14.1 Detailed Description

Write by position in network (Auto Increment Physical Write) Like the APRD datagram, but a write version.

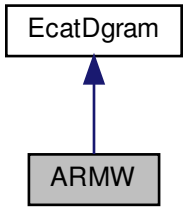
The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

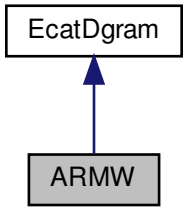
5.15 ARMW Struct Reference

Read by position in network and write to the same address of all following nodes.

Inheritance diagram for ARMW:



Collaboration diagram for ARMW:



Additional Inherited Members

5.15.1 Detailed Description

Read by position in network and write to the same address of all following nodes.

The read is performed on the node who's position matches the passed address. Pass 0 for the node closest to the master, 1 for the next node, etc. All nodes after that position will have the data read from the earlier node written to the same address

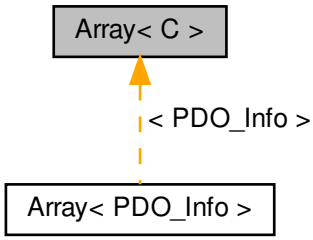
The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

5.16 Array< C > Class Template Reference

This class template implements a simple dynamic array of a given type.

Inheritance diagram for Array< C >:



Public Member Functions

- Array (int init=32, int step=-1)

  Default constructor for a dynamic array.
- virtual ~Array ()

  Default destructor.
- int length (void)

  Returns the current number of elements currently stored in the Array.
- C & operator[ ] (int ndx)

  Return a reference to the item at the specified index.
- void add (C val)

  Append a value to the end of the array.
- void rem (int ndx)

  Remove the element at the specified index.

5.16.1 Detailed Description

template<class C>
class Array< C >

This class template implements a simple dynamic array of a given type.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Array()

```
Array (
            int init = 32,
            int step = -1 )  [inline]
```

Default constructor for a dynamic array.

Parameters

init	Number of objects to initially allocate space for
step	When new data is needed, the array size will be increased by this amount If this parameter is negative (default), then the initial value will be used. If this parameter is zero, then the array size will be doubled on each increase.

5.16.3 Member Function Documentation

5.16.3.1 add()

```
void add (
            C val )  [inline]
```

Append a value to the end of the array.

The array will grow to accomodate the new data if necessary

Parameters

val	Value to append.

5.16.3.2 length()

```
int length (
            void  )  [inline]
```

Returns the current number of elements currently stored in the Array.

Returns

Number of stored elements

5.16.3.3 operator[]()

```
C& operator[] (
            int ndx )  [inline]
```

Return a reference to the item at the specified index.

Parameters

ndx	Index of the element to return

Returns

Reference to the data at the specified index.

5.16.3.4 rem()

```
void rem (
            int ndx )  [inline]
```

Remove the element at the specified index.

The array will be compacted

Parameters

ndx	The index of the value to remove

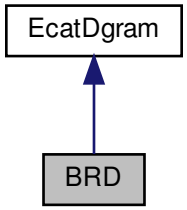
The documentation for this class was generated from the following file:

- CML_Array.h

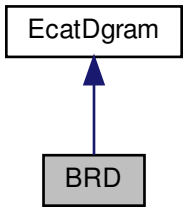
5.17 BRD Struct Reference

Broadcast read.

Inheritance diagram for BRD:



Collaboration diagram for BRD:



Additional Inherited Members

5.17.1 Detailed Description

Broadcast read.

This type of datagram reads data from the same location on every node in the network. Returned data is ORed, so any bit set in any node will be set in the accumulated response. The ADP address is sent out from the master as zero and incremented by every slave that responds to the read.

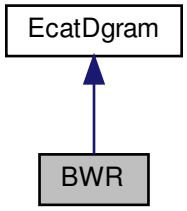
The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

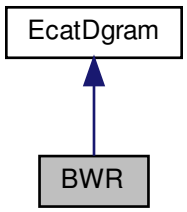
5.18 BWR Struct Reference

Broadcast write. This type of datagram writes data to the same location on every node in the network.

Inheritance diagram for BWR:



Collaboration diagram for BWR:



Additional Inherited Members

5.18.1 Detailed Description

Broadcast write. This type of datagram writes data to the same location on every node in the network.

The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

5.19 CammingConfig Struct Reference

Configuration structure used to set up the camming.

Public Member Functions

- CammingConfig (void)
  Default constructor.

Public Attributes

- uint16 cammingModeConfig
  Camming Mode configuration. See documentation for details.
- uint16 cammingDelayForward
  Camming delay forward.(Units:master command counts)
- uint16 cammingDelayReverse
  Camming delay forward.(Units:master command counts)
- int32 cammingMasterVel
  Camming master velocity.

5.19.1 Detailed Description

Configuration structure used to set up the camming.

These settings may be up/download from the amplifier using the functions Amp::SetCammingConfig and Amp::Get↩ CammingConfig.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 CammingConfig()

```
CammingConfig (
            void  )  [inline]
```

Default constructor.

Initializes all structure elements to zero.

5.19.3 Member Data Documentation

5.19.3.1 cammingMasterVel

```
int32 cammingMasterVel
```

Camming master velocity.

Constnat velocity of the Camming internal generator. (Units 0.1 counts/s)

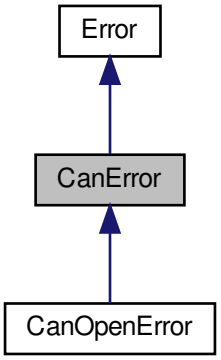
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

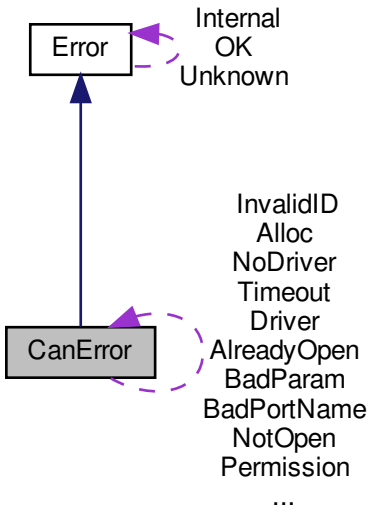
5.20 CanError Class Reference

Class used to represent an error condition returned from a CAN interface function.

Inheritance diagram for CanError:



Collaboration diagram for CanError:



Static Public Attributes

- static const CanError BadPortName

  Indicates that the specified port name is invalid.
- static const CanError NotOpen

  Indicates that the CAN port is not open.
- static const CanError AlreadyOpen

  Indicates an illegal attempt to open an already open port.
- static const CanError BadParam

  A parameter passed to the CAN member function is not valid.
- static const CanError Driver

  Generic error from the CAN driver.
- static const CanError BadBaud

  Illegal baud rate specified.
- static const CanError Timeout

  Timeout waiting on read/write.
- static const CanError Overflow

  CAN buffer overflow.
- static const CanError BusOff

  CAN bus is in the OFF state.
- static const CanError InvalidID

  Indicates an invalid CAN ID passed.
- static const CanError Unknown

  Unknown CAN error condition.
- static const CanError NoDriver

  Unable to open CAN driver, or missing dll file.
- static const CanError Alloc

  CAN driver memory allocation error.
- static const CanError Permission

  Permission error opening CAN port.

Additional Inherited Members

5.20.1 Detailed Description

Class used to represent an error condition returned from a CAN interface function.

The documentation for this class was generated from the following file:

- CML_Can.h

5.21 CanFrame Struct Reference

Low level CAN data frame.

Public Attributes

- CAN_FRAME_TYPE type

  Identifies the frame type.
- int32 id

  The CAN message ID.
- uint32 timestamp

  Timestamp of received frame if supported by CAN interface.
- byte length

  Gives the number of bytes of data included in the frame.
- byte data [8]

  Holds any data sent with the frame.

5.21.1 Detailed Description

Low level CAN data frame.

This class is used to represent the basic frame of information that is passed over the CAN network.

A frame of CAN data consists of a message ID value (either 11 or 29 bits depending on whether standard or extended frames are in use), 0 to 8 bytes of data, and some special attributes.

Frame objects are passed to the CanInterface::Xmit() function, and filled in by the CanInterface::Recv() function.

5.21.2 Member Data Documentation

5.21.2.1 data

```
byte data[8]
```

Holds any data sent with the frame.

A frame may be accompanied by 0 to 8 bytes of data.

5.21.2.2 id

```
int32 id
```

The CAN message ID.

If bit 29 is clear, this is a standard 11 bit ID (bits 0-10 hold the ID). If bit 29 is set, this is an extended 29 bit ID (bits 0-28 hold the ID). Bits 30 and 31 are not presently used.

5.21.2.3 length

```
byte length
```

Gives the number of bytes of data included in the frame.

The length of a frame may range from 0 to 8 bytes

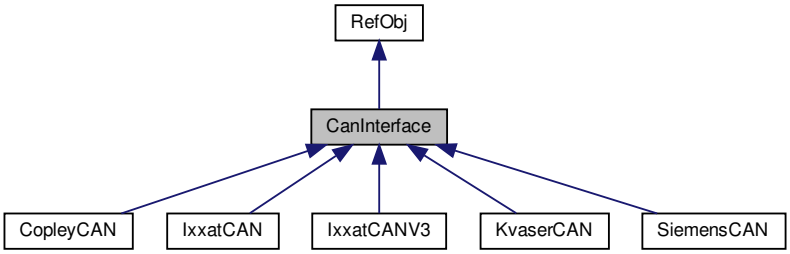
The documentation for this struct was generated from the following file:

- CML_Can.h

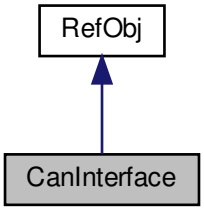
5.22 CanInterface Class Reference

Abstract class used for low level interaction with CAN hardware.

Inheritance diagram for CanInterface:



Collaboration diagram for CanInterface:



Public Member Functions

- CanInterface (void)

  Default constructor for a CAN interface object.
- CanInterface (const char *port)

  Standard constructor for the CAN interface object.
- virtual ~CanInterface ()

  Standard destructor for base CanInterface object.
- virtual const Error * SetName (const char *name)

  Set the name of the port.
- virtual const Error * Open (void)

  Open the CAN interface.
- virtual const Error * Close (void)

  Close the CAN interface.
- virtual const Error * SetBaud (int32 baud)

  Set the CAN interface baud rate.
- const Error * Recv (CanFrame &frame, Timeout timeout=-1)

  Receive the next CAN frame.
- const Error * Xmit (CanFrame &frame, Timeout timeout=0)

  Write a CAN frame to the CAN network.
- virtual bool SupportsTimestamps (void)

  Return true if the CAN interface supports timestamps on received frames.

Static Public Member Functions

- static const Error * ChkID (int32 id)

  Check an ID to make sure it's valid.

Protected Member Functions

- virtual const Error * RecvFrame (CanFrame &frame, Timeout timeout)

  Receive the next CAN frame.
- virtual const Error * XmitFrame (CanFrame &frame, Timeout timeout)

  Write a CAN frame to the CAN network.

Protected Attributes

- char * portName

  This string is initialized by the default constructor.

5.22.1 Detailed Description

Abstract class used for low level interaction with CAN hardware.

This class contains methods that are used to open and close the CAN network adapter, as well as transmit and receive frames over the CAN network.

The base CanInterface class defines a standard minimal interface to the CAN network. This base class does not actually provide support for any hardware, rather it should be extended by a class that provides access to the actual CAN network adapter being used.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 CanInterface()

```
CanInterface (
            const char * port )  [inline]
```

Standard constructor for the CAN interface object.

The only thing that the constructor does is initialize the portName member variable to the value passed.

Parameters

port	A string that may be used to identify which port to open.

5.22.3 Member Function Documentation

5.22.3.1 ChkID()

```
static const Error* ChkID (
            int32 id )  [inline], [static]
```

Check an ID to make sure it's valid.

To be valid, a message ID must either be an 11 bit standard ID, or a 28-bit extended ID. By convention, all extended ID's must have bit 29 set to identify them as such.

Parameters

id	The ID to be checked

Returns

A pointer to an error object, or NULL on success.

5.22.3.2 Close()

```
virtual const Error* Close (
            void  )  [inline], [virtual]
```

Close the CAN interface.

Returns

A valid CAN error object

Reimplemented in SiemensCAN, KvaserCAN, IxxatCANV3, IxxatCAN, and CopleyCAN.

5.22.3.3 Open()

```
virtual const Error* Open (
            void  )  [inline], [virtual]
```

Open the CAN interface.

Returns

A valid CAN error object

Reimplemented in SiemensCAN, KvaserCAN, IxxatCANV3, IxxatCAN, and CopleyCAN.

5.22.3.4 Recv()

```
const Error * Recv (
            CanFrame & frame,
            Timeout timeout = -1 )
```

Receive the next CAN frame.

This is the public function used to read CAN messages from the network.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A pointer to an error object, or NULL on success.

5.22.3.5 RecvFrame()

```
const Error * RecvFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Receive the next CAN frame.

This is called by the public Recv function, and must be implemented by the actual CanInterace class. It handles the hardware specific details of reading a message from the network.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A pointer to an error object, or NULL on success.

Reimplemented in SiemensCAN, KvaserCAN, IxxatCANV3, IxxatCAN, and CopleyCAN.

5.22.3.6 SetBaud()

```
virtual const Error* SetBaud (
            int32 baud )  [inline], [virtual]
```

Set the CAN interface baud rate.

Parameters

baud	In bits / second

Returns

A valid CAN error object

Reimplemented in SiemensCAN, KvaserCAN, IxxatCANV3, IxxatCAN, and CopleyCAN.

5.22.3.7 SetName()

```
const Error * SetName (
            const char * name )  [virtual]
```

Set the name of the port.

Parameters

name	The port name.

Returns

A pointer to an error object, or NULL on success.

5.22.3.8 SupportsTimestamps()

```
virtual bool SupportsTimestamps (
            void  )  [inline], [virtual]
```

Return true if the CAN interface supports timestamps on received frames.

Returns

true if timestamps are supported

Reimplemented in CopleyCAN.

5.22.3.9 Xmit()

```
const Error * Xmit (
            CanFrame & frame,
            Timeout timeout = 0 )
```

Write a CAN frame to the CAN network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A pointer to an error object, or NULL on success.

5.22.3.10 XmitFrame()

```
const Error * XmitFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Write a CAN frame to the CAN network.

This is called by the public Xmit function, and must be implemented by the actual CanInterface class. It handles the hardware specific details of writing a message to the network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A pointer to an error object, or NULL on success.

Reimplemented in SiemensCAN, KvaserCAN, IxxatCANV3, IxxatCAN, and CopleyCAN.

5.22.4 Member Data Documentation

5.22.4.1 portName

```
char* portName  [protected]
```

This string is initialized by the default constructor.

It may be used to identify which CAN port to open, etc.

The documentation for this class was generated from the following files:

- CML_Can.h
- Can.cpp

5.23 CanNetworkConfig Struct Reference

CANopen Node ID and bit rate configuration.

Public Member Functions

- CanNetworkConfig ()
  Default constructor. Set default values.
- void FromAmpFormat (uint16 a)
  Load the structure from a 16-bit word.
- uint16 ToAmpFormat (void)
  Encode the contents of the structure into a 16-bit word in the format used by the amplifier.

Public Attributes

- uint8 numInPins
  Number of general purpose input pins to read on startup for node ID selection.
- bool useSwitch
  If true, use the CAN address switch as part of the node ID calculation.
- uint8 offset
  Offset added to the value read from the input pins & address switch.
- CAN_BIT_RATE bitRate
  CANopen network bit rate to use.
- uint32 pinMapping
  Input pin mapping for node ID selection.
- uint16 quickStop
  CANopen quick stop option code.
- uint16 shutDownOption
  CANopen shutdown option code.
- uint16 disableOption
  CANopen disable option code.
- uint16 haltOption
  CANopen halt option code.
- uint16 heartbeat
  CANopen heartbeat time.
- uint16 nodeGuard
  CANopen Node Guard time.
- uint16 nodeGuardLife
  CANopen Node Guarding Life time factor.

5.23.1 Detailed Description

CANopen Node ID and bit rate configuration.

The amplifier's CANopen node ID and network bit rate can be configured using the members of this structure. Note that the ID and bit rate are only set on power-up or reset, so after programming a new configuration the amplifier must be reset for the configuration to become active.

The CANopen node ID is a 7 bit number in the range 1 to 128. The value 0 is reserved and is not considered a valid node ID. Selecting a node ID of 0 will cause the amplifier to stop communicating over the CANopen network.

The node ID is calculated using a combination of any of the following:

- The CAN address switch on amplifiers which support this feature
- 0 to 7 general purpose input pins.
- A programmable offset in the range 0 to 128.

On startup, the input pins are read first. The inputs that will be used for CAN address selection are the highest numbered pins available. For example, on an amplifier with 12 input pins (0 to 11), if 3 inputs are used for CANopen node ID selection, then the pins used will be 9, 10 and 11. These three pins will result in a base node address between 0 and 7.

If the CAN address switch is being used, then the value read from the input pins will be shifted up four bits (multiplied by 16) and the value of the input switch will be added to it.

Finally, the programmed offset will be added to the value read from the input pins and address switch. The lowest 7 bits of this sum will be used as the CANopen node ID. If this value results in an ID of zero, the CANopen interface will be disabled.

For example, to program an amplifier to ignore input pins and the address switch, and just set a fixed ID of 7, set the number of input pins to zero, turn off the address switch, and set the offset to 7.

5.23.2 Member Function Documentation

5.23.2.1 FromAmpFormat()

```
void FromAmpFormat (
            uint16 a )
```

Load the structure from a 16-bit word.

Parameters

a	A 16-bit value encoding the network configuration. See the amplifier documentation for details on the format.

5.23.2.2 ToAmpFormat()

```
uint16 ToAmpFormat (
            void  )
```

Encode the contents of the structure into a 16-bit word in the format used by the amplifier.

See the amplifier documentation for details on this format.

Returns

A 16-bit word representing the contents of this structure.

5.23.3 Member Data Documentation

5.23.3.1 heartbeat

```
uint16 heartbeat
```

CANopen heartbeat time.

The frequency at which the amp will produce heartbeat messages. This parameter may be set to zero to disable heartbeat production.

5.23.3.2 nodeGuard

```
uint16 nodeGuard
```

CANopen Node Guard time.

This parameter gives the time between nodeguarding requests that are sent form the CANopen master to this amplifier

5.23.3.3 nodeGuardLife

```
uint16 nodeGuardLife
```

CANopen Node Guarding Life time factor.

This objct gives a multiple of the CANopen Node Guarding Time

5.23.3.4 numInPins

```
uint8 numInPins
```

Number of general purpose input pins to read on startup for node ID selection.

This parameter may be set from 0 to 7.

5.23.3.5 offset

```
uint8 offset
```

Offset added to the value read from the input pins & address switch.

5.23.3.6 pinMapping

```
uint32 pinMapping
```

Input pin mapping for node ID selection.

When network node id indicates that 1 or more input pins will be used to select the node ID, this parameter is used to map input pins to ID bits. Availabel after 3.35 firmware. See documentation

5.23.3.7 useSwitch

```
bool useSwitch
```

If true, use the CAN address switch as part of the node ID calculation.

Note that on amplifiers which do not support this switch this parameter is ignored.

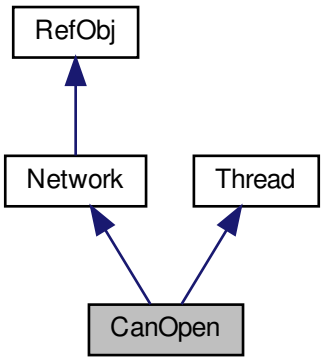
The documentation for this struct was generated from the following files:

- CML_AmpStruct.h
- AmpStruct.cpp

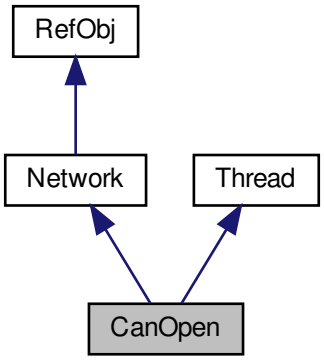
5.24 CanOpen Class Reference

The CanOpen class is the top level interface into the CANopen network.

Inheritance diagram for CanOpen:



Collaboration diagram for CanOpen:



Public Member Functions

- CanOpen (void)

Default constructor.

- virtual ~CanOpen (void)

  CanOpen Destructor.
- const Error * Open (CanInterface &can)

  Open the CANopen network.
- const Error * Open (CanInterface &can, CanOpenSettings &settings)

  Open the CANopen network.
- void Close (void)

  Close the CANopen network.
- NetworkType GetNetworkType (void)

  Return the network type.
- const Error * AttachNode (Node *n)

  Attach the passed node to this network.
- const Error * DetachNode (Node *n)

  Detach the passed node from this network.
- const Error * SetNodeGuard (Node *n, GuardProtocol type, Timeout timeout=50, uint8 life=3)

  Configure the node guarding protocol for a CANopen node.
- const Error * ResetNode (Node *n)

  Send a network management message to reset the specified node.
- const Error * ResetComm (Node *n)

  Send a network management message to reset the communications of the specified node.
- const Error * PreOpNode (Node *n)

  Send a network management message to put the specified node in pre-operational state.
- const Error * StartNode (Node *n)

  Send a network management message to start the specified node.
- const Error * StopNode (Node *n)

  Send a network management message to stop the specified node.
- const Error * BootModeNode (Node *n)

  For CANopen networks, this is the same as CanOpen::StopNode();.
- const Error * Xmit (CanFrame &frame, Timeout timeout=2000)

  Transmit a frame over the CANopen network.
- const Error * XmitSDO (Node *n, uint8 *data, uint16 len, uint16 *ret, Timeout timeout=2000)

  Transmit an SDO message over the CANopen network and wait for a response.
- const Error * XmitPDO (class PDO *pdo, Timeout timeout=2000)

  Transmit a PDO over the CANopen network.
- int16 GetSynchProducer (void)

  Return the node ID of the synch producer for this network.
- void SetSynchProducer (int16 nodeID)

  Set the node ID of the synch producer for this network.
- int32 GetErrorFrameCounter (void)

  Return the number of error frames received over then CAN network since the last time the counter was cleared.
- void ClearErrorFrameCounter (void)

  Clear the error frame counter.
- const Error * EnableReceiver (uint32 canMsgID, class Receiver *rcvr)

  Enable reception handling of the message identified by this Receiver object.
- const Error * DisableReceiver (uint32 canMsgID)

  Disable reception handling of a particular CANopen message type.

Additional Inherited Members

5.24.1 Detailed Description

The CanOpen class is the top level interface into the CANopen network.

There should be at least one object of this class in every CANopen based application. Normally, only one object will be necessary, however if more then one independent CANopen network is in use, then more then one object will be necessary.

On startup, a low level CAN interface object should be created. This object should be passed to the CANopen object's Open() method.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 CanOpen()

```
CanOpen (
            void  )
```

Default constructor.

Simply initializes some local variables.

5.24.2.2 ~CanOpen()

```
~CanOpen (
            void  )  [virtual]
```

CanOpen Destructor.

This closes the CANopen network.

5.24.3 Member Function Documentation

5.24.3.1 AttachNode()

```
const Error * AttachNode (
            Node * n )  [virtual]
```

Attach the passed node to this network.

This function is called by the node object when it's initialized. It connects the node to the CANopen network object so that messages bound for the node can be properly delivered.

Parameters

n	Pointer to the node to attach to the network.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.2 BootModeNode()

```
const Error * BootModeNode (
            Node * n )  [virtual]
```

For CANopen networks, this is the same as CanOpen::StopNode();.

Parameters

n	Pointer to the node to stop.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.3 Close()

```
void Close (
            void  )
```

Close the CANopen network.

This disables all receivers and stops the thread that listens on the CAN network.

5.24.3.4 DetachNode()

```
const Error * DetachNode (
            Node * n )  [virtual]
```

Detach the passed node from this network.

This function is called by the node object when it's uninitialized. It removes the connection between the node and the CANopen network object.

Parameters

n	Pointer to the node to detach from the network.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.5 DisableReceiver()

```
const Error * DisableReceiver (
            uint32 canMsgID )
```

Disable reception handling of a particular CANopen message type.

Parameters

canMsgID	The CAN message ID.

Returns

A pointer to an error object, or NULL on success.

5.24.3.6 EnableReceiver()

```
const Error * EnableReceiver (
            uint32 msgID,
            class Receiver * rcvr )
```

Enable reception handling of the message identified by this Receiver object.

The receiver is enabled by adding it to a binary tree of receiver objects maintained by the CanOpen object.

Parameters

msgID	The CAN message ID to assocaite with this receiver.
rcvr	Pointer to the receiver to add

Returns

A pointer to an error object, or NULL on success.

5.24.3.7 GetErrorFrameCounter()

```
int32 GetErrorFrameCounter (
            void  )  [inline]
```

Return the number of error frames received over then CAN network since the last time the counter was cleared.

Returns

The number of error frames received since the last call to CanOpen::ClearErrorFrameCounter();

5.24.3.8 GetNetworkType()

```
NetworkType GetNetworkType (
            void  )  [inline], [virtual]
```

Return the network type.

Returns

Always returns the value NET_TYPE_CANOPEN

Implements Network.

5.24.3.9 GetSynchProducer()

```
int16 GetSynchProducer (
            void  )  [inline]
```

Return the node ID of the synch producer for this network.

Returns

The synch producer node ID, or 0 if no synch producer has been registered.

5.24.3.10 Open() [1/2]

```
const Error * Open (
            CanInterface & can )
```

Open the CANopen network.

This function performs the one time initialization necessary to communication via the CANopen network. It should be the first function called for the CANopen object.

All configurable settings will be set to their defaults when the CanOpen object is opened using this method. For a list of CanOpen object settings and their default values, please see the CanOpenSettings object.

Parameters

can	A reference to the CAN interface object that will be used for all low level communication over the network.

Returns

A pointer to an error object, or NULL on success.

5.24.3.11 Open() [2/2]

```
const Error * Open (
            CanInterface & ci,
            CanOpenSettings & settings )
```

Open the CANopen network.

This function performs the one time initialization necessary to communication via the CANopen network. It should be the first function called for the CANopen object.

This version of the Open function takes a CanOpenSettings object reference as it's second parameter. The data members of the settings object may be used to configure some of the CanOpen object's behavior.

Parameters

ci	A reference to the CAN interface object that will be used for all low level communication over the network.
settings	A reference to a CanOpenSettings object. This object is used to customize the behavior of the CanOpen object.

Returns

A pointer to an error object, or NULL on success.

5.24.3.12 PreOpNode()

```
const Error * PreOpNode (
            Node * n )  [virtual]
```

Send a network management message to put the specified node in pre-operational state.

Parameters

n	Pointer to the node.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.13 ResetComm()

```
const Error * ResetComm (
            Node * n )  [virtual]
```

Send a network management message to reset the communications of the specified node.

All nodes have their communications reset if the passed node ID is zero.

Parameters

n	A pointer to the node to reset

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.14 ResetNode()

```
const Error * ResetNode (
            Node * n )  [virtual]
```

Send a network management message to reset the specified node.

All nodes are reset if the passed node ID is zero.

Parameters

n	A pointer to the node to reset

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.15 SetNodeGuard()

```
const Error * SetNodeGuard (
            Node * n,
            GuardProtocol type,
            Timeout to = 50,
            uint8 life = 3 )  [virtual]
```

Configure the node guarding protocol for a CANopen node.

Parameters

n	The node to configure guarding on.
type	The type of node guarding to configure.
to	A timeout (milliseconds) to use for this node guarding protocol. If not specified, this parameter defaults to 50 milliseconds.
life	A life time factor for use with the node guarding protocol (default 3). Only the node guarding protocol uses this parameter, it may be omitted for other node guarding types.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.16 SetSynchProducer()

```
void SetSynchProducer (
            int16 nodeID )  [inline]
```

Set the node ID of the synch producer for this network.

Parameters

nodeID	The new synch producer node ID, or 0 for none.

5.24.3.17 StartNode()

```
const Error * StartNode (
            Node * n )  [virtual]
```

Send a network management message to start the specified node.

Parameters

n	Pointer to the node to start.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.18 StopNode()

```
const Error * StopNode (
            Node * n )  [virtual]
```

Send a network management message to stop the specified node.

Parameters

n	Pointer to the node to stop.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.19 Xmit()

```
const Error * Xmit (
            CanFrame & frame,
            Timeout timeout = 2000 )
```

Transmit a frame over the CANopen network.

Parameters

frame	Refernce to the frame to transmit
timeout	Max time to wait for the frame to be sent.

Returns

A pointer to an error object, or NULL on success.

5.24.3.20 XmitPDO()

```
const Error * XmitPDO (
            class PDO * pdo,
            Timeout timeout = 2000 )  [virtual]
```

Transmit a PDO over the CANopen network.

If the PDO is a transmit PDO (i.e. the type of PDO that's normally sent from the node and received by the master), then a remote request is sent.

Parameters

pdo	Pointer to the PDO to be transmitted.
timeout	Max time to wait for the PDO to be sent.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

5.24.3.21 XmitSDO()

```
const Error * XmitSDO (
            Node * n,
            uint8 * buff,
            uint16 len,
            uint16 * ret,
            Timeout timeout = 2000 )  [virtual]
```

Transmit an SDO message over the CANopen network and wait for a response.

The SDO should have alrady been formatted into the passed array. This function assigns the standard CAN message ID and transmits the resulting CAN frame.

Parameters

n	Pointer to the node to whom the SDO is addressed.
buff	Buffer containing the formatted SDO data
len	Length of the SDO data in bytes. Should be 8 bytes for standard CANopen SDOs.
ret	Pointer to a location where the number of received bytes will be stored on success.
timeout	Max time to wait for the SDO to be sent.

Returns

A pointer to an error object, or NULL on success.

Implements Network.

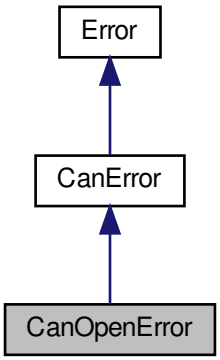
The documentation for this class was generated from the following files:

- CML_CanOpen.h
- CanOpen.cpp

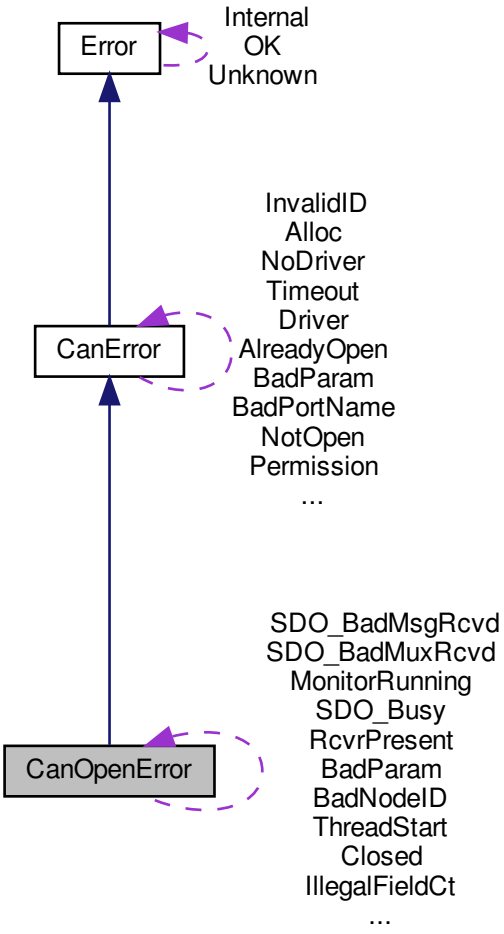
5.25 CanOpenError Class Reference

This class holds the error codes that describe CANopen error conditions.

Inheritance diagram for CanOpenError:



Collaboration diagram for CanOpenError:



Static Public Attributes

- static const CanOpenError ThreadStart

  Indicates that the specified port name is invalid.
- static const CanOpenError BadParam

  One of the parameters passed to the CANopen function is invalid.
- static const CanOpenError SDO_Busy

  The SDO object is busy.
- static const CanOpenError SDO_Timeout

  The SDO up/download failed with a timeout.
- static const CanOpenError SDO_Unknown

  Some unknown error occurred.

- static const CanOpenError SDO_BadMuxRcvd

  The mux (index/sub-index) received in a SDO message is inconsistent with the object being accessed.
- static const CanOpenError SDO_BadMsgRcvd

  An improperly formatted SDO message was received.
- static const CanOpenError BadNodeID

  An illegal node ID was specified.
- static const CanOpenError NotInitialized

  The object being used has not been initialized.
- static const CanOpenError Initialized

  An attempt was made to initialize an object that has already been initialized, and doesn't allow multiple initialization.
- static const CanOpenError NotSupported

  The requested feature is not supported by this node.
- static const CanOpenError MonitorRunning

  Monitor already running - An attempt is made to start the heartbeat or node guarding and it's already running.
- static const CanOpenError IllegalFieldCt

  The node returned an illegal field count for the object being requested in it's object dictionary.
- static const CanOpenError RcvrNotFound

  An attempt was made to disable a receiver that wasn't enabled.
- static const CanOpenError RcvrPresent

  An attempt was made to enable a receiver that was already enabled.
- static const CanOpenError Closed

  The CANopen port is closed.

Additional Inherited Members

5.25.1 Detailed Description

This class holds the error codes that describe CANopen error conditions.

5.25.2 Member Data Documentation

5.25.2.1 IllegalFieldCt

```
const CanOpenError IllegalFieldCt  [static]
```

The node returned an illegal field count for the object being requested in it's object dictionary.

5.25.2.2 Initialized

```
const CanOpenError Initialized  [static]
```

An attempt was made to initialize an object that has already been initialized, and doesn't allow multiple initialization.

5.25.2.3 MonitorRunning

```
const CanOpenError MonitorRunning  [static]
```

Monitor already running - An attempt is made to start the heartbeat or node guarding and it's already running.

5.25.2.4 NotInitialized

```
const CanOpenError NotInitialized  [static]
```

The object being used has not been initialized.

This error indicates a coding error, i.e. trying to use a Receiver object without initializing it.

5.25.2.5 SDO_BadMuxRcvd

```
const CanOpenError SDO_BadMuxRcvd  [static]
```

The mux (index/sub-index) received in a SDO message is inconsistent with the object being accessed.

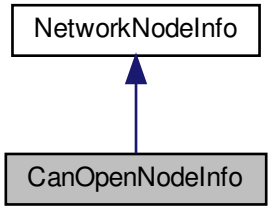
The documentation for this class was generated from the following file:

- CML_CanOpen.h

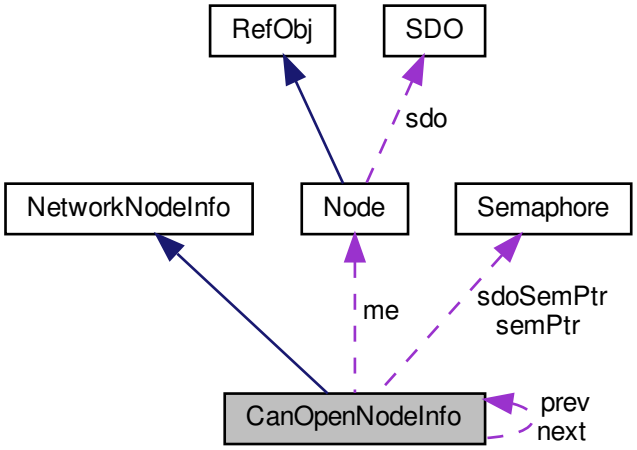
5.26 CanOpenNodeInfo Struct Reference

The CanOpenNodeInfo structure holds some data required by the CANopen network interface which is present in every node it manages.

Inheritance diagram for CanOpenNodeInfo:



Collaboration diagram for CanOpenNodeInfo:



Public Attributes

- uint32 eventTime

  Time (milliseconds) when next guarding event is due.

- int16 guardToggle

  This value keeps track of the toggle bit used with node guarding.
- int32 guardTimeout

  This variable gives the guard time in milliseconds.
- uint8 lifeTime

  Max number of guard timeouts before an error is generated.
- uint8 lifeCounter

  Counter used to track guard timeouts.
- GuardProtocol guardType

  This variable identifies the type of guard protocol being used by this node.
- NodeState desired

  State I'm waiting for.
- Semaphore * semPtr

  Pointer to semaphore used when waiting for state change.
- Semaphore * sdoSemPtr

  Pointer to semaphore used when waiting for an SDO response.
- uint8 * sdoBuff

  Pointer to SDO data buffer.

5.26.1 Detailed Description

The CanOpenNodeInfo structure holds some data required by the CANopen network interface which is present in every node it manages.

The contents of this structure should be considered the private property of the CANopen class.

5.26.2 Member Data Documentation

5.26.2.1 guardTimeout

```
int32 guardTimeout
```

This variable gives the guard time in milliseconds.

It's used both in heartbeat and node guarding modes.

5.26.2.2 guardToggle

```
int16 guardToggle
```

This value keeps track of the toggle bit used with node guarding.

It's set to -1 if the toggle bit isn't used.

5.26.2.3 guardType

```
GuardProtocol guardType
```

This variable identifies the type of guard protocol being used by this node.

The options are none, heartbeat monitor, and node guarding.

The documentation for this struct was generated from the following files:

- CML_CanOpen.h
- CanOpen.cpp

5.27 CanOpenSettings Class Reference

Configuration object used to customize global settings for the CANopen network.

Public Member Functions

- CanOpenSettings ()

  Default constructor for CanOpenSettings object.

Public Attributes

- int readThreadPriority

  Defines the read thread priority.
- bool useAsTimingReference

  If true, the master (i.e.
- uint32 syncID

  CAN message ID of the SYNC message.
- uint32 timeID

  CAN message ID of the timestamp message.

5.27.1 Detailed Description

Configuration object used to customize global settings for the CANopen network.

An object of this type may be passed to the CanOpen::Open() method when the network is first opened.

If no CanOpenSettings object is passed to the CanOpen::Open() method, then the behavior is exactly the same as passing a CanOpenSettings object with the default settings.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 CanOpenSettings()

```
CanOpenSettings (
            void  )
```

Default constructor for CanOpenSettings object.

This constructor simply sets all the settings to their default values.

5.27.3 Member Data Documentation

5.27.3.1 readThreadPriority

```
int readThreadPriority
```

Defines the read thread priority.

The read thread is started when the CanOpen object is first opened (using CanOpen::Open()). This thread is responsible for reading messages from the CANopen network and calling the message handlers associated with them. It should be run at a relatively high priority. Default: 9

5.27.3.2 syncID

```
uint32 syncID
```

CAN message ID of the SYNC message.

This is only used if the CanOpen object wll act as the timing reference in the system. If so, this ID must match the corresponding value passed in the AmpSettings object. Default 0x80

5.27.3.3 timeID

```
uint32 timeID
```

CAN message ID of the timestamp message.

This is only used if the CanOpen object wll act as the timing reference in the system. If so, this ID must match the corresponding value passed in the AmpSettings object. Default 0x180

5.27.3.4 useAsTimingReference

```
bool useAsTimingReference
```

If true, the master (i.e.

the computer that CML is running on) will generate sync timing messages used to synchronize the clocks of the nodes on the CANopen network. This is only possible if the CAN interface being used is able to capture accurate PC timer based time stamps of the received CAN frames. Since this isn't supported by most CAN interfaces, the default setting for this is false. If false, one of the nodes on the CANopen network will generate the time stamps. Default: false

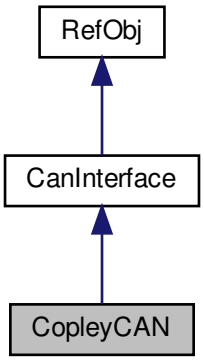
The documentation for this class was generated from the following files:

- CML_CanOpen.h
- CanOpen.cpp

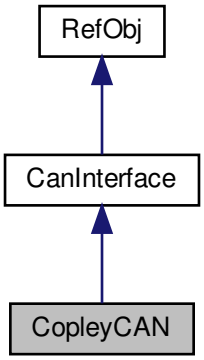
5.28 CopleyCAN Class Reference

This class extends the generic CanInterface class into a working interface for the Copley can device driver.

Inheritance diagram for CopleyCAN:



Collaboration diagram for CopleyCAN:



Public Member Functions

- CopleyCAN (void)

  Construct a CAN object.
- CopleyCAN (const char *port)

  Construct a new CAN object.
- virtual ~CopleyCAN (void)

  Destructor for CopleyCAN object.
- const Error * Open (void)

  Open the CAN bus.
- const Error * Close (void)

  Close the CAN interface.
- const Error * SetBaud (int32 baud)

  Set the CAN interface baud rate.
- bool SupportsTimestamps (void)

  Return true if the CAN interface supports timestamps on received frames.

Protected Member Functions

- const Error * RecvFrame (CanFrame &frame, Timeout timeout)

  Receive the next CAN frame.
- const Error * XmitFrame (CanFrame &frame, Timeout timeout)

  Write a CAN frame to the CAN network.

Protected Attributes

- int open

  tracks the state of the interface as open or closed.
- int32 baud

  Holds a copy of the last baud rate set.
- void * local

  This pointer is used to keep track of private data used by the driver.

Additional Inherited Members

5.28.1 Detailed Description

This class extends the generic CanInterface class into a working interface for the Copley can device driver.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 CopleyCAN() [1/2]

```
CopleyCAN (
            void  )
```

Construct a CAN object.

Calls CanInterface constructor.

5.28.2.2 CopleyCAN() [2/2]

```
CopleyCAN (
            const char * port )
```

Construct a new CAN object.

Calls CanInterface constuctor.

5.28.3 Member Function Documentation

5.28.3.1 Close()

```
const Error * Close (
            void  )  [virtual]
```

Close the CAN interface.

Returns

A CAN error object identifying the error.

Reimplemented from CanInterface.

5.28.3.2 Open()

```
const Error * Open (
            void  )  [virtual]
```

Open the CAN bus.

Returns

A CAN error object identifying the error.

Reimplemented from CanInterface.

5.28.3.3 RecvFrame()

```
const Error * RecvFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Receive the next CAN frame.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A CAN error object identifying the error.

Reimplemented from CanInterface.

5.28.3.4 SetBaud()

```
const Error * SetBaud (
            int32 b )  [virtual]
```

Set the CAN interface baud rate.

This should be set before the CAN interface is open.

Parameters

b	The baud rate to set.

Returns

A CAN error object identifying the error.

Reimplemented from CanInterface.

5.28.3.5 SupportsTimestamps()

```
bool SupportsTimestamps (
            void  )  [inline], [virtual]
```

Return true if the CAN interface supports timestamps on received frames.

Returns

true if timestamps are supported

Reimplemented from CanInterface.

5.28.3.6 XmitFrame()

```
const Error * XmitFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Write a CAN frame to the CAN network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A CAN error object identifying the error.

Reimplemented from CanInterface.

5.28.4 Member Data Documentation

5.28.4.1 local

```
void* local  [protected]
```

This pointer is used to keep track of private data used by the driver.

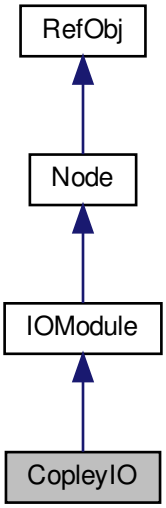
The documentation for this class was generated from the following files:

- can_copley.h
- can_copley.cpp

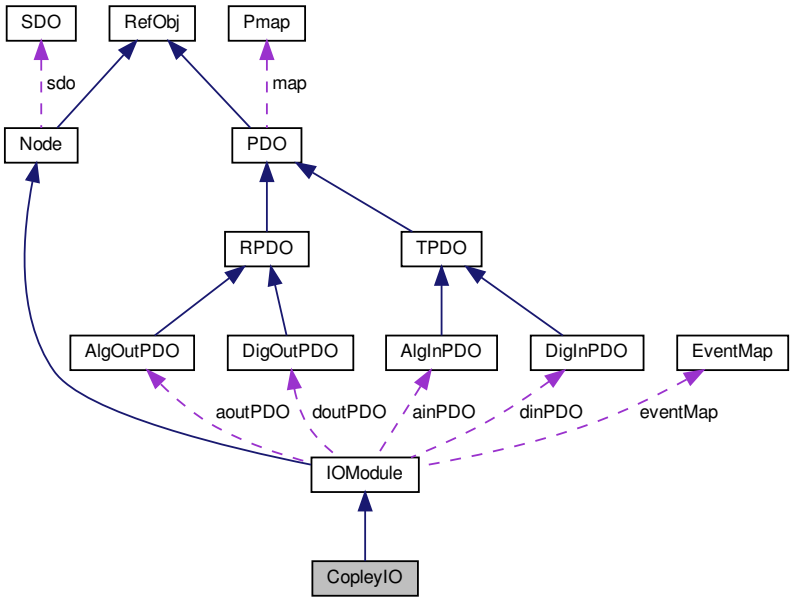
5.29 CopleyIO Class Reference

This class represents a Copley CANopen I/O module.

Inheritance diagram for CopleyIO:



Collaboration diagram for CopleyIO:



Public Member Functions

- CopleyIO (void)

  Default constructor for a Copley I/O module.
- CopleyIO (Network &net, int16 nodeID)

  Construct a CopleyIO object and initialize it using default settings.
- CopleyIO (Network &net, int16 nodeID, IOModuleSettings &settings)

  Construct a CopleyIO object and initialize it using custom settings.
- virtual ~CopleyIO ()

  Virtual destructor for the IOModule object.
- virtual const Error * Init (Network &net, int16 nodeID)

  Initialize a Copley IO module using default settings.
- virtual const Error * Init (Network &net, int16 nodeID, IOModuleSettings &settings)

  Initialize an I/O module using custom settings.
- const Error * GetIOInfo (CopleyIOInfo &info)

  Read the I/O Module information parameters from the drive.
- const Error * GetIODigi (CopleyIODigi &digi)

  Read the I/O Module digital parameters from the drive.
- const Error * GetIOAnlg (CopleyIOAnlg &anlg)

  Read the I/O Module analog parameters from the drive.
- const Error * GetIOPWM (CopleyIOPWM &pwm, CopleyIOInfo &info)

  Read the I/O Module PWM parameters from the drive.
- const Error * GetIOCfg (CopleyIOCfg &cfg)

  Read the complete I/O configuration from the module and return it in the passed structure.
- const Error * SetIOInfo (CopleyIOInfo &info)

  Write the I/O Module information parameters to the drive.
- const Error * SetIODigi (CopleyIODigi &digi)

  Write the I/O Module digital parameters to the drive.
- const Error * SetIOAnlg (CopleyIOAnlg &anlg)

  Write the I/O Module analog parameters to the drive.
- const Error * SetIOPWM (CopleyIOPWM &pwm, CopleyIOInfo &info)

  Write the I/O Module PWM parameters to the drive.
- const Error * SetIOConfig (CopleyIOCfg &cfg)

  Write the complete I/O configuration from the module and return it in the passed structure.
- const Error * SaveIOConfig (void)

  Save all I/O parameters to internal flash memory.
- const Error * SaveIOConfig (CopleyIOCfg &cfg)

  Upload the passed io module configuration to the module's working memory, and then copy that working memory to flash.
- const Error * LoadFromFile (const char *name, int &line)

  Load the specified io module data file.
- const Error * SerialCmd (uint8 opcode, uint8 &ct, uint8 max, uint16 *data)

  Send a serial port message to a Copley device over the CANopen network.

Additional Inherited Members

5.29.1 Detailed Description

This class represents a Copley CANopen I/O module.

It extendes the standard I/O module with methods that can be used to restore from a CME generated I/O settings file.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 CopleyIO() [1/2]

```
CopleyIO (
            Network & net,
            int16 nodeID )
```

Construct a CopleyIO object and initialize it using default settings.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.

5.29.2.2 CopleyIO() [2/2]

```
CopleyIO (
            Network & net,
            int16 nodeID,
            IOModuleSettings & settings )
```

Construct a CopleyIO object and initialize it using custom settings.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.
settings	The settings to use when configuring the module

5.29.3 Member Function Documentation

5.29.3.1 GetIOAnlg()

```
const Error * GetIOAnlg (
            CopleyIOAnlg & anlg )
```

Read the I/O Module analog parameters from the drive.

Parameters

anlg	A structure that will be filled with the analog I/O values

Returns

A pointer to an error object, or NULL on success

5.29.3.2 GetIOCfg()

```
const Error * GetIOCfg (
            CopleyIOCfg & cfg )
```

Read the complete I/O configuration from the module and return it in the passed structure.

This structure holds every module parameter that can be stored to the module's internal flash memory. The contents of the structure represent the complete I/O configuration.

Parameters

cfg	The structure which will hold the uploaded configuration.

Returns

A pointer to an error object, or NULL on success

5.29.3.3 GetIODigi()

```
const Error * GetIODigi (
            CopleyIODigi & digi )
```

Read the I/O Module digital parameters from the drive.

Parameters

digi	A structure that will be filled with the digital I/O values

Returns

A pointer to an error object, or NULL on success

5.29.3.4 GetIOInfo()

```
const Error * GetIOInfo (
            CopleyIOInfo & info )
```

Read the I/O Module information parameters from the drive.

Parameters

info	A structure that will be filled with the I/O module info

Returns

A pointer to an error object, or NULL on success

5.29.3.5 GetIOPWM()

```
const Error * GetIOPWM (
            CopleyIOPWM & pwm,
            CopleyIOInfo & info )
```

Read the I/O Module PWM parameters from the drive.

Parameters

pwm	A structure that will be filled with the PWM I/O values
info	A structure containing additional information about the module.

Returns

A pointer to an error object, or NULL on success

5.29.3.6 Init() [1/2]

```
const Error * Init (
            Network & net,
            int16 nodeID )  [virtual]
```

Initialize a Copley IO module using default settings.

This function associates the object with the CANopen network it will be used on.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.

Returns

A pointer to an error object, or NULL on success

Reimplemented from IOModule.

5.29.3.7 Init() [2/2]

```
const Error * Init (
            Network & net,
            int16 nodeID,
            IOModuleSettings & settings )  [virtual]
```

Initialize an I/O module using custom settings.

This function associates the object with the CANopen network it will be used on.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.
settings	The settings to use when configuring the module

Returns

A pointer to an error object, or NULL on success

Reimplemented from IOModule.

5.29.3.8 LoadFromFile()

```
const Error * LoadFromFile (
            const char * name,
            int & line )
```

Load the specified io module data file.

This function presently supports loading *.cci files created by the CME-2 program, version 1 and later.

Parameters

name	The name (and optionally path) of the file to load
line	If not NULL, the last line number read from the file is returned here. This is useful for finding file format errors.

Returns

A pointer to an error object, or NULL on success.

5.29.3.9 SaveIOConfig() [1/2]

```
const Error * SaveIOConfig (
            void  )
```

Save all I/O parameters to internal flash memory.

Flash memory is a type of non-volatile RAM which allows module parameters to be saved between power cycles. When this function is called, any module parameters that may be stored to flash will be copied from their working (RAM) locations to the stored (flash) locations.

For a list of those I/O parameters which may be saved to flash memory, see the IOConfig structure. Every member of that structure represents an io module parameter that may be saved to flash.

Returns

A pointer to an error object, or NULL on success

5.29.3.10 SaveIOConfig() [2/2]

```
const Error * SaveIOConfig (
            CopleyIOCfg & cfg )
```

Upload the passed io module configuration to the module's working memory, and then copy that working memory to flash.

Parameters

cfg	The structure which holds the new configuration.

Returns

A pointer to an error object, or NULL on success

5.29.3.11 SerialCmd()

```
const Error * SerialCmd (
            uint8 opcode,
            uint8 & ct,
            uint8 max,
            uint16 * data )
```

Send a serial port message to a Copley device over the CANopen network.

Copley devices use serial ports for some basic configuration purposes. Most of the functions available over the serial interface are also available in the CANopen object dictionary, but not everything.

This function allows a native serial port command to be sent over the CANopen network. It allows any remaining features of the device to be accessed when only a CANopen connection is available.

Parameters

opcode	The command code to be sent to the amplifier.
ct	The number of 16-bit words of data to be sent to the amplifier. On return, this parameter will hold the number of 16-bit words of response data passed back from the amplifier.
max	The maximum number of words of response data that the data array can hold.
data	An array of data to be passed to the node with the command. On return, any response data (up to max words) will be passed here. If this parameter is not specified, then no data will be passed or returned regardless of the values passed in max and ct.

Returns

An error object, or null on success.

5.29.3.12 SetIOAnlg()

```
const Error * SetIOAnlg (
            CopleyIOAnlg & anlg )
```

Write the I/O Module analog parameters to the drive.

Parameters

anlg	A structure that will be filled with the analog I/O values

Returns

A pointer to an error object, or NULL on success

5.29.3.13 SetIOConfig()

```
const Error * SetIOConfig (
            CopleyIOCfg & cfg )
```

Write the complete I/O configuration from the module and return it in the passed structure.

This structure holds every I/O parameter that can be stored to the module's internal flash memory. The contents of the structure represent the complete I/O configuration.

Parameters

cfg	The structure which will hold the uploaded configuration.

Returns

A pointer to an error object, or NULL on success

5.29.3.14 SetIODigi()

```
const Error * SetIODigi (
            CopleyIODigi & digi )
```

Write the I/O Module digital parameters to the drive.

Parameters

digi	A structure that will be filled with the digital I/O values

Returns

A pointer to an error object, or NULL on success

5.29.3.15 SetIOInfo()

```
const Error * SetIOInfo (
            CopleyIOInfo & info )
```

Write the I/O Module information parameters to the drive.

Parameters

info	A structure that will be filled with the I/O module info

Returns

A pointer to an error object, or NULL on success

5.29.3.16 SetIOPWM()

```
const Error * SetIOPWM (
            CopleyIOPWM & pwm,
            CopleyIOInfo & info )
```

Write the I/O Module PWM parameters to the drive.

Parameters

pwm	A structure that will be filled with the PWM I/O values
info	A structure containing additional information about the module.

Returns

A pointer to an error object, or NULL on success

The documentation for this class was generated from the following files:

- CML_CopleyIO.h
- CopleyIO.cpp
- CopleyIOFile.cpp

5.30 CopleyIOAnIg Struct Reference

This structure is used to return information about the analog inputs of a Copley I/O module.

Public Attributes

- uint16 iRaw [COPLEYIO_NUM_AIN]
  Analog input raw value.
- uint32 iScaled [COPLEYIO_NUM_AIN]
  Analog input scaled value.
- uint32 iFactor [COPLEYIO_NUM_AIN]
  Analog input scaling factor.
- uint32 iOffset [COPLEYIO_NUM_AIN]
  Analog input offset.
- uint32 iUpLimit [COPLEYIO_NUM_AIN]
  Analog input upper limit for interrupt.
- uint32 iLoLimit [COPLEYIO_NUM_AIN]
  Analog input lower limit for interrupt.
- uint32 iAbsDelta [COPLEYIO_NUM_AIN]
  Analog input absolute delta value for interrrupt.
- uint32 iPosDelta [COPLEYIO_NUM_AIN]
  Analog input positive delta value for interrrupt.
- uint32 iNegDelta [COPLEYIO_NUM_AIN]
  Analog input negative delta value for interrrupt.
- uint16 iFlags [COPLEYIO_NUM_AIN]
  Analog input interrrupt flags.
- uint16 iMask [COPLEYIO_NUM_AIN]
  Analog input interrrupt mask.

5.30.1 Detailed Description

This structure is used to return information about the analog inputs of a Copley I/O module.

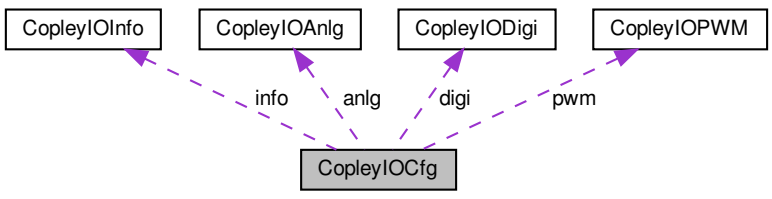
The documentation for this struct was generated from the following file:

- CML_CopleyIO.h

5.31 CopleyIOCfg Struct Reference

IO Module configuration structure.

Collaboration diagram for CopleyIOCfg:



Public Attributes

- CopleyIOInfo info
  Global IO Module parameters.
- CopleyIODigi digi
  Digital IO parameters.
- CopleyIOAnlg anlg
  Analog input parameters.
- CopleyIOPWM pwm
  PWM output parameters.

5.31.1 Detailed Description

IO Module configuration structure.

This structure contains all user configurable parameters used by an IO module which may be stored in non-volatile memory.

The documentation for this struct was generated from the following file:

- CML_CopleyIO.h

5.32 CopleyIODigi Struct Reference

This structure is used to return information about the digital I/O of a Copley I/O module.

Public Attributes

- uint16 bankMode [COPLEYIO_DIO_BANKS]
  Digital I/O bank mode.
- uint16 pullupMsk [COPLEYIO_DIO_BANKS]
  Digital I/O pull-up resistor mask.
- uint16 typeMsk [COPLEYIO_DIO_BANKS]
  Digital I/O output type mask.
- uint16 faultMsk [COPLEYIO_DIO_BANKS]
  Digital I/O output fault state mask.
- uint16 invMsk [COPLEYIO_DIO_BANKS]
  Digital I/O inversion mask.
- uint16 valueMsk [COPLEYIO_DIO_BANKS]
  Digital I/O data value mask.
- uint16 modeMsk [COPLEYIO_DIO_BANKS]
  Digital I/O output fault mode mask.
- uint16 rawMsk [COPLEYIO_DIO_BANKS]
  Digital I/O raw data value mask.

- uint16 hiLoMsk [COPLEYIO_DIO_BANKS]
  Digital I/O input low->high interrupt mask.
- uint16 loHiMsk [COPLEYIO_DIO_BANKS]
  Digital I/O input high->low interrupt mask.
- uint16 debounce0 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 0.
- uint16 debounce1 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 1.
- uint16 debounce2 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 2.
- uint16 debounce3 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 3.
- uint16 debounce4 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 4.
- uint16 debounce5 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 5.
- uint16 debounce6 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 6.
- uint16 debounce7 [COPLEYIO_DIO_BANKS]
  Digital I/O debounce time, bit 7.

5.32.1 Detailed Description

This structure is used to return information about the digital I/O of a Copley I/O module.

The documentation for this struct was generated from the following file:

- CML_CopleyIO.h

5.33 CopleyIOInfo Struct Reference

IO Module characteristics data structure.

Public Attributes

- uint32 serial
  Serial number.
- char model [COPLEYIO_MAX_STRING]
  Model number string.
- char mfgInfo [COPLEYIO_MAX_STRING]
  Amplifier's manufacturing information string.
- uint16 hwType
  Hardware type code.

- uint16 loopRate

  Main loop update rate (Hz)
- uint16 fwVersion

  Firmware version number.
- uint32 baud

  Serial port baud rate (bps)
- uint16 maxWords

  Maximum number of words sent with any command.
- char name [COPLEYIO_MAX_STRING]

  I/O module name.
- char hostCfg [COPLEYIO_MAX_STRING]

  Host configuration state (CME use only)
- int16 nodeCfg

  CAN node ID configuration.
- uint16 rateCfg

  CAN bit rate configuration.
- uint16 nodeID

  CAN node ID.
- uint16 status

  CAN network status word.
- uint16 rate

  CAN network bit rate.
- uint16 anlgInt

  Active analog interrupts.
- uint16 anlgIntEna

  Analog input global interrupt enable.
- uint16 digiIntEna

  Digital input global interrupt enable.
- uint32 pwmPeriodA

  PWM bank A period.
- uint32 pwmPeriodB

  PWM bank B period.

5.33.1 Detailed Description

IO Module characteristics data structure.

This structure is used to hold information about the IO Module such as it's model number, serial number, etc.

Use the IOModule::GetIOInfo method to retrieve this information from the module.

The documentation for this struct was generated from the following file:

- CML_CopleyIO.h

5.34 CopleyIOPWM Struct Reference

This structure is used to return information about the PWM outputs of a Copley I/O module.

Public Attributes

- uint16 oRaw [COPLEYIO_NUM_PWM]

  PWM output raw value.
- uint32 oScaled [COPLEYIO_NUM_PWM]

  PWM output scaled value.
- uint32 oFactor [COPLEYIO_NUM_PWM]

  PWM output scaling factor.
- int32 oOffset [COPLEYIO_NUM_PWM]

  PWM output offset.

5.34.1 Detailed Description

This structure is used to return information about the PWM outputs of a Copley I/O module.

The documentation for this struct was generated from the following file:

- CML_CopleyIO.h

5.35 CopleyMotionLibrary Class Reference

Copley Motion Libraries utility object.

Public Member Functions

- CopleyMotionLibrary ()

  Default constructor for the CopleyMotionLibrary object.
- ~CopleyMotionLibrary ()

  Destructor for CopleyMotionLibrary object.
- const char * GetVersionString ()

  Get the CML library version string.
- void SetDebugLevel (CML_LOG_LEVEL level)

  Set the debug message level.
- void SetFlushLog (bool flush)

  Enable/Disable the flushing of the log file after each write.
- void SetMaxLogSize (int32 max)

  Set the max CML log file size.
- void FlushLog (void)

Flush the log file (if one is open).

- void SetLogFile (const char *fname)

  Set the debug message log file name.
- void Debug (const char *fmt,...)

  Write a debug message to the log file.
- void Warn (const char *fmt,...)

  Write a warning message to the log file.
- void Error (const char *fmt,...)

  Write an error message to the log file.
- void LogCAN (bool recv, struct CanFrame &frame)

  Write the CAN frame to the log file.
- CML_LOG_LEVEL GetDebugLevel (void)

  Return the debug level.
- bool GetFlushLog (void)

  Get the state of the log flushing setting.
- int32 GetMaxLogSize (void)

  Return the max log size.
- const char * GetLogFile (void)

  Return the name of the log file.

5.35.1 Detailed Description

Copley Motion Libraries utility object.

This object defines a number of handy methods related to the libraries as a whole.

A single global CML object is created by the libraries automatically.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 ~CopleyMotionLibrary()

```
~CopleyMotionLibrary ( )
```

Destructor for CopleyMotionLibrary object.

This simply closes the log file if one is open.

5.35.3 Member Function Documentation

5.35.3.1 Debug()

```
void Debug (
            const char * fmt,
             ...  )
```

Write a debug message to the log file.

The debug level must be set >= LOG_DEBUG for this message to be written.

Note that message logging is only available if file support is enabled in the CML_Settings.h file.

Parameters

fmt	A printf style format string

5.35.3.2 Error()

```
void Error (
            const char * fmt,
             ...  )
```

Write an error message to the log file.

The debug level must be set >= LOG_ERRORS for this message to be written.

Note that message logging is only available if file support is enabled in the CML_Settings.h file.

Parameters

fmt	A printf style format string

5.35.3.3 FlushLog()

```
void FlushLog (
            void  )
```

Flush the log file (if one is open).

This forces the log contents to be written to disk, thus preventing it from being lost if the program exits without calling the CML object destructor.

5.35.3.4 GetDebugLevel()

```
CML_LOG_LEVEL GetDebugLevel (
             void  )  [inline]
```

Return the debug level.

Returns

The debug level presently set

5.35.3.5 GetFlushLog()

```
bool GetFlushLog (
             void  )  [inline]
```

Get the state of the log flushing setting.

Returns

true if enabled.

5.35.3.6 GetLogFile()

```
const char* GetLogFile (
             void  )  [inline]
```

Return the name of the log file.

Returns

The log file name as a zero terminated string

5.35.3.7 GetMaxLogSize()

```
int32 GetMaxLogSize (
             void  )  [inline]
```

Return the max log size.

Returns

The max log size in bytes

5.35.3.8 GetVersionString()

```
const char * GetVersionString (
             void  )
```

Get the CML library version string.

Returns

A string giving the CML version number

5.35.3.9 LogCAN()

```
void LogCAN (
             bool recv,
             struct CanFrame & frame )
```

Write the CAN frame to the log file.

The log level must be at least LOG_FILT_CAN for this to be written.

Parameters

recv	True if this was a received message, false for transmit messages
frame	The frame to log

5.35.3.10 SetDebugLevel()

```
void SetDebugLevel (
             CML_LOG_LEVEL level )
```

Set the debug message level.

The library code includes some debug messages that may be enabled by setting the debug level to a value greater then LOG_NONE.

If the level was previously set higher then LOG_NONE, and is then set to LOG_NONE, any open log file will be closed.

The default log level is LOG_NONE (no messages).

Parameters

level	The log level.

5.35.3.11 SetFlushLog()

```
void SetFlushLog (
            bool flush )
```

Enable/Disable the flushing of the log file after each write.

This can be useful if debug messages are being lost due to a crash, however it can increase the time necessary to write to the file.

The default is false (don't flush).

Parameters

flush	True if file flushing is desired

5.35.3.12 SetLogFile()

```
void SetLogFile (
            const char * fname )
```

Set the debug message log file name.

This file will be used to log debug messages. The file will be created (or truncated if it alread exists) when the first message is written to the file. Note that the debug level must be set $>$ LOG_NONE for any messages to be written. Also note that file access must be enabled in CML_Settings.h for the log file to be used.

If the log file is already open when this method is called, it will be closed and a new log file with the specified name will be open on the next write to it.

The default log file name is "cml.log"

Parameters

fname	The log file name

5.35.3.13 SetMaxLogSize()

```
void SetMaxLogSize (
            int32 max )
```

Set the max CML log file size.

This option can be used to prevent a debug log file from getting so large it uses all available disk space on long runs. Once the log file exceeds this size, it will be renamed logfilename.bak (where logfilename is replaced by the log file name), and a new log file will be started. Any old backup log file will be overwritten.

The default max log size is 1,000,000 bytes.

Parameters

max	Maximum log file size in bytes

5.35.3.14 Warn()

```
void Warn (
            const char * fmt,
             ...  )
```

Write a warning message to the log file.

The debug level must be set >= LOG_WARNINGS for this message to be written.

Note that message logging is only available if file support is enabled in the CML_Settings.h file.

Parameters

fmt	A printf style format string

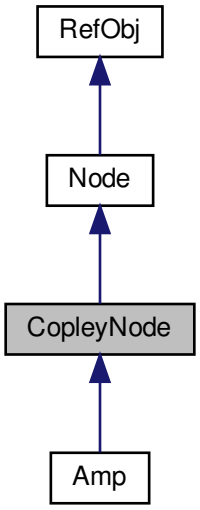
The documentation for this class was generated from the following files:

- CML.h
- CML.cpp

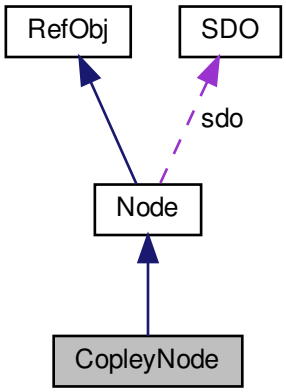
5.36 CopleyNode Class Reference

Copley CANopen Node class.

Inheritance diagram for CopleyNode:



Collaboration diagram for CopleyNode:



Public Member Functions

- const Error * FirmwareUpdate (Firmware &fw)

Use a special protocol to update the firmware in Copley CANopen devices.

- const Error * SerialCmd (uint8 opcode, uint8 &ct, uint8 max=0, uint16 *data=0)

  Send a serial port message to a Copley device over the CANopen network.

Additional Inherited Members

5.36.1 Detailed Description

Copley CANopen Node class.

Objects of this class represent Copley products attached to the CANopen bus.

5.36.2 Member Function Documentation

5.36.2.1 FirmwareUpdate()

```
const Error * FirmwareUpdate (
            Firmware & fw )
```

Use a special protocol to update the firmware in Copley CANopen devices.

Note that this is not part of normal operation, but rather a special utility function which allows the device firmware to be updated if necessary.

Only firmware files produced by Copley Controls Corp. should be used to update Copley devices. Attempting to update a device with incorrectly formatted firmware files will render the it inoperable.

If an error occurs during the download of new firmware, it may be necessary to reprogram the device through it's serial port. The CME-2 software can be used to recover an device in this case.

Parameters

fw	The firmware object holding the data to be programmed.

Returns

An error object, or NULL on success.

5.36.2.2 SerialCmd()

```
const Error * SerialCmd (
            uint8 opcode,
            uint8 & ct,
            uint8 max = 0,
            uint16 * data = 0 )
```

Send a serial port message to a Copley device over the CANopen network.

Copley devices use serial ports for some basic configuration purposes. Most of the functions available over the serial interface are also available in the CANopen object dictionary, but not everything.

This function allows a native serial port command to be sent over the CANopen network. It allows any remaining features of the device to be accessed when only a CANopen connection is available.

Parameters

opcode	The command code to be sent to the amplifier.
ct	The number of 16-bit words of data to be sent to the amplifier. On return, this parameter will hold the number of 16-bit words of response data passed back from the amplifier.
max	The maximum number of words of response data that the data array can hold.
data	An array of data to be passed to the node with the command. On return, any response data (up to max words) will be passed here. If this parameter is not specified, then no data will be passed or returned regardless of the values passed in max and ct.

Returns

An error object, or null on success.

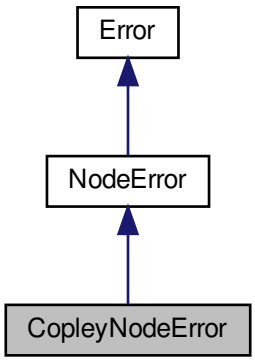
The documentation for this class was generated from the following files:

- CML_Copley.h
- AmpFW.cpp
- CopleyNode.cpp

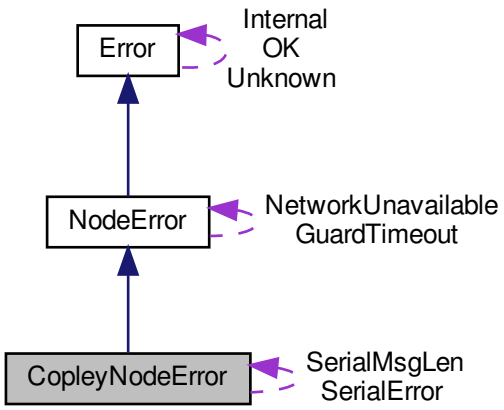
5.37 CopleyNodeError Class Reference

This class represents errors that can be returned by the CopleyNode class.

Inheritance diagram for CopleyNodeError:



Collaboration diagram for CopleyNodeError:



Static Public Attributes

- static const CopleyNodeError SerialMsgLen

  Too much data passed for a serial port command.
- static const CopleyNodeError SerialError

  The device return a serial port error code.

Protected Member Functions

- CopleyNodeError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.37.1 Detailed Description

This class represents errors that can be returned by the CopleyNode class.

There is one static member for each defined error.

The documentation for this class was generated from the following file:

- CML_Copley.h

5.38 CrntLoopConfig Struct Reference

This structure holds the current loop configuration parameters.

Public Member Functions

- CrntLoopConfig (void)

  Default constructor.

Public Attributes

- int16 kp

  Proportional gain.
- int16 ki

  Integral gain.
- int16 offset

  Current offset.
- int16 peakLim

  Peak current limit (0.01 amp units) This is the maximum current that can be applied to the motor at any time Also used as Boost current in stepper mode.
- int16 contLim

  Continuous current limit (0.01 amp units) This is the maximum current that can continuously be applied to the load.
- int16 peakTime

  Time at peak current limit (milliseconds) If peak current is requested, it will fall back to the continuous limit within this amount of time.
- uint16 stepHoldCurrent

  Stepper hold current (0.01 amps).
- uint16 stepRun2HoldTime

  Run to hold time(milliseconds) The period of time, beginning when a move is complete, to when output current switched to hold current.
- uint16 stepVolControlDelayTime

  Voltage control mode time delay (milliseconds) Time delay to enter into a special voltage control mode.
- int32 slope

  Rate of change of current command (milliamps/sec).

5.38.1 Detailed Description

This structure holds the current loop configuration parameters.

The current loop is one of three servo control loops used by the amplifier to control a motor. The configuration parameters used by this control loop allow the servo performance to be 'tuned' for various motors and loads.

This structure also holds the parameters used to control current limiting. The current limiting acts on the commanded current before it is sent to the current loop.

The amplifier member functions Amp::GetCrntLoopConfig and Amp::SetCrntLoopConfig are used to read and write this data to the amplifier.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 CrntLoopConfig()

```
CrntLoopConfig (
            void  )  [inline]
```

Default constructor.

Simply initializes all members to zero.

5.38.3 Member Data Documentation

5.38.3.1 contLim

```
int16 contLim
```

Continuous current limit (0.01 amp units) This is the maximum current that can continuously be applied to the load.

Also used as Run current in stepper mode.

5.38.3.2 peakTime

```
int16 peakTime
```

Time at peak current limit (milliseconds) If peak current is requested, it will fall back to the continuous limit within this amount of time.

Also used as Boost current time in stepper mode.

5.38.3.3 slope

```
int32 slope
```

Rate of change of current command (milliamps/sec).

This parameter is only used when running in the low level programmed current mode (AMPMODE_PROG_CRNT), or in the CANopen profile torque mode (AMPMODE_CAN_TORQUE). In other modes this parameter is ignored and no limit is placed on the slope of the current command.

If this parameter is set to zero (default) it is not used in any mode of operation.

Note that this parameter is internally the same as the torque slope parameter which can be set using the function Amp↔ ::SetTorqueSlope. The units are different however as this parameter controls slope in units of current and the torque slope function adjusts in units of torque.

5.38.3.4 stepHoldCurrent

```
uint16 stepHoldCurrent
```

Stepper hold current (0.01 amps).

Current used to hold the motor at rest. Used in stepper mode only.

5.38.3.5 stepRun2HoldTime

```
uint16 stepRun2HoldTime
```

Run to hold time(milliseconds) The period of time, beginning when a move is complete, to when output current switched to hold current.

Used in stepper mode only.

5.38.3.6 stepVolControlDelayTime

```
uint16 stepVolControlDelayTime
```

Voltage control mode time delay (milliseconds) Time delay to enter into a special voltage control mode.

If set to zero this feature is disabled. Used for stepper mode only.

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.39 DAConfig Struct Reference

Configuration structure used to hold the settings for a drive's D/A converter.

Public Member Functions

- DAConfig (void)

  Default constructor Initialize all elements to zero.

Public Attributes

- int32 daConverterConfig

  D/A converter configuration.

5.39.1 Detailed Description

Configuration structure used to hold the settings for a drive's D/A converter.

These settings may be up/download from the amplifier using the functions Amp::SetDAConverterConfig and Amp::↩ GetDAConverterConfig

5.39.2 Member Data Documentation

5.39.2.1 daConverterConfig

```
int32 daConverterConfig
```

D/A converter configuration.

This paramter sets the mode for the D/A converter on drives so equipped. See CAN ID 0x21E0 for details.

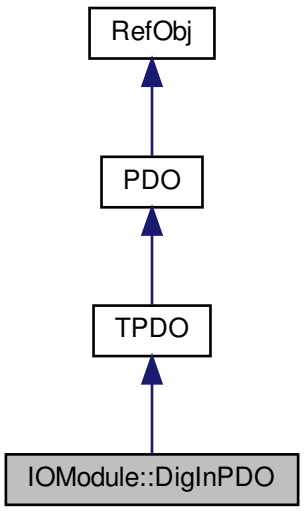
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

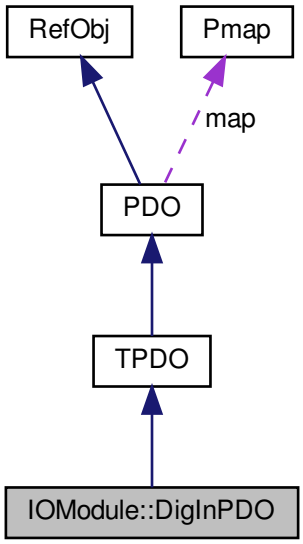
5.40 IOModule::DigInPDO Class Reference

Transmit PDO for mapping digital inputs.

Inheritance diagram for IOModule::DigInPDO:



Collaboration diagram for IOModule::DigInPDO:



Public Member Functions

- const Error * Init (class IOModule *io, uint32 cobID, uint8 ct, uint8 id[ ], IOMODULE_EVENTS event)

  Initialize a digital input PDO object.
- bool GetInVal (uint8 id, uint8 &value)

  Read the specified input bank from the PDO's cached data.
- bool GetBitVal (uint16 id, bool &value)

  Update the locally stored value of one bit in this PDO.
- void Received (void)

  New transmit PDO received.

Additional Inherited Members

5.40.1 Detailed Description

Transmit PDO for mapping digital inputs.

This class represents the standard transmit PDO into which up to 64 digital inputs may be mapped.

5.40.2 Member Function Documentation

5.40.2.1 GetBitVal()

```
bool GetBitVal (
            uint16 id,
            bool & value )
```

Update the locally stored value of one bit in this PDO.

Parameters

id	The output ID to be updatad.
value	The new value for the output.

Returns

true if the value was updated, false if the output isn't mapped to this PDO.

5.40.2.2 GetInVal()

```
bool GetInVal (
            uint8 id,
            uint8 & value )
```

Read the specified input bank from the PDO's cached data.

The value returned will be the last value received via PDO for this input bank.

Parameters

id	The input block ID to be checked.
value	The input value for the block will be returned here.

Returns

true if the value was returned, false if the block isn't mapped to this PDO.

5.40.2.3 Init()

```
const Error * Init (
            class IOModule * io,
            uint32 cobID,
```

```
            uint8 ct,
            uint8 id[],
            IOMODULE_EVENTS event )
```

Initialize a digital input PDO object.

Parameters

io	Pointer to the I/O module to which this PDO is assigned.
cobID	The CAN ID for this PDO message.
ct	The number of input blocks to be mapped (1 to 8)
id	An array of ct input block ID numbers. These will be mapped (in order) to the PDO.
event	The event bit to post when a PDO message is received.

Returns

A pointer to an error object, or NULL on success

5.40.2.4 Received()

```
void Received (
            void  )  [virtual]
```

New transmit PDO received.

This method is called by the CANopen reader thread when a new PDO message is received. It causes this PDO object to post it's event to the IOModule object's event map. This will cause any waiting threads to wake up.

Reimplemented from TPDO.

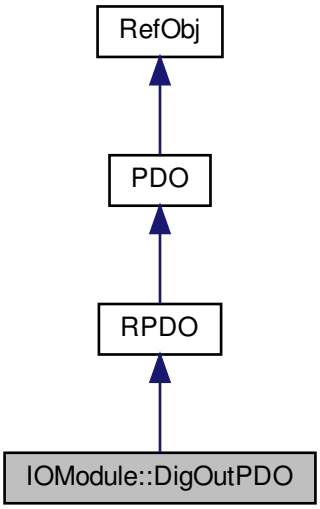
The documentation for this class was generated from the following files:

- CML_IO.h
- IOmodule.cpp

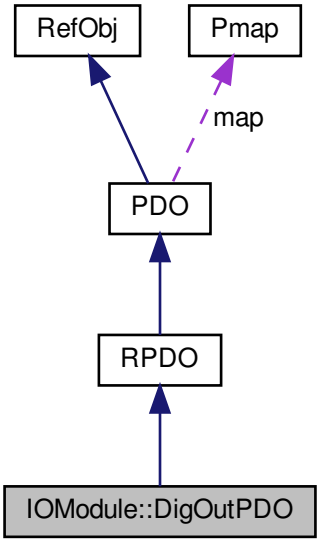
5.41 IOModule::DigOutPDO Class Reference

Receive PDO for mapping digital output pins.

Inheritance diagram for IOModule::DigOutPDO:



Collaboration diagram for IOModule::DigOutPDO:



Public Member Functions

- const Error * Init (class IOModule *io, uint32 cobID, uint8 ct, uint8 id[])

  Initialize a digital output PDO object.
- bool Update (uint8 id, uint8 value)

  Update the locally stored value of one of the 8-bit digital output blocks associated with this PDO.
- bool UpdateBit (uint16 id, bool value)

  Update the locally stored value of one bit in this PDO.
- const Error * Transmit (void)

  Transmit this PDO.

Additional Inherited Members

5.41.1 Detailed Description

Receive PDO for mapping digital output pins.

This class represents the standard receive PDO into which up to 64 digital output pins may be mapped.

5.41.2 Member Function Documentation

5.41.2.1 Init()

```
const Error * Init (
            class IOModule * io,
            uint32 cobID,
            uint8 ct,
            uint8 id[] )
```

Initialize a digital output PDO object.

Parameters

io	Pointer to the I/O module to which this PDO is assigned.
cobID	The CAN ID for this PDO message.
ct	The number of output blocks to be mapped (1 to 8)
id	An array of ct output block ID numbers. These will be mapped (in order) to the PDO.

Returns

A pointer to an error object, or NULL on success

5.41.2.2 Transmit()

```
const Error * Transmit (
            void  )
```

Transmit this PDO.

Returns

A pointer to an error object, or NULL on success

5.41.2.3 Update()

```
bool Update (
            uint8 id,
            uint8 value )
```

Update the locally stored value of one of the 8-bit digital output blocks associated with this PDO.

Parameters

id	The output block ID to be updatad.
value	The new value for the output block.

Returns

true if the value was updated, false if the block isn't mapped to this PDO.

5.41.2.4 UpdateBit()

```
bool UpdateBit (
            uint16 id,
            bool value )
```

Update the locally stored value of one bit in this PDO.

Parameters

id	The output ID to be updatad.
value	The new value for the output.

Returns

true if the value was updated, false if the output isn't mapped to this PDO.

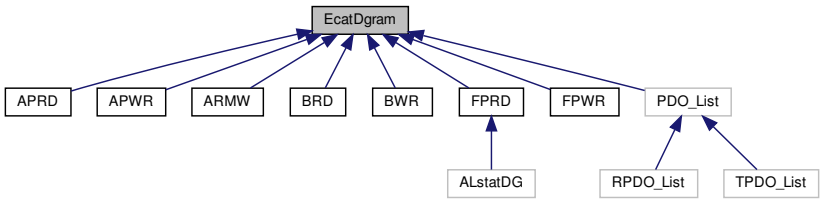
The documentation for this class was generated from the following files:

- CML_IO.h
- IOmodule.cpp

5.42 EcatDgram Class Reference

Generic EtherCAT datagram class.

Inheritance diagram for EcatDgram:



Public Member Functions

- EcatDgram (uint8 cmd, int16 adp, int16 ado, int16 len, void *ptr=0)
  Construct a datagram with a variable data length.
- EcatDgram (uint8 cmd, int16 adp, int16 ado, int16 len, int32 val)
  Construct a datagram with up to 4 bytes of data.
- virtual ~EcatDgram ()
  Default destructor for an EtherCAT datagram.
- void Init (uint8 cmd, int16 adp, int16 ado, int16 len, int32 val)
  Initialize a datagram with up to 4 bytes of data.
- void Init (uint8 cmd, int16 adp, int16 ado, int16 len, void *ptr=0)
  Initialize a datagram with an arbitrary amount of data.
- void Reset (void)
  Reset the datagram.
- void setNext (EcatDgram *n)
  Add a new datagram after this one.
- EcatDgram * getNext (void)
  Get a pointer to the next datagram stored in the frame.
- virtual const Error * Load (void *buff, int16 &off)
  Pack this datagram into a memory buffer that's large enough to hold an entire EtherCAT frame.

- bool checkNdx (void)

  Compare the index stored in the frame that this datagram is part of to the expected value.
- void setData (void *ptr)

  Update the data value stored in this datagram.
- void setData (int32 val)

  Update the data value stored in this datagram.
- void setNdx (uint8 ndx)

  Update the index value associated with this datagram.
- uint8 getNdx (void)

  Return the current datagram index value.
- int16 getDgramLen (void)

  Return the datagram length in bytes.

5.42.1 Detailed Description

Generic EtherCAT datagram class.

At the lowest levels, an EtherCAT packet is made up of one or more reads and/or writes to memory locations on the slave nodes. Each of these memory accesses is called a datagram.

There are several different types of datagrams; reads/writes to a node based on it's location in the network, reads/writes to a node based on it's address, broadcast accesses to all nodes on the network, etc.

This class makes up the base for all datagrams.

In general, CML uses don't need to concern themselves with EtherCAT datagrams. This class is used internally in the libary to support low level communications over the EtherCAT network.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 EcatDgram() [1/2]

```
EcatDgram (
            uint8 cmd,
            int16 adp,
            int16 ado,
            int16 len,
            void * ptr = 0 )
```

Construct a datagram with a variable data length.

Parameters

cmd	Identifies the type of datagram
adp	Generally the EtherCAT node on the network
ado	Generally identifies the memory address being accessed.
len	Length of the memory access in bytes
ptr	Pointer where the read/write data is held

5.42.2.2 EcatDgram() [2/2]

```
EcatDgram (
            uint8 cmd,
            int16 adp,
            int16 ado,
            int16 len,
            int32 val )
```

Construct a datagram with up to 4 bytes of data.

Parameters

cmd	Identifies the type of datagram
adp	Generally the EtherCAT node on the network
ado	Generally identifies the memory address being accessed.
len	Length of the memory access in bytes (up to 4)
val	Value of the data to send.

5.42.3 Member Function Documentation

5.42.3.1 checkNdx()

```
bool checkNdx (
            void  )
```

Compare the index stored in the frame that this datagram is part of to the expected value.

This can be used to help verify that the frame received over the EtherCAT network contains the expected datagrams.

Returns

true if the index stored in the frame holds the expected value.

5.42.3.2 getDgramLen()

```
int16 getDgramLen (
            void  )
```

Return the datagram length in bytes.

That's the length of the data plus the 12 byte header.

Returns

The datagram length in bytes.

5.42.3.3 getNdx()

```
uint8 getNdx (
            void  )
```

Return the current datagram index value.

Returns

The current index value.

5.42.3.4 getNext()

```
EcatDgram * getNext (
            void  )
```

Get a pointer to the next datagram stored in the frame.

Returns

A pointer to the next datagram, or NULL if there is none.

5.42.3.5 Init() [1/2]

```
void Init (
            uint8 cmd,
            int16 adp,
            int16 ado,
            int16 len,
            int32 val )
```

Initialize a datagram with up to 4 bytes of data.

The data value will be stored in a local buffer

Parameters

cmd	Identifies the type of datagram
adp	Generally the EtherCAT node on the network
ado	Generally identifies the memory address being accessed.
len	Length of the memory access in bytes (up to 4)
val	Value of the data to send.

5.42.3.6 Init() [2/2]

```
void Init (
            uint8 cmd,
            int16 adp,
            int16 ado,
            int16 len,
            void * ptr = 0 )
```

Initialize a datagram with an arbitrary amount of data.

Parameters

cmd	Identifies the type of datagram
adp	Generally the EtherCAT node on the network
ado	Generally identifies the memory address being accessed.
len	Length of the memory access in bytes
ptr	Pointer where the read/write data is held

5.42.3.7 Load()

```
const Error * Load (
            void * ptr,
            int16 & off )  [virtual]
```

Pack this datagram into a memory buffer that's large enough to hold an entire EtherCAT frame.

Parameters

ptr	Points to the frame buffer
off	Gives the offset in the frame where this datagram should be loaded. On return, this is increased by the size of the datagram

Returns

An error code or null on success.

5.42.3.8 setData() [1/2]

```
void setData (
            void * ptr )
```

Update the data value stored in this datagram.

Parameters

ptr	Pointer to the data to store. The data referenced by this pointer should be at least as long as the length of the datagram.

5.42.3.9 setData() [2/2]

```
void setData (
            int32 val )
```

Update the data value stored in this datagram.

Parameters

val	The value to be stored in the datagram. The datagram is expected to be no longer then 4 bytes.

5.42.3.10 setNdx()

```
void setNdx (
            uint8 ndx )
```

Update the index value associated with this datagram.

Parameters

ndx	The new index value to store.

5.42.3.11 setNext()

```
void setNext (
            EcatDgram * n )
```

Add a new datagram after this one.

Parameters

n	Pointer to the next datagram. A local copy of this pointer will be stored

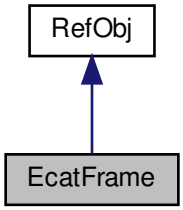
The documentation for this class was generated from the following files:

- CML_EtherCAT.h
- EtherCAT.cpp

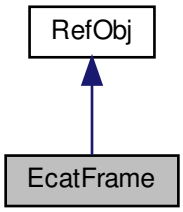
5.43 EcatFrame Class Reference

EtherCAT frame class.

Inheritance diagram for EcatFrame:



Collaboration diagram for EcatFrame:



Additional Inherited Members

5.43.1 Detailed Description

EtherCAT frame class.

This is used internally to represent a single frame of data sent over the EtherCAT network. The frame consists of a header structure followed by a series of datagrams.

The documentation for this class was generated from the following files:

- CML_EtherCAT.h
- EtherCAT.cpp

5.44 EncoderErrorConfig Struct Reference

This structure holds configuration info about the encoder error filter.

Public Attributes

- uint32 encoderErrorConfig

  Encoder Error Filter Configuration This parameter it set up as follows: Bits 0-3: Maximum number of consecutice bad samples to allow.

5.44.1 Detailed Description

This structure holds configuration info about the encoder error filter.

5.44.2 Member Data Documentation

5.44.2.1 encoderErrorConfig

```
uint32 encoderErrorConfig
```

Encoder Error Filter Configuration This parameter it set up as follows: Bits 0-3: Maximum number of consecutice bad samples to allow.

Bits 4-15: Reserved for future use. Bits 16-28: Maximum difference between the actual and extrapolated reading before treating reading as bad. Bits 29-31: Reserved for future use.

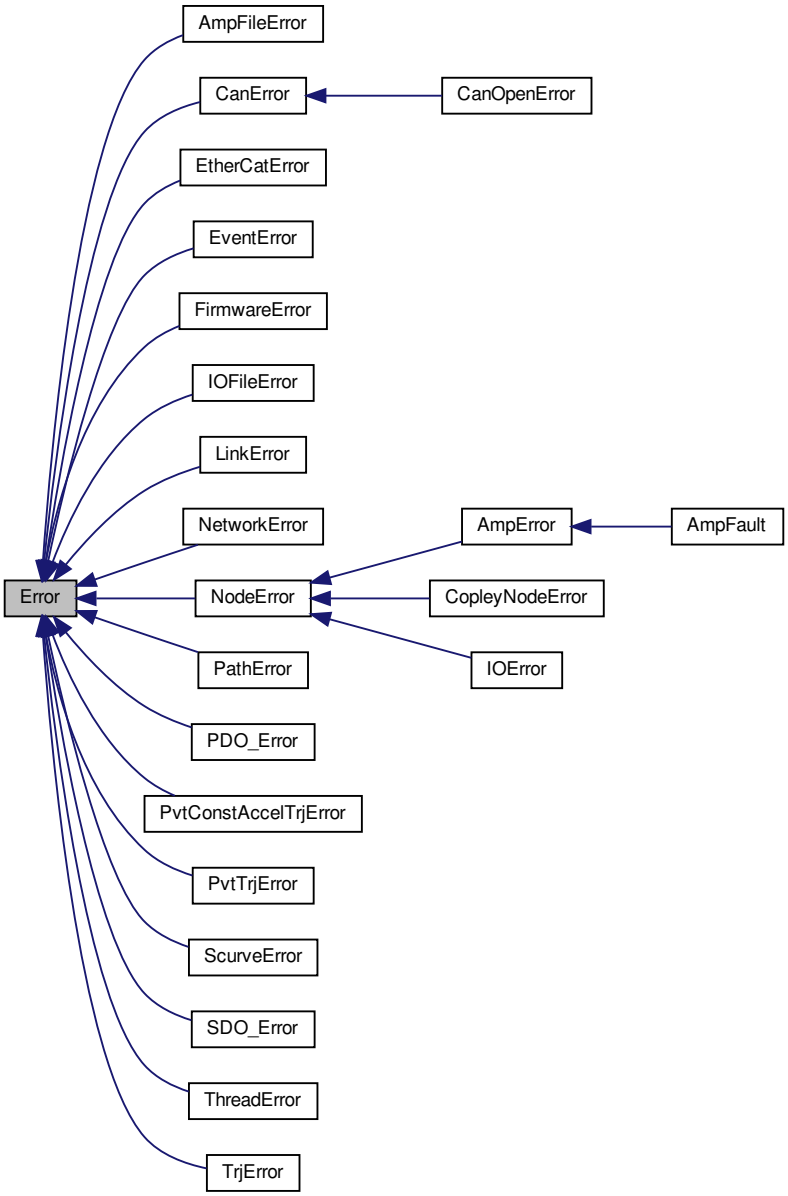
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

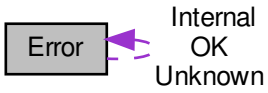
5.45 Error Class Reference

This class is the root class for all error codes returned by functions defined within the Motion Library.

Inheritance diagram for Error:



Collaboration diagram for Error:



Public Member Functions

- const char * toString () const

  Return a C style string describing the error condition represented by this error object.
- uint16 GetID (void) const

  Return an integer ID that can be used to identify the error.

Static Public Member Functions

- static const Error * Lookup (int16 id)

  Lookup the constant error object associated with the passed ID code.

Static Public Attributes

- static const Error OK

  A constant error object that represents no error.
- static const Error Unknown

  An invalid error ID code was passed to Error::Lookup.
- static const Error Internal

  Generic internal software error.

Protected Member Functions

- Error (uint16 i, const char *desc)

  Constructor used to create an error object of a particular type.

5.45.1 Detailed Description

This class is the root class for all error codes returned by functions defined within the Motion Library.

Every error condition defined in the library has a constant, static error object associated with it. Pointers to these objects are returned from the various function calls.

All library functions that return an error object pointer will return a NULL pointer in the case of no error. This allows one to simply test the error pointer returned to determine if it indicates an error.

For example:

```
const Error *err = SomeFunctionCall();


if( err )
   printf( "Error: %%s\\n", err->toString() );
else
   printf( "no error\\n" );
```

To test for a specific error condition, the following code can be used:

```
const Error *err = SomeFunctionCall();


if( err == &ThreadError::Timeout )
   printf( "A timeout occurred\\n" );
```

Note that the constructor used to create a new unique error code is protected, therefore only sub-classes of the Error object are allowed to create new unique error codes.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 Error()

```
Error (
            uint16 i,
            const char * desc )  [inline], [protected]
```

Constructor used to create an error object of a particular type.

This constructor is protected, so only sub-classes of the Error class can construct objects in this manner.

A unique ID coded and a description string must be provided with the new object.

Parameters

i	The unique error ID value
desc	A description of the error

5.45.3 Member Function Documentation

5.45.3.1 GetID()

```
uint16 GetID (
            void  ) const  [inline]
```

Return an integer ID that can be used to identify the error.

Each error code in the system has a unique 16-bit integer identifier associated with it. This function can be used to return this identifier.

These ID codes are primarily intended for internal use by the Error object, they are only provided externally for use in system which require an integer error ID code to interface with other parts of the system. The function Error::Lookup can be used to convert the ID value back into an error object.

Returns

The 16-bit integer ID value associated with this Error object.

5.45.3.2 Lookup()

```
const Error * Lookup (
            int16 id )  [static]
```

Lookup the constant error object associated with the passed ID code.

If the passed ID doesn't correspond to any known Error object, then the address of the Error::Unknown will be returned.

Parameters

id	The ID code of the error to be found

Returns

A pointer to an error object.

5.45.3.3 toString()

```
const char* toString ( ) const  [inline]
```

Return a C style string describing the error condition represented by this error object.

Returns

A constant character string describing the error condition.

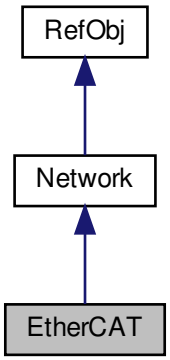
The documentation for this class was generated from the following files:

- CML_Error.h
- Error.cpp

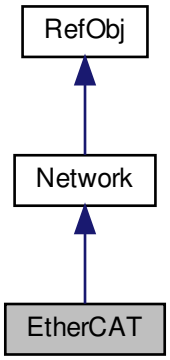
5.46 EtherCAT Class Reference

The EtherCAT class is the top level interface into the EtherCAT network.

Inheritance diagram for EtherCAT:



Collaboration diagram for EtherCAT:



Public Member Functions

- NetworkType GetNetworkType (void)
  Return the network type.
- const Error * GetIdFromEEPROM (Node *n, struct NodeIdentity &id)
  Read the node ID info that was pullsed from EEPROM on startup.
- const Error * SetNodeGuard (Node *n, GuardProtocol type, Timeout timeout=200, uint8 life=0)
  Configure the heartbeat protocol for an EtherCAT node.
- const Error * SetSync0Period (Node *n, uint32 ns)
  Set the period of the SYNC0 signal used on nodes with a distributed clock.
- uint16 getNodeCount (void)
  Return the number of nodes discovered on the network.
- const Error * GetNodeAddress (Node *n, uint16 &addr)
  Return the EtherCAT address assigned to this node.
- const Error * FoE_DnldStart (Node *n, const char *filename, uint32 password, Timeout to=2000)
  Initiate a new file download using the File over EtherCAT (FoE) protocol.
- const Error * FoE_DnldData (Node *n, int32 len, uint8 *buff, Timeout to=2000, bool end=true)
  Download a block of file data to a node using the File over EtherCAT (FoE) protocol.
- const Error * FoE_UpldStart (Node *n, const char *filename, uint32 password, Timeout to=2000)
  Initiate a new file upload using the File over EtherCAT (FoE) protocol.
- const Error * FoE_UpldData (Node *n, int32 max, int32 *len, uint8 *data, Timeout to=2000)
  Upload a block of file data from a node using the File over EtherCAT (FoE) protocol.
- int32 FoE_LastErrInfo (Node *n, char *msg, int maxMsg)
  Return the detailed error code and message for the most recent FoE error response on this node.
- int32 maxSdoToNode (Node *n)
  Return the maximum number of bytes that can be sent in an SDO message.
- int32 maxSdoFromNode (Node *n)
  Return the maximum number of bytes that can be received in an SDO message.
- const Error * WaitCycleUpdate (Timeout to)
  Wait for the cyclic thread to update.

Protected Member Functions

- const Error * InitDistClk (void)

  Initialize the distributed clock system on this network.
- const Error * MailboxTransfer (Node *n, uint16 len, uint16 *ret, Timeout timeout=2000)

  Write a block of data to the mailbox of an EtherCAT node and wait for a response.
- const Error * MailboxTransfer (Node *n, void *send, uint16 len, void *resp, uint16 *ret, uint16 max, Timeout timeout=2000, bool getMtx=false)

  Write a block of data to the mailbox of an EtherCAT node and wait for a response.
- const Error * AddToFrame (class EcatFrame *frame, class EcatDgram *dg)

  Add a datagram to the frame.

Friends

- class **Linkage**
- class **Node**

Additional Inherited Members

5.46.1 Detailed Description

The EtherCAT class is the top level interface into the EtherCAT network.

There should be at least one object of this class in every EtherCAT based application.

5.46.2 Member Function Documentation

5.46.2.1 AddToFrame()

```
const Error * AddToFrame (
            class EcatFrame * frame,
            class EcatDgram * dg )  [protected]
```

Add a datagram to the frame.

If the frame is too large to add the datagram, then first send the frame, reset it and add the datagram.

Parameters

frame	The frame
dg	The datagram

Returns

An error pointer or NULL on success

5.46.2.2 FoE_DnldData()

```
const Error * FoE_DnldData (
            Node * n,
            int32 len,
            uint8 * data,
            Timeout to = 2000,
            bool end = true )
```

Download a block of file data to a node using the File over EtherCAT (FoE) protocol.

The transfer should have been previously initialized by calling the EtherCAT::FoE_DnldStart function.

Parameters

n	Pointer to the node to download to
len	Length of data to download in bytes
data	Buffer holding the data to download
to	A timeout value.
end	True if this is the last block of data to be sent

Returns

An error pointer on failure, or null on success.

5.46.2.3 FoE_DnldStart()

```
const Error * FoE_DnldStart (
            Node * n,
            const char * filename,
            uint32 password,
            Timeout to = 2000 )
```

Initiate a new file download using the File over EtherCAT (FoE) protocol.

This function should be used to start a new file download. On success, the EtherCAT::FoE_DnldData function can be used to pass the file data.

Parameters

n	Points to the node who will receive this download
filename	Name of the file to be sent. May be null if no file name is to be passed.
password	Optional password value to be sent to the slave device (0 if not used)
to	Timeout to wait for response.

Returns

An error pointer on failure, or null on success.

5.46.2.4 FoE_LastErrInfo()

```
int32 FoE_LastErrInfo (
            Node * n,
            char * msg,
            int maxMsg )
```

Return the detailed error code and message for the most recent FoE error response on this node.

Parameters

n	Points to the node in question
msg	Buffer where error message should be stored (can be null)
maxMsg	Size of the message buffer

Returns

The most recent error code returned by this node.

5.46.2.5 FoE_UpldData()

```
const Error * FoE_UpldData (
            Node * n,
            int32 max,
            int32 * len,
            uint8 * data,
            Timeout to = 2000 )
```

Upload a block of file data from a node using the File over EtherCAT (FoE) protocol.

The transfer should have been previously initialized by calling the EtherCAT::FoE_UpldStart function.

Parameters

n	Pointer to the node to upload from
max	Maximum length of data to upload in bytes
len	On return, holds the actual number of bytes uploaded.
data	Buffer where data will be written
to	A timeout value.

Returns

An error pointer on failure, or null on success.

5.46.2.6 FoE_UpldStart()

```
const Error * FoE_UpldStart (
            Node * n,
            const char * filename,
            uint32 password,
            Timeout to = 2000 )
```

Initiate a new file upload using the File over EtherCAT (FoE) protocol.

This function should be used to start a new file upload. On success, the EtherCAT::FoE_UpldData function can be used to read the file data.

Parameters

n	Points to the node from which the data will be uploaded.
filename	Name of the file to be read. May be null if no file name is to be passed.
password	Optional password value to be sent to the slave device (0 if not used)
to	Timeout to wait for response.

Returns

An error pointer on failure, or null on success.

5.46.2.7 GetIdFromEEPROM()

```
const Error * GetIdFromEEPROM (
            Node * n,
            struct NodeIdentity & id )
```

Read the node ID info that was pullsed from EEPROM on startup.

This can be useful on nodes that don't support the CoE protocol and therefor can't use the more standard Node::Get↩ Identity method.

Parameters

n	Points to the node to access
id	Structure where identiy info will be returned

Returns

An error pointer on failure, or null on success.

5.46.2.8 GetNetworkType()

```
NetworkType GetNetworkType (
            void  )  [inline], [virtual]
```

Return the network type.

Returns

Always returns the value NET_TYPE_ETHERCAT

Implements Network.

5.46.2.9 GetNodeAddress()

```
const Error * GetNodeAddress (
            Node * n,
            uint16 & addr )
```

Return the EtherCAT address assigned to this node.

When each node is added to an EtherCAT network, the network object assigns it a unique address. This address is then used by the master to communicate with the node over the network.

Parameters

n	Pointer to the node object
addr	The assigned address will be returned here

Returns

An error pointer on failure, or null on success.

5.46.2.10 getNodeCount()

```
uint16 getNodeCount (
            void  )  [inline]
```

Return the number of nodes discovered on the network.

Returns

The number of nodes on the EtherCAT network

5.46.2.11 InitDistClk()

```
const Error * InitDistClk (
            void  )  [protected]
```

Initialize the distributed clock system on this network.

This is called at startup by EtherCAT::Open();

Returns

An error object, or NULL on success

5.46.2.12 MailboxTransfer() [1/2]

```
const Error * MailboxTransfer (
            Node * n,
            uint16 len,
            uint16 * ret,
            Timeout timeout = 2000 )  [protected]
```

Write a block of data to the mailbox of an EtherCAT node and wait for a response.

Note that the mailbox mutex should be held when this is called.

The input and output data used by this function is stored in the previously allocated sync buffers

Parameters

n	The node to access.
len	The number of bytes of data to send
ret	returns the actual number of bytes returned.
timeout	The maximum time to wait for a response.

Returns

An error pointer on failure, or null on success.

5.46.2.13 MailboxTransfer() [2/2]

```
const Error * MailboxTransfer (
            Node * n,
            void * send,
            uint16 len,
            void * resp,
            uint16 * ret,
            uint16 max,
            Timeout timeout = 2000,
            bool getMtx = false )  [protected]
```

Write a block of data to the mailbox of an EtherCAT node and wait for a response.

Note that the mailbox mutex should be held when this is called.

Parameters

n	The node to access.
send	Buffer holding the data to send to the node
len	The number of bytes of data to send
recv	Buffer where the received data will be returned
ret	returns the actual number of bytes returned.
max	Maximum number of bytes to return
timeout	The maximum time to wait for a response.
getMtx	True if the mailbox mutex should be acquired by this function. False if the mutex is already held by the calling function.

Returns

An error pointer on failure, or null on success.

5.46.2.14 maxSdoFromNode()

```
int32 maxSdoFromNode (
            Node * n )  [virtual]
```

Return the maximum number of bytes that can be received in an SDO message.

For CANopen this is always 8 (the max size of a CAN frame). For EtherCAT it's the size of the mailbox buffer, and is node specific

Parameters

n	The node to query

Returns

The maximum number of bytes in an SDO receive message

Reimplemented from Network.

5.46.2.15 maxSdoToNode()

```
int32 maxSdoToNode (
            Node * n )  [virtual]
```

Return the maximum number of bytes that can be sent in an SDO message.

For CANopen this is always 8 (the max size of a CAN frame). For EtherCAT it's the size of the mailbox buffer, and is node specific

Parameters

n	The node to query

Returns

The maximum number of bytes in an SDO transmit message

Reimplemented from Network.

5.46.2.16 SetNodeGuard()

```
const Error * SetNodeGuard (
            Node * n,
            GuardProtocol type,
            Timeout timeout = 200,
            uint8 life = 0 )  [virtual]
```

Configure the heartbeat protocol for an EtherCAT node.

This sets the heartbeat timeout used for process data on the EtherCAT node.

Parameters

n	The node to configure heartbeat on
type	The type of node guarding to configure.
timeout	A timeout (milliseconds) to use for this node guarding protocol. If not specified, this parameter defaults to 200 milliseconds.
life	This parameter is not used under EtherCAT.

Returns

An error object, or NULL on success

Implements Network.

5.46.2.17 SetSync0Period()

```
const Error * SetSync0Period (
            Node * n,
            uint32 ns )
```

Set the period of the SYNC0 signal used on nodes with a distributed clock.

This also starts generation of the SYNC0 signal.

Parameters

n	The node to modify
ns	The period in nanoseconds.

Returns

An error object, or NULL on success

5.46.2.18 WaitCycleUpdate()

```
const Error * WaitCycleUpdate (
            Timeout to )
```

Wait for the cyclic thread to update.

Parameters

to	Max time to wait before returning an error

Returns

An error object on failure, or NULL on success.

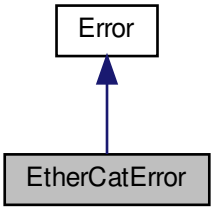
The documentation for this class was generated from the following files:

- CML_EtherCAT.h
- ecatdc.cpp
- EtherCAT.cpp

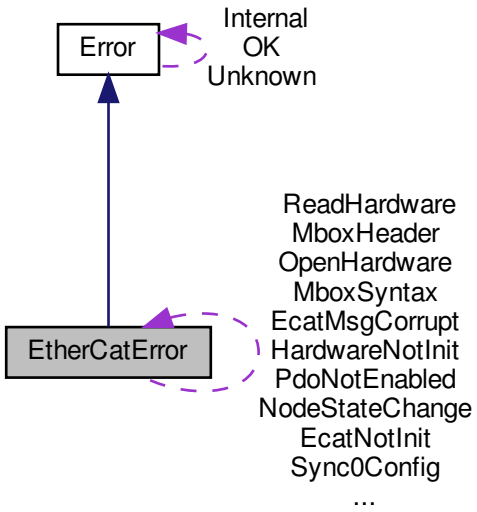
5.47 EtherCatError Class Reference

This class holds the error codes that describe EtherCAT error conditions.

Inheritance diagram for EtherCatError:



Collaboration diagram for EtherCatError:



Static Public Attributes

- static const EtherCatError ThreadStart

  Unable to start main EtherCAT.
- static const EtherCatError HardwareNotInit

  EtherCAT hardware has not been initialized.
- static const EtherCatError OpenHardware

  Unable to open EtherCAT hardware.
- static const EtherCatError ReadHardware

  Error reading from Ethernet socket.
- static const EtherCatError WriteHardware

  Error writing to Ethernet socket.
- static const EtherCatError NoResponse

  Remote device did not respond to request (working counter is zero).
- static const EtherCatError EcatNotInit

  EtherCAT network object not initialized.
- static const EtherCatError NodeNotFound

  Specified node was not found on the EtherCAT network.
- static const EtherCatError NodeBootMode

  EtherCAT node is currently in boot mode.
- static const EtherCatError NodeStateChange

  Error changing node operational state.
- static const EtherCatError EcatMsgCorrupt

EtherCAT message received from network is corrupt.

- static const EtherCatError DatagramWontFit

  Not enough space in frame for new datagram.

- static const EtherCatError MboxError

  EtherCAT node returned an unknown mailbox error code.

- static const EtherCatError MboxSyntax

  EtherCAT node reported the syntax of mailbox header is invalid.

- static const EtherCatError MboxProtocol

  EtherCAT node does not support the requested mailbox protocol.

- static const EtherCatError MboxChannel

  EtherCAT node returned an invalide mailbox channel code.

- static const EtherCatError MboxService

  EtherCAT node does not support the requested mailbox service.

- static const EtherCatError MboxHeader

  EtherCAT node reported an invalid mailbox protocol header.

- static const EtherCatError MboxTooShort

  EtherCAT node reported the length of the mailbox data is too short.

- static const EtherCatError MboxMemory

  EtherCAT node reported insufficient memory for mailbox transfer.

- static const EtherCatError MboxSize

  EtherCAT node reported inconsistent mailbox data length.

- static const EtherCatError FoEformat

  EtherCAT node returned incorrectly formatted FoE response.

- static const EtherCatError FoEerror

  EtherCAT node returned an FoE error response.

- static const EtherCatError NodeNotInit

  EtherCAT node has not been initialized.

- static const EtherCatError PdoNotEnabled

  PDO is not currently enabled on network.

- static const EtherCatError NetworkWiringError

  EtherCAT network is not correctly wired.

- static const EtherCatError Sync0Config

  Error configuring SYNC0 timer on slave device.

Additional Inherited Members

5.47.1 Detailed Description

This class holds the error codes that describe EtherCAT error conditions.

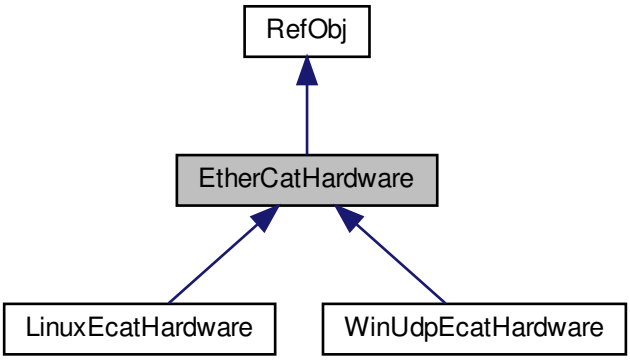
The documentation for this class was generated from the following file:

- CML_EtherCAT.h

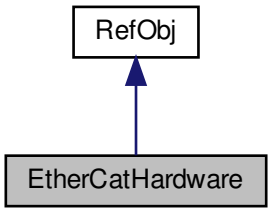
5.48 EtherCatHardware Class Reference

Low level Ethernet hardware interface.

Inheritance diagram for EtherCatHardware:



Collaboration diagram for EtherCatHardware:



Additional Inherited Members

5.48.1 Detailed Description

Low level Ethernet hardware interface.

This pure virtual class is the base for an OS specific class which implements the interface to the Ethernet hardware.

The documentation for this class was generated from the following file:

- CML_EtherCAT.h

5.49 EtherCatSettings Class Reference

Configuration object used to customize global settings for the EtherCAT network.

Public Member Functions

- EtherCatSettings ()

  Default constructor for EtherCatSettings object.

Public Attributes

- int readThreadPriority

  Defines the EtherCAT read thread priority.
- int cycleThreadPriority

  Defines the EtherCAT cycle thread priority.
- Timeout cyclePeriod

  EtherCAT cycle period.

5.49.1 Detailed Description

Configuration object used to customize global settings for the EtherCAT network.

An object of this type may be passed to the EtherCAT::Open() method when the network is first opened.

If no settings object is passed to the EtherCAT::Open() method, then the behavior is exactly the same as passing a EtherCatSettings object with the default settings.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 EtherCatSettings()

```
EtherCatSettings ( )  [inline]
```

Default constructor for EtherCatSettings object.

This sets all settings to their default values

5.49.3 Member Data Documentation

5.49.3.1 cyclePeriod

```
Timeout cyclePeriod
```

EtherCAT cycle period.

This parameter defines the update rate at which the EtherCAT network is polled. Default: 1 ms.

5.49.3.2 cycleThreadPriority

```
int cycleThreadPriority
```

Defines the EtherCAT cycle thread priority.

The cycle thread is started when the EtherCAT object is first opened (using EtherCAT::Open()). This thread is responsible for polling the EtherCAT network at a set frequency. It should be run at a relatively high priority. Default: 9

5.49.3.3 readThreadPriority

```
int readThreadPriority
```

Defines the EtherCAT read thread priority.

The read thread is started when the EtherCAT object is first opened (using EtherCAT::Open()). This thread is responsible for reading Ethernet messages from the hardware and processing them as they arrive. It should be run at a relatively high priority. Default: 9

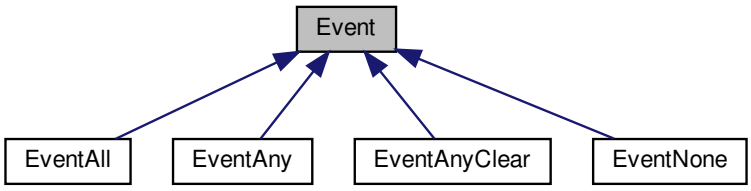
The documentation for this class was generated from the following file:

- CML_EtherCAT.h

5.50 Event Class Reference

Events are a generic mechanism used to wait on some condition.

Inheritance diagram for Event:



Public Member Functions

- Event (uint32 val=0)

  Default constructor for an Event object.
- virtual ~Event ()

  Event distructor.
- Event (const Event &)

  Event object copy constructor.
- Event & operator= (const Event &)

  Event assignment operator.
- const Error * setValue (uint32 val)

  Change the value that the event will wait for.
- void setChain (class EventMap &map, uint32 mask)

  Setup event chaining.
- void delChain (void)

  Remove any pointer to a chained event map.
- uint32 getValue (void)

  Return the value that this event will wait on.
- uint32 getMask (void)

  Return the most recent mask value that caused the event to succeed.
- const Error * Wait (EventMap &m, Timeout timeout)

  Wait on an event.
- virtual bool isTrue (uint32 mask)

  Test the event to see if it's condition is true.

Protected Attributes

- uint32 value

  This is the value that the event is waiting for.

Friends

- class **EventMap**

5.50.1 Detailed Description

Events are a generic mechanism used to wait on some condition.

They are used in conjunction with the EventMap object.

Every EventMap object has a 32-bit mask register which describes it's state. Event objects may be attached to an EventMap to wait for any combination of bits in the mask to become active.

The base Event class is a virtual class, and therefore shouldn't be used directly. It is extended by a number of sub-classes which are used to wait for certain map bits to be set, cleared, etc.

An Event object may only be assigned to one EventMap at a time. An attempt to attach it to multiple EventMap objects will result in an error. Further, Event objects are not thread safe. Only one thread should access a particular Event object at a time. EventMap objects however are thread safe, so any number of threads may attach their own Event objects to the same EventMap object without issue.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 Event() [1/2]

```
Event (
            uint32 val = 0 )
```

Default constructor for an Event object.

Parameters

val	The value that the event will wait for. If not specified, defaults to zero.

5.50.2.2 ~Event()

```
~Event (
            void  )  [virtual]
```

Event distructor.

This makes sure the event isn't mapped when it's destroyed

5.50.2.3 Event() [2/2]

```
Event (
            const Event & e )
```

Event object copy constructor.

Parameters

e	Another event that this will copy.

5.50.3 Member Function Documentation

5.50.3.1 delChain()

```
void delChain (
            void  )
```

Remove any pointer to a chained event map.

This undoes any chaining that was setup using the Event::setChain method.

5.50.3.2 getMask()

```
uint32 getMask (
            void  )  [inline]
```

Return the most recent mask value that caused the event to succeed.

After a successful Wait, this can be used to return the mask that caused the successful match.

Returns

The mask value

5.50.3.3 getValue()

```
uint32 getValue (
            void  )  [inline]
```

Return the value that this event will wait on.

Returns

The event value

5.50.3.4 isTrue()

```
virtual bool isTrue (
            uint32 mask )  [inline], [virtual]
```

Test the event to see if it's condition is true.

This method should be implemented in the sub-class to define the type of event matching used. The base class always returns false.

Parameters

mask	The EventMap mask compare to.

Returns

true if the event is satisfied, false if not.

Reimplemented in EventNone, EventAll, EventAnyClear, and EventAny.

5.50.3.5 operator=()

```
Event & operator= (
            const Event & e )
```

Event assignment operator.

This uses the value of the passed event object to update this event's value. Note that it's not legal to change an events value while the event is attached to an EventMap object. If this is attempted, then this assignment will silently fail. The preferred method for changing an event's value is through the Event::setValue method.

Parameters

e	Another event that will be used to initialize this one.

Returns

A reference to this object

5.50.3.6 setChain()

```
void setChain (
            class EventMap & map,
            uint32 mask )
```

Setup event chaining.

Each time the event is updated, it will either set or clear the bits specified by mask in the referenced EventMap object. The bits will be set if the update causes the event to be true, and cleared otherwise.

WARNING: a local reference to the passed map object is held by this event after this method is called. The event map may not be deleted until the Event::delChain method has been used to remove this reference. This is not handled automatically by the EventMap destructor.

Parameters

map	The map to chain.
mask	The bit(s) to set/clear in the chained map based on the state of this event.

5.50.3.7 setValue()

```
const Error * setValue (
              uint32 val )
```

Change the value that the event will wait for.

The event's value can not be changed while it is attached to an EventMap object. If this is attempted, then &Event↩ Error::AlreadyOwned will be returned.

Parameters

val	The new event value

Returns

A pointer to an error object, or NULL on success.

5.50.3.8 Wait()

```
const Error * Wait (
              EventMap & eventMapObj,
              Timeout timeout )
```

Wait on an event.

This function causes the calling thread to pend until the event is true, or the timeout expires.

Note that the event should not be owned by any event map when this is called.

Parameters

eventMapObj	The event map that this event should watch.
timeout	The maximum amount of time to wait (milliseconds). If < 0, then the task will wait forever.

Returns

A pointer to an error object on failure, or NULL on success.

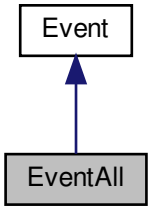
The documentation for this class was generated from the following files:

- CML_EventMap.h
- EventMap.cpp

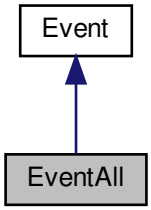
5.51 EventAll Class Reference

This is an event that matches if all of a group of bits are set in the EventMap mask.

Inheritance diagram for EventAll:



Collaboration diagram for EventAll:



Public Member Functions

- EventAll (uint32 v=0)

  Construct a new object specifying a group of bits that should be checked.
- EventAll (const EventAll &e)

  Construct a new event object using the value of a passed event.
- virtual bool isTrue (uint32 mask)

  Check the event against the passed mask.

Additional Inherited Members

5.51.1 Detailed Description

This is an event that matches if all of a group of bits are set in the EventMap mask.

5.51.2 Constructor & Destructor Documentation

5.51.2.1 EventAll() [1/2]

```
EventAll (
            uint32 v = 0 )  [inline]
```

Construct a new object specifying a group of bits that should be checked.

Parameters

v	The bit mask that the event will wait on. Default is zero

5.51.2.2 EventAll() [2/2]

```
EventAll (
            const EventAll & e )  [inline]
```

Construct a new event object using the value of a passed event.

Parameters

e	Another event object who's value will be used to initialize this one.

5.51.3 Member Function Documentation

5.51.3.1 isTrue()

```
virtual bool isTrue (
            uint32 mask )  [inline], [virtual]
```

Check the event against the passed mask.

If all of the selected bits are set in the mask, then the event is satisfied.

Parameters

mask	The mask to test against

Returns

true if all bits of interest are set in the mask.

Reimplemented from Event.

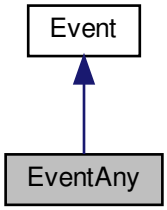
The documentation for this class was generated from the following file:

- CML_EventMap.h

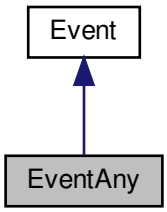
5.52 EventAny Class Reference

This is an event that matches if any of a group of bits are set in the EventMap mask.

Inheritance diagram for EventAny:



Collaboration diagram for EventAny:



Public Member Functions

- EventAny (uint32 v=0)

  Construct a new object specifying a group of bits that should be checked.
- EventAny (const EventAny &e)

  Construct a new event object using the value of a passed event.
- virtual bool isTrue (uint32 mask)

  Check the event.

Additional Inherited Members

5.52.1 Detailed Description

This is an event that matches if any of a group of bits are set in the EventMap mask.

5.52.2 Constructor & Destructor Documentation

5.52.2.1 EventAny() [1/2]

```
EventAny (
            uint32 v = 0 )  [inline]
```

Construct a new object specifying a group of bits that should be checked.

Parameters

v	The bit mask that the event will wait on. Default is zero

5.52.2.2 EventAny() [2/2]

```
EventAny (
            const EventAny & e )  [inline]
```

Construct a new event object using the value of a passed event.

Parameters

e	Another event object who's value will be used to initialize this one.

5.52.3 Member Function Documentation

5.52.3.1 isTrue()

```
virtual bool isTrue (
            uint32 mask )  [inline], [virtual]
```

Check the event.

The event is satisfied if any of the selected bits are set in the mask.

Parameters

mask	Bitmap identifying the event bits that are of interest.

Returns

true if any bits of interest are set in the mask.

Reimplemented from Event.

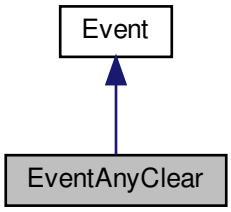
The documentation for this class was generated from the following file:

- CML_EventMap.h

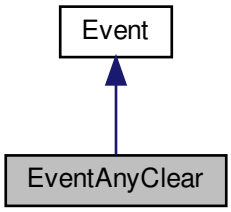
5.53 EventAnyClear Class Reference

This is an event that matches if any of a group of bits are clear in the EventMap mask.

Inheritance diagram for EventAnyClear:



Collaboration diagram for EventAnyClear:



Public Member Functions

- EventAnyClear (uint32 v=0)

  Construct a new object specifying a group of bits that should be checked.
- EventAnyClear (const EventAnyClear &e)

  Construct a new event object using the value of a passed event.
- virtual bool isTrue (uint32 mask)

  Check the event.

Additional Inherited Members

5.53.1 Detailed Description

This is an event that matches if any of a group of bits are clear in the EventMap mask.

5.53.2 Constructor & Destructor Documentation

5.53.2.1 EventAnyClear() [1/2]

```
EventAnyClear (
            uint32 v = 0 )  [inline]
```

Construct a new object specifying a group of bits that should be checked.

Parameters

v	The bit mask that the event will wait on. Default is zero

5.53.2.2 EventAnyClear() [2/2]

```
EventAnyClear (
            const EventAnyClear & e )  [inline]
```

Construct a new event object using the value of a passed event.

Parameters

e	Another event object who's value will be used to initialize this one.

5.53.3 Member Function Documentation

5.53.3.1 isTrue()

```
virtual bool isTrue (
            uint32 mask )  [inline], [virtual]
```

Check the event.

The event is satisfied if any of the selected bits are clear in the mask.

Parameters

mask	A bitmap identifying the event bits of interest.

Returns

true if any bits of interest are set in the mask.

Reimplemented from Event.

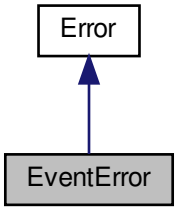
The documentation for this class was generated from the following file:

- CML_EventMap.h

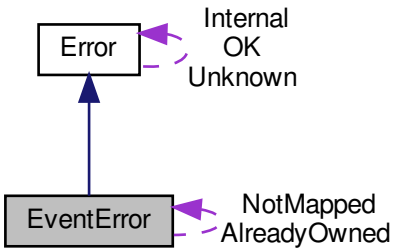
5.54 EventError Class Reference

This class represents error conditions related to the Event object.

Inheritance diagram for EventError:



Collaboration diagram for EventError:



Static Public Attributes

- static const EventError AlreadyOwned

  The event is already mapped to another object.
- static const EventError NotMapped

  The event is not mapped to this object.

Protected Member Functions

- EventError (uint16 id, const char ∗desc)

  Standard protected constructor.

Additional Inherited Members

5.54.1 Detailed Description

This class represents error conditions related to the Event object.

The documentation for this class was generated from the following file:

- CML_EventMap.h

5.55 EventMap Class Reference

An event map is a mechanism that allows one or more threads to wait on some pre-defined event, or group of events.

Public Member Functions

- virtual ~EventMap (void)

  Event map destructor.
- const Error * Add (Event *e)

  Add the passed event to the list of events pending on this map.
- const Error * Remove (Event *e)

  Remove the passed event from the list of events pending on this map.
- uint32 getMask (void)

  Get the current value of the mask for this event map.
- void setMask (uint32 mask)

  Update the event mask.
- void setBits (uint32 bits)

  Set bits in the event mask.
- void clrBits (uint32 bits)

  Clear bits in the event mask.
- void changeBits (uint32 bits, uint32 value)

  Change the value of specified bits in the mask for this event.

Friends

- class **Event**

5.55.1 Detailed Description

An event map is a mechanism that allows one or more threads to wait on some pre-defined event, or group of events.

For a particular event map, there are one or more events that are grouped with that map. Any number of threads may pend on the state of these events.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 ~EventMap()

```
~EventMap (
            void  )  [virtual]
```

Event map destructor.

Removes any attached events Note that it is an error to destroy an EventMap if it is currently pointed to by any events using 'setChain'. Make sure to remove any such chaining before the event map is destroyed.

5.55.3 Member Function Documentation

5.55.3.1 Add()

```
const Error * Add (
            Event * eventObjPntr )
```

Add the passed event to the list of events pending on this map.

If the event is already mapped to a map (this one or another) then this function will return &EventError::AlreadyOwned.

Parameters

eventObj	Points to the event to add

Returns

A pointer to an error object on failure, or NULL on success.

5.55.3.2 changeBits()

```
void changeBits (
            uint32 bits,
            uint32 value )  [inline]
```

Change the value of specified bits in the mask for this event.

The bits to change are identified by one parameter, and the new value for these bits is specified in the other parameter.

Parameters

bits	Identifies which bits in the mask to change. Only those bits which are set in this parameter will be effected in the event mask.
value	The new value for the bits identified in the first parameter.

5.55.3.3 clrBits()

```
void clrBits (
            uint32 bits )  [inline]
```

Clear bits in the event mask.

Parameters

bits	Any bit set in this parameter will be cleared in the event mask.

5.55.3.4 getMask()

```
uint32 getMask (
            void  )  [inline]
```

Get the current value of the mask for this event map.

Returns

The 32-bit mask value.

5.55.3.5 Remove()

```
const Error * Remove (
            Event * eventObjPntr )
```

Remove the passed event from the list of events pending on this map.

If the event is not presently attached to this map, then this function will return &EventError::NotMapped.

Parameters

e	Points to the event to remove

Returns

A pointer to an error object on failure, or NULL on success.

5.55.3.6 setBits()

```
void setBits (
            uint32 bits )  [inline]
```

Set bits in the event mask.

Parameters

bits	Any bit set in this parameter will be set in the event mask.

5.55.3.7 setMask()

```
void setMask (
            uint32 mask )  [inline]
```

Update the event mask.

The new mask value will equal the passed parameter.

Parameters

mask	The new mask value.

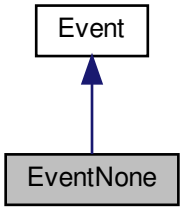
The documentation for this class was generated from the following files:

- CML_EventMap.h
- EventMap.cpp

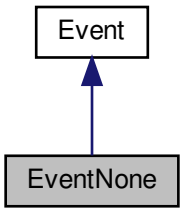
5.56 EventNone Class Reference

This is an event that matches if none of a group of bits are set in the EventMap mask.

Inheritance diagram for EventNone:



Collaboration diagram for EventNone:



Public Member Functions

- EventNone (uint32 v=0)

  Construct a new object specifying a group of bits that should be checked.
- EventNone (const EventNone &e)

  Construct a new event object using the value of a passed event.
- virtual bool isTrue (uint32 mask)

  Check the event against the passed mask.

Additional Inherited Members

5.56.1 Detailed Description

This is an event that matches if none of a group of bits are set in the EventMap mask.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 EventNone() [1/2]

```
EventNone (
            uint32 v = 0 )  [inline]
```

Construct a new object specifying a group of bits that should be checked.

Parameters

v	The bit mask that the event will wait on. Default is zero

5.56.2.2 EventNone() [2/2]

```
EventNone (
            const EventNone & e )  [inline]
```

Construct a new event object using the value of a passed event.

Parameters

e	Another event object who's value will be used to initialize this one.

5.56.3 Member Function Documentation

5.56.3.1 isTrue()

```
virtual bool isTrue (
            uint32 mask )  [inline], [virtual]
```

Check the event against the passed mask.

If none of the selected bits are set in the mask, then the event is satisfied.

Parameters

mask	The mask to test against

Returns

true if none bits of interest are set in the mask.

Reimplemented from Event.

The documentation for this class was generated from the following file:

- CML_EventMap.h

5.57 Filter Class Reference

Generic filter structure.

Public Member Functions

- Filter (void)
  Default constructor for filter object.
- const Error * LoadFromCCX (int32 coef[ ], int ct)
  Load the filter coefficients from an array of integer values read from a .ccx file.
- void getIntCoef (int16 &a1, int16 &a2, int16 &b0, int16 &b1, int16 &b2, int16 &k)
  Return the filter coefficients in integer format as used by DSP based amplifiers.
- void getFloatCoef (float &a1, float &a2, float &b0, float &b1, float &b2)
  Return the filter coefficients in floating point format as used by FPGA based amplifiers.

5.57.1 Detailed Description

Generic filter structure.

This structure holds the coefficients used by the amplifier in various configurable filters.

5.57.2 Constructor & Destructor Documentation

5.57.2.1 Filter()

```
Filter (
            void  )
```

Default constructor for filter object.

Simply sets all coefficients to zero.

5.57.3 Member Function Documentation

5.57.3.1 LoadFromCCX()

```
const Error * LoadFromCCX (
            int32 coef[],
            int ct )
```

Load the filter coefficients from an array of integer values read from a .ccx file.

For DSP based amps the ccx file contains an array of 9 short integers holding the coefficient data. For FPGA based amps the ccx file holds an array of 7 long integer values.

Parameters

coef	The array of coefficient data read from the ccx file
ct	The number of coefficients (should be either 7 or 9)

Returns

An error pointer or NULL on success

The documentation for this class was generated from the following files:

- CML_Filter.h
- Filter.cpp

5.58 Firmware Class Reference

Copley Controls amplifier firmware object.

Public Member Functions

- uint16 getFileVersion ()
  Returns the firmware file version number.
- uint16 getAmpType ()
  Returns the amplifier type for this firmware.
- uint32 getStart ()
  Returns the firmware starting address.
- uint32 getLength ()
  Returns the firmware length (in words)
- uint16 * getData (void)
  Returns the firmware binary data.
- virtual void progress (uint32 addr)
  This virtual function is called repeatedly during an amplifier firmware update.

5.58.1 Detailed Description

Copley Controls amplifier firmware object.

This object is used to represent a firmware file which can be uploaded to an amplifier.

Note that uploading firmware to the amplifier is not part of normal operation. The amplifier firmware is stored in internal Flash memory, and only needs to be updated if a new version with new features or bug fixes is produced by Copley Controls Corporation.

5.58.2 Member Function Documentation

5.58.2.1 getAmpType()

```
uint16 getAmpType ( )  [inline]
```

Returns the amplifier type for this firmware.

Returns

The amplifier type for this firmware

5.58.2.2 getData()

```
uint16* getData (
            void  )  [inline]
```

Returns the firmware binary data.

Returns

The firmware binary data

5.58.2.3 getFileVersion()

```
uint16 getFileVersion ( )  [inline]
```

Returns the firmware file version number.

Returns

The firmware file version number

5.58.2.4 getLength()

```
uint32 getLength ( )  [inline]
```

Returns the firmware length (in words)

Returns

The firmware length (in words)

5.58.2.5 getStart()

```
uint32 getStart ( )  [inline]
```

Returns the firmware starting address.

Returns

The firmware starting address

5.58.2.6 progress()

```
virtual void progress (
            uint32 addr )  [inline], [virtual]
```

This virtual function is called repeatedly during an amplifier firmware update.

It can be overloaded to display the progress of the download. This version does nothing.

Parameters

addr	The address currently being downloaded.

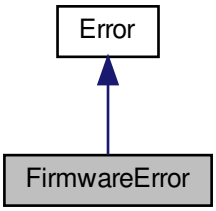
The documentation for this class was generated from the following file:

- CML_Firmware.h

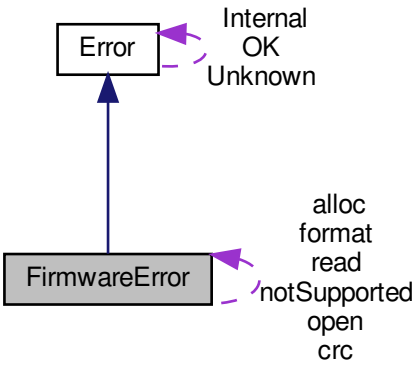
5.59 FirmwareError Class Reference

This class represents error conditions that can occur while accessing a Copley Controls amplifier firmware object.

Inheritance diagram for FirmwareError:



Collaboration diagram for FirmwareError:



Static Public Attributes

- static const FirmwareError open
  Unable to open specified firmware file.
- static const FirmwareError read
  Error reading from firmware file.
- static const FirmwareError format
  File format error.
- static const FirmwareError crc
  File CRC error.
- static const FirmwareError alloc
  Memory allocation error.
- static const FirmwareError notSupported
  Firmware update not supported.

Protected Member Functions

- FirmwareError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.59.1 Detailed Description

This class represents error conditions that can occur while accessing a Copley Controls amplifier firmware object.

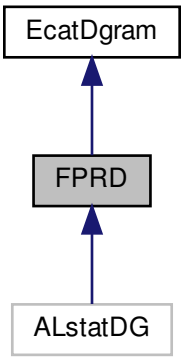
The documentation for this class was generated from the following file:

- CML_Firmware.h

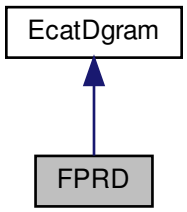
5.60 FPRD Struct Reference

Read by assigned node ID (Configured Address Physical Read) The master assigns each node a unique 16-bit address at startup.

Inheritance diagram for FPRD:



Collaboration diagram for FPRD:



Additional Inherited Members

5.60.1 Detailed Description

Read by assigned node ID (Configured Address Physical Read) The master assigns each node a unique 16-bit address at startup.

This datagram reads from memory locations within the slave based on these assigned addresses

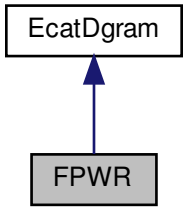
The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

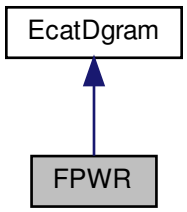
5.61 FPWR Struct Reference

Write by assigned node ID (Configured Address Physical Write)

Inheritance diagram for FPWR:



Collaboration diagram for FPWR:



Additional Inherited Members

5.61.1 Detailed Description

Write by assigned node ID (Configured Address Physical Write)

The documentation for this struct was generated from the following file:

- CML_EtherCAT.h

5.62 FuncGenConfig Struct Reference

Configuration parameters for amplifier's internal function generator.

Public Member Functions

- FuncGenConfig (void)
  Default constructor, sets all members to zero.

Public Attributes

- int16 cfg
  Configuration.
- int16 duty
  Duty cycle in 0.1% (i.e. 0 to 1000)
- int16 freq
  Frequency (Hz)
- int32 amp
  Amplitude.

5.62.1 Detailed Description

Configuration parameters for amplifier's internal function generator.

5.62.2 Member Data Documentation

5.62.2.1 amp

```
int32 amp
```

Amplitude.

Units depend on what the function generator is driving 0.01 Amps for current. 0.1 encoder counts/sec for velocitiy. Encoder counts for position.

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.63 GainScheduling Struct Reference

Configuration structure used to set up the Gain Scheduling.

Public Member Functions

- GainScheduling (void)
  Default constructor.

Public Attributes

- uint32 gainSchedulingConfig
  Gain Scheduling configuration. See documentation for details.

5.63.1 Detailed Description

Configuration structure used to set up the Gain Scheduling.

These settings may be up/download from the amplifier using the functions Amp::SetGainScheduling and Amp::Get↩ GainScheduling.

5.63.2 Constructor & Destructor Documentation

5.63.2.1 GainScheduling()

```
GainScheduling (
            void  )  [inline]
```

Default constructor.

Initializes all structure elements to zero.

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.64 HomeConfig Struct Reference

Homing parameter structure.

Public Member Functions

- HomeConfig (void)
  Default constructor, just set the method to none and the other parameters to zero.

Public Attributes

- COPLEY_HOME_METHOD method
  Homing method to use.
- uint16 extended
  Extended home method.
- uunit velFast
  Velocity to use for fast moves during the home procedure.
- uunit velSlow
  Velocity to use when seeking a sensor edge.
- uunit accel
  Acceleration to use for the home procuedure.
- uunit offset
  Offset from located home position to zero position.
- int16 current
  Home current limit.
- int16 delay
  Home delay.

5.64.1 Detailed Description

Homing parameter structure.

This structure allows all homing parameters to be grouped together and passed to the amplifier for convenience.

5.64.2 Constructor & Destructor Documentation

5.64.2.1 HomeConfig()

```
HomeConfig (
            void  )  [inline]
```

Default constructor, just set the method to none and the other parameters to zero.

5.64.3 Member Data Documentation

5.64.3.1 accel

```
uunit accel
```

Acceleration to use for the home procuedure.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.64.3.2 current

```
int16 current
```

Home current limit.

This parameter is only used when running in one of the 'home to hard stop' homing modes. In all other modes this parameter may be left uninitialized. The current should be specified in units of 0.01 Amps (i.e. 100 for 1.0 Amp)

5.64.3.3 delay

```
int16 delay
```

Home delay.

This parameter is only used when running in one of the 'home to hard stop' homing modes. In all other modes this parameter may be left uninitialized. The delay is specified in units of milliseconds.

5.64.3.4 extended

```
uint16 extended
```

Extended home method.

If the main home method is set to any value other then 'CHM_EXTENDED' then this parameter is ignored. If the home method is set to this value, then this value will be used to define the low level homing routine used by the amplifier.

For the most part this parameter can be ignored. It's intended to allow access to some low level features of the amplifier's homing state machine which are otherwise not available through the more generic home methods.

Encodings for this parameter can be found in the CANopen programmers guide for CANopen object 0x2352, or in the serial port programmers guide for variable 0xC2.

5.64.3.5 offset

```
uunit offset
```

Offset from located home position to zero position.

After the home position is found as defined by the home method, this offset will be added to it and the resulting position will be considered the zero position. This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.64.3.6 velFast

```
uunit velFast
```

Velocity to use for fast moves during the home procedure.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.64.3.7 velSlow

```
uunit velSlow
```

Velocity to use when seeking a sensor edge.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.65 InputShaper Class Reference

Generic input shaper structure.

Public Member Functions

- InputShaper (void)
  Default constructor for InputShaper object.
- const Error * LoadFromCCX (int32 inputShaping[ ], int ct)
  Load the filter coefficients from an array of integer values read from a .ccx file.
- void setInputShapeFilter (float inputShaperData[ ])
  Return the amplitude for each impulse in the input shaping filter.

5.65.1 Detailed Description

Generic input shaper structure.

This structure holds the amplitudes and times of the impulse functions used to create the input shaper, as well as the info used by CME2 to indentify the filter type and settings.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 InputShaper()

```
InputShaper (
            void  )
```

Default constructor for InputShaper object.

Simply sets all impulses to 0

5.65.3 Member Function Documentation

5.65.3.1 LoadFromCCX()

```
const Error * LoadFromCCX (
            int32 inputShaping[],
            int ct )
```

Load the filter coefficients from an array of integer values read from a .ccx file.

The .ccx file holds an array of 20 long integer values for the input shaping filter.

Parameters

inputShaping	The array of input shaping data read from the ccx file
ct	The number of coefficients (should be 20)

Returns

An error pointer or NULL on success

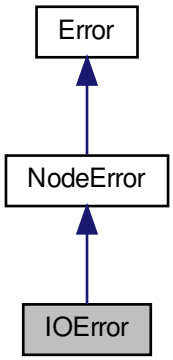
The documentation for this class was generated from the following files:

- CML_InputShaper.h
- InputShaper.cpp

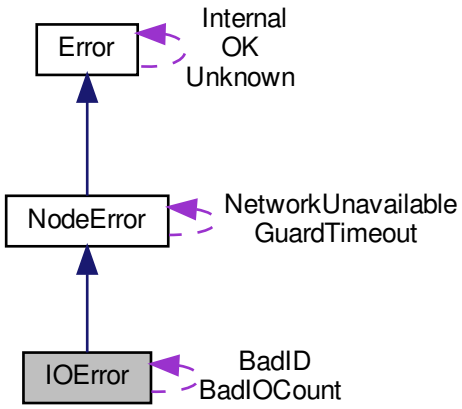
5.66 IOError Class Reference

I/O module errors.

Inheritance diagram for IOError:



Collaboration diagram for IOError:



Static Public Attributes

- static const IOError BadID

  The passed digital I/O pin ID number is invalid.
- static const IOError BadIOCount

  The number of passed I/O ID blocks is invalid.

Protected Member Functions

- IOError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.66.1 Detailed Description

I/O module errors.

This class is used to represent errors that may be returned by a standard I/O module.

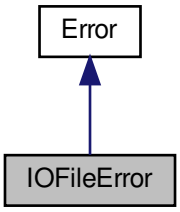
The documentation for this class was generated from the following file:

- CML_IO.h

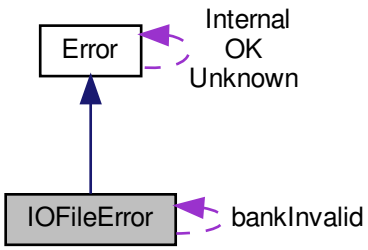
5.67 IOFileError Class Reference

This class represents error conditions that can occur when loading IO module data from a data file.

Inheritance diagram for IOFileError:



Collaboration diagram for IOFileError:



Static Public Attributes

- static const IOFileError bankInvalid

  I/O bank invalid.

Protected Member Functions

- IOFileError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.67.1 Detailed Description

This class represents error conditions that can occur when loading IO module data from a data file.

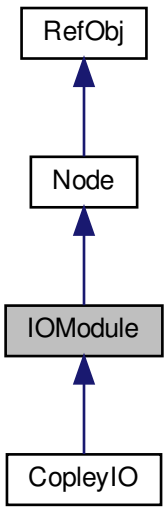
The documentation for this class was generated from the following file:

- CML_CopleyIO.h

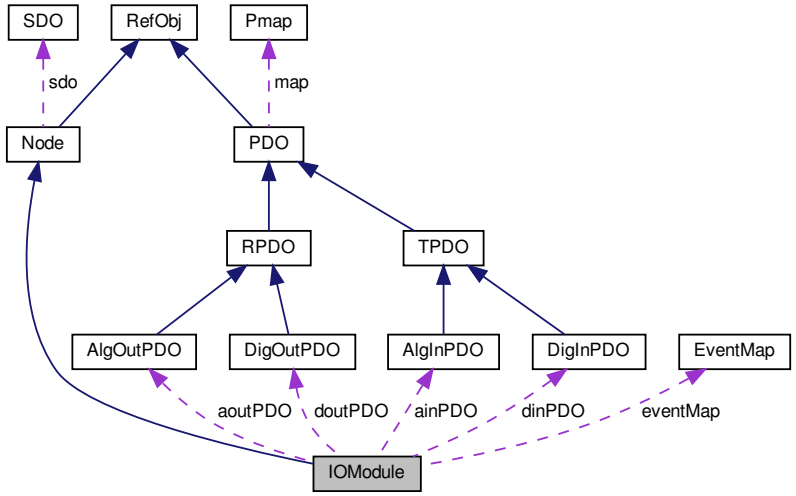
5.68 IOModule Class Reference

Standard CANopen I/O module.

Inheritance diagram for IOModule:



Collaboration diagram for IOModule:



Classes

- class AlgInPDO

  Transmit PDO for mapping analog inputs.
- class AlgOutPDO

  Receive PDO for mapping analog outputs.
- class DigInPDO

  Transmit PDO for mapping digital inputs.
- class DigOutPDO

  Receive PDO for mapping digital output pins.

Public Member Functions

- IOModule (void)

  Default constructor for an I/O module.
- IOModule (Network &net, int16 nodeID)

  Construct an IOModule object and initialize it using default settings.
- IOModule (Network &net, int16 nodeID, IOModuleSettings &settings)

  Construct an IOModule object and initialize it using custom settings.
- virtual ~IOModule ()

  Virtual destructor for the IOModule object.
- virtual const Error * Init (Network &net, int16 nodeID)

  Initialize an I/O module using default settings.
- virtual const Error * Init (Network &net, int16 nodeID, IOModuleSettings &settings)

  Initialize an I/O module using custom settings.

- virtual const Error * WaitIOEvent (IOMODULE_EVENTS event, Timeout timeout=-1)

  Wait on an event associated with this I/O module.
- virtual const Error * WaitIOEvent (Event &e, Timeout timeout, IOMODULE_EVENTS &match)

  Wait for an event associated with this I/O module.

Digital input control

If the module contains digital inputs, these methods may be used to configure and read those inputs.

The inputs may be read and controlled individually, or in groups of 8, 16 or 32 inputs.

All I/O modules should support access to digital inputs in groups of 8. Support for individual access or different groupings is optional under the spec. If a particular device does not support such groupings, an attempt to use them should return an error code.

Each input pin or group of pins is assigned an ID number used to access it. When single inputs are accessed, these ID numbers range from 0 (the first input) to N-1 (the last input), where N is the total number of input pins available on the module.

When groups of inputs are accessed as a unit, the group is assigned a number. The first group of inputs will be assigned ID number 0, the second will be ID 1, etc. The number of groups of a particular size will be the total number of inputs divided by the group size.

For example, to access the fifty third input pin individually you would use id number 52. To access it as part of a group of 8 inputs, you would access group number 6 (52/8). Input 52 would be bit 4 (52%8) of that group.

- virtual const Error * DinGetIntEna (bool &value)

  Get the current setting of the global interrupt enable for digital inputs.
- virtual const Error * DinSetIntEna (bool value)

  Set the current setting of the global interrupt enable for digital inputs.
- virtual const Error * DinGetCt (uint16 &ct)

  Return the number of individual inputs available on this device.
- virtual const Error * DinRead (uint16 id, bool &value, bool viaSDO=false)

  Read a single digital input.
- virtual const Error * DinGetPol (uint16 id, bool &value)

  Get the current polarity settings for a digital input.
- virtual const Error * DinSetPol (uint16 id, bool value)

  Set the current polarity setting for a digital input.
- virtual const Error * DinGetFilt (uint16 id, bool &value)

  Get the current filter constant setting for a digital input.
- virtual const Error * DinSetFilt (uint16 id, bool value)

  Set the current filter constant setting for a digital input.
- virtual const Error * DinGetMaskAny (uint16 id, bool &value)

  Get the 'any transition' interrupt mask settings for a digital input.
- virtual const Error * DinSetMaskAny (uint16 id, bool value)

  Set the 'any transition' interrupt mask settings for a digital input.
- virtual const Error * DinGetMaskLow2High (uint16 id, bool &value)

  Get the 'low to high' interrupt mask settings for a digital input.
- virtual const Error * DinSetMaskLow2High (uint16 id, bool value)

  Set the 'low to high' interrupt mask settings for a digital input.
- virtual const Error * DinGetMaskHigh2Low (uint16 id, bool &value)

  Get the 'high to low' interrupt mask settings for a digital input.
- virtual const Error * DinSetMaskHigh2Low (uint16 id, bool value)

  Set the 'high to low' interrupt mask settings for a digital input.
- virtual const Error * Din8GetCt (uint8 &ct)

  Return the number of 8-bit groups of inputs available on this device.
- virtual const Error * Din8Read (uint8 id, uint8 &value, bool viaSDO=false)

  Read a group of 8 digital inputs.

- virtual const Error * Din8GetPol (uint8 id, uint8 &value)
  Get the current polarity settings for a group of 8 digital inputs.
- virtual const Error * Din8SetPol (uint8 id, uint8 value)
  Set the current polarity setting for a group of 8 digital inputs.
- virtual const Error * Din8GetFilt (uint8 id, uint8 &value)
  Get the current filter constant settings for a group of 8 digital inputs.
- virtual const Error * Din8SetFilt (uint8 id, uint8 value)
  Set the current filter constant setting for a group of 8 digital inputs.
- virtual const Error * Din8GetMaskAny (uint8 id, uint8 &value)
  Get the 'any transition' interrupt mask settings for a group of 8 digital inputs.
- virtual const Error * Din8SetMaskAny (uint8 id, uint8 value)
  Set the 'any transition' interrupt mask settings for a group of 8 digital inputs.
- virtual const Error * Din8GetMaskLow2High (uint8 id, uint8 &value)
  Get the 'low to high' interrupt mask settings for a group of 8 digital inputs.
- virtual const Error * Din8SetMaskLow2High (uint8 id, uint8 value)
  Set the 'low to high' interrupt mask settings for a group of 8 digital inputs.
- virtual const Error * Din8GetMaskHigh2Low (uint8 id, uint8 &value)
  Get the 'high to low' interrupt mask settings for a group of 8 digital inputs.
- virtual const Error * Din8SetMaskHigh2Low (uint8 id, uint8 value)
  Set the 'high to low' interrupt mask settings for a group of 8 digital inputs.
- virtual const Error * Din16GetCt (uint8 &ct)
  Return the number of 16-bit groups of inputs available on this device.
- virtual const Error * Din16Read (uint8 id, uint16 &value, bool viaSDO=false)
  Read a group of 16 digital inputs.
- virtual const Error * Din16GetPol (uint8 id, uint16 &value)
  Get the current polarity settings for a group of 16 digital inputs.
- virtual const Error * Din16SetPol (uint8 id, uint16 value)
  Set the current polarity setting for a group of 16 digital inputs.
- virtual const Error * Din16GetFilt (uint8 id, uint16 &value)
  Get the current filter constant settings for a group of 16 digital inputs.
- virtual const Error * Din16SetFilt (uint8 id, uint16 value)
  Set the current filter constant setting for a group of 16 digital inputs.
- virtual const Error * Din16GetMaskAny (uint8 id, uint16 &value)
  Get the 'any transition' interrupt mask settings for a group of 16 digital inputs.
- virtual const Error * Din16SetMaskAny (uint8 id, uint16 value)
  Set the 'any transition' interrupt mask settings for a group of 16 digital inputs.
- virtual const Error * Din16GetMaskLow2High (uint8 id, uint16 &value)
  Get the 'low to high' interrupt mask settings for a group of 16 digital inputs.
- virtual const Error * Din16SetMaskLow2High (uint8 id, uint16 value)
  Set the 'low to high' interrupt mask settings for a group of 16 digital inputs.
- virtual const Error * Din16GetMaskHigh2Low (uint8 id, uint16 &value)
  Get the 'high to low' interrupt mask settings for a group of 16 digital inputs.
- virtual const Error * Din16SetMaskHigh2Low (uint8 id, uint16 value)
  Set the 'high to low' interrupt mask settings for a group of 16 digital inputs.
- virtual const Error * Din32GetCt (uint8 &ct)
  Return the number of 32-bit groups of inputs available on this device.
- virtual const Error * Din32Read (uint8 id, uint32 &value, bool viaSDO=false)
  Read a group of 32 digital inputs.
- virtual const Error * Din32GetPol (uint8 id, uint32 &value)
  Get the current polarity settings for a group of 32 digital inputs.
- virtual const Error * Din32SetPol (uint8 id, uint32 value)
  Set the current polarity setting for a group of 32 digital inputs.
- virtual const Error * Din32GetFilt (uint8 id, uint32 &value)
  Get the current filter constant settings for a group of 32 digital inputs.
- virtual const Error * Din32SetFilt (uint8 id, uint32 value)

Set the current filter constant setting for a group of 32 digital inputs.

- virtual const Error * Din32GetMaskAny (uint8 id, uint32 &value)

  Get the 'any transition' interrupt mask settings for a group of 32 digital inputs.
- virtual const Error * Din32SetMaskAny (uint8 id, uint32 value)

  Set the 'any transition' interrupt mask settings for a group of 32 digital inputs.
- virtual const Error * Din32GetMaskLow2High (uint8 id, uint32 &value)

  Get the 'low to high' interrupt mask settings for a group of 32 digital inputs.
- virtual const Error * Din32SetMaskLow2High (uint8 id, uint32 value)

  Set the 'low to high' interrupt mask settings for a group of 32 digital inputs.
- virtual const Error * Din32GetMaskHigh2Low (uint8 id, uint32 &value)

  Get the 'high to low' interrupt mask settings for a group of 32 digital inputs.
- virtual const Error * Din32SetMaskHigh2Low (uint8 id, uint32 value)

  Set the 'high to low' interrupt mask settings for a group of 32 digital inputs.

Digital output control

If the module contains digital outputs, these methods may be used to configure and set those outputs.

The outputs may be set and controlled individually, or in groups of 8, 16 or 32 outputs.

All I/O modules should support access to digital outputs in groups of 8. Support for individual access or different groupings is optional under the spec. If a particular device does not support such groupings, an attempt to use them should return an error code.

Each output pin or group of pins is assigned an ID number used to access it. When single outputs are accessed, these ID numbers range from 0 (the first output) to N-1 (the last output), where N is the total number of output pins available on the module.

When groups of outputs are accessed as a unit, the group is assigned a number. The first group of outputs will be assigned ID number 0, the second will be ID 1, etc. The number of groups of a particular size will be the total number of outputs divided by the group size.

For example, to access the twenty seventy output pin individually you would use id number 26. To access it as part of a group of 8 outputs, you would access group number 3 (26/8). Output 26 would be bit 2 (26%8) of that group.

- virtual const Error * DoutGetCt (uint16 &ct)

  Return the number of individual outputs available on this device.
- virtual const Error * DoutWrite (uint16 id, bool value, bool viaSDO=false)

  Write an individual digital output.
- virtual const Error * DoutGetPol (uint16 id, bool &value)

  Get the current polarity setting for an individual digital output.
- virtual const Error * DoutSetPol (uint16 id, bool value)

  Set the current polarity setting for an individual digital output.
- virtual const Error * DoutGetFilt (uint16 id, bool &value)

  Get the current filter constant setting for an individual digital output.
- virtual const Error * DoutSetFilt (uint16 id, bool value)

  Set the current filter constant setting for an individual digital output.
- virtual const Error * DoutGetErrMode (uint16 id, bool &value)

  Get the current error mode setting for an individual digital output.
- virtual const Error * DoutSetErrMode (uint16 id, bool value)

  Set the current error mode setting for an individual digital output.
- virtual const Error * DoutGetErrValue (uint16 id, bool &value)

  Get the current error value setting for an individual digital output.
- virtual const Error * DoutSetErrValue (uint16 id, bool value)

  Set the current error value setting for an individual digital output.
- virtual const Error * Dout8GetCt (uint8 &ct)

  Return the number of 8-bit groups of outputs available on this device.
- const Error * Dout8Write (uint8 id, uint8 value, bool viaSDO=false)

Write a group of 8 digital outputs.

- const Error * Dout8Read (uint8 id, uint8 &value)

  Read back the last value written to this bank of 8 digital outputs.
- virtual const Error * Dout8GetPol (uint8 id, uint8 &value)

  Get the current polarity settings for a group of 8 digital outputs.
- virtual const Error * Dout8SetPol (uint8 id, uint8 value)

  Set the current polarity setting for a group of 8 digital outputs.
- virtual const Error * Dout8GetFilt (uint8 id, uint8 &value)

  Get the current filter constant settings for a group of 8 digital outputs.
- virtual const Error * Dout8SetFilt (uint8 id, uint8 value)

  Set the current filter constant setting for a group of 8 digital outputs.
- virtual const Error * Dout8GetErrMode (uint8 id, uint8 &value)

  Get the current error mode settings for a group of 8 digital outputs.
- virtual const Error * Dout8SetErrMode (uint8 id, uint8 value)

  Set the current error mode settings for a group of 8 digital outputs.
- virtual const Error * Dout8GetErrValue (uint8 id, uint8 &value)

  Get the current error value settings for a group of 8 digital outputs.
- virtual const Error * Dout8SetErrValue (uint8 id, uint8 value)

  Set the current error value settings for a group of 8 digital outputs.
- virtual const Error * Dout16GetCt (uint8 &ct)

  Return the number of 16-bit groups of outputs available on this device.
- virtual const Error * Dout16Write (uint8 id, uint16 value, bool viaSDO=false)

  Write a group of 16 digital outputs.
- const Error * Dout16Read (uint8 id, uint16 &value)

  Read back the last value written to this bank of 16 digital outputs.
- virtual const Error * Dout16GetPol (uint8 id, uint16 &value)

  Get the current polarity settings for a group of 16 digital outputs.
- virtual const Error * Dout16SetPol (uint8 id, uint16 value)

  Set the current polarity setting for a group of 16 digital outputs.
- virtual const Error * Dout16GetFilt (uint8 id, uint16 &value)

  Get the current filter constant settings for a group of 16 digital outputs.
- virtual const Error * Dout16SetFilt (uint8 id, uint16 value)

  Set the current filter constant setting for a group of 16 digital outputs.
- virtual const Error * Dout16GetErrMode (uint8 id, uint16 &value)

  Get the current error mode settings for a group of 16 digital outputs.
- virtual const Error * Dout16SetErrMode (uint8 id, uint16 value)

  Set the current error mode settings for a group of 16 digital outputs.
- virtual const Error * Dout16GetErrValue (uint8 id, uint16 &value)

  Get the current error value settings for a group of 16 digital outputs.
- virtual const Error * Dout16SetErrValue (uint8 id, uint16 value)

  Set the current error value settings for a group of 16 digital outputs.
- virtual const Error * Dout32GetCt (uint8 &ct)

  Return the number of 32-bit groups of outputs available on this device.
- virtual const Error * Dout32Write (uint8 id, uint32 value, bool viaSDO=false)

  Write a group of 32 digital outputs.
- const Error * Dout32Read (uint8 id, uint32 &value)

  Read back the last value written to this bank of 32 digital outputs.
- virtual const Error * Dout32GetPol (uint8 id, uint32 &value)

  Get the current polarity settings for a group of 32 digital outputs.
- virtual const Error * Dout32SetPol (uint8 id, uint32 value)

  Set the current polarity setting for a group of 32 digital outputs.
- virtual const Error * Dout32GetFilt (uint8 id, uint32 &value)

  Get the current filter constant settings for a group of 32 digital outputs.
- virtual const Error * Dout32SetFilt (uint8 id, uint32 value)

  Set the current filter constant setting for a group of 32 digital outputs.

- virtual const Error * Dout32GetErrMode (uint8 id, uint32 &value)
  Get the current error mode settings for a group of 32 digital outputs.
- virtual const Error * Dout32SetErrMode (uint8 id, uint32 value)
  Set the current error mode settings for a group of 32 digital outputs.
- virtual const Error * Dout32GetErrValue (uint8 id, uint32 &value)
  Get the current error value settings for a group of 32 digital outputs.
- virtual const Error * Dout32SetErrValue (uint8 id, uint32 value)
  Set the current error value settings for a group of 32 digital outputs.

Analog input control

If the module contains analog inputs, these methods may be used to configure and read those inputs.

Most manufacturers support 16-bit access to analog inputs. Other input sizes are optional in the spec. and may or may not be available.

- virtual const Error * Ain8GetCt (uint8 &ct)
  Return the number of 8-bit analog inputs available on this device.
- virtual const Error * Ain8Read (uint8 id, int8 &value)
  Read an 8-bit analog input.
- virtual const Error * Ain16GetCt (uint8 &ct)
  Return the number of 16-bit analog inputs available on this device.
- virtual const Error * Ain16Read (uint8 id, int16 &value, bool viaSDO=false)
  Read a 16-bit analog input.
- virtual const Error * Ain32GetCt (uint8 &ct)
  Return the number of 32-bit analog inputs available on this device.
- virtual const Error * Ain32Read (uint8 id, int32 &value)
  Read a 32-bit analog input.
- virtual const Error * AinFltGetCt (uint8 &ct)
  Return the number of floating point analog inputs available on this device.
- virtual const Error * AinFltRead (uint8 id, float &value)
  Read a floating point analog input.
- virtual const Error * Ain32GetOffset (uint8 id, int32 &value)
  Get the analog input offset value as a 32-bit integer.
- virtual const Error * Ain32SetOffset (uint8 id, int32 value)
  Set the analog input offset value as a 32-bit integer.
- virtual const Error * Ain32GetScaling (uint8 id, int32 &value)
  Get the analog input scaling factor as a 32-bit integer.
- virtual const Error * Ain32SetScaling (uint8 id, int32 value)
  Set the analog input scaling factor as a 32-bit integer.
- virtual const Error * AinFltGetOffset (uint8 id, float &value)
  Get the analog input offset value as a floating point value.
- virtual const Error * AinFltSetOffset (uint8 id, float value)
  Set the analog input offset value as a floating point value.
- virtual const Error * AinFltGetScaling (uint8 id, float &value)
  Get the analog input scaling factor as a floating point value.
- virtual const Error * AinFltSetScaling (uint8 id, float value)
  Set the analog input scaling factor as a floating point value.
- virtual const Error * AinGetIntEna (bool &value)
  Get the current setting of the global interrupt enable for analog inputs.
- virtual const Error * AinSetIntEna (bool value)
  Set the current setting of the global interrupt enable for analog inputs.
- virtual const Error * AinGetTrigType (uint8 id, IO_AIN_TRIG_TYPE &value)
  Get the analog input trigger type associated with the input channel.
- virtual const Error * AinSetTrigType (uint8 id, IO_AIN_TRIG_TYPE value)
  Set the analog input trigger type associated with the input channel.

- virtual const Error * AinGetIntSource (uint8 id, uint32 &value)
  Get the analog input interrupt source.
- virtual const Error * Ain16GetUpperLimit (uint8 id, int16 &value)
  Get the analog input upper limit value as a 16-bit integer.
- virtual const Error * Ain16SetUpperLimit (uint8 id, int16 value)
  Set the analog input upper limit value as a 16-bit integer.
- virtual const Error * Ain16GetLowerLimit (uint8 id, int16 &value)
  Get the analog input lower limit value as a 16-bit integer.
- virtual const Error * Ain16SetLowerLimit (uint8 id, int16 value)
  Set the analog input lower limit value as a 16-bit integer.
- virtual const Error * Ain16GetUnsignedDelta (uint8 id, int16 &value)
  Get the analog input unsigned delta value as a 16-bit integer.
- virtual const Error * Ain16SetUnsignedDelta (uint8 id, int16 value)
  Set the analog input unsigned delta value as a 16-bit integer.
- virtual const Error * Ain16GetNegativeDelta (uint8 id, int16 &value)
  Get the analog input negative delta value as a 16-bit integer.
- virtual const Error * Ain16SetNegativeDelta (uint8 id, int16 value)
  Set the analog input negative delta value as a 16-bit integer.
- virtual const Error * Ain16GetPositiveDelta (uint8 id, int16 &value)
  Get the analog input positive delta value as a 16-bit integer.
- virtual const Error * Ain16SetPositiveDelta (uint8 id, int16 value)
  Set the analog input positive delta value as a 16-bit integer.
- virtual const Error * Ain32GetUpperLimit (uint8 id, int32 &value)
  Get the analog input upper limit value as a 32-bit integer.
- virtual const Error * Ain32SetUpperLimit (uint8 id, int32 value)
  Set the analog input upper limit value as a 32-bit integer.
- virtual const Error * Ain32GetLowerLimit (uint8 id, int32 &value)
  Get the analog input lower limit value as a 32-bit integer.
- virtual const Error * Ain32SetLowerLimit (uint8 id, int32 value)
  Set the analog input lower limit value as a 32-bit integer.
- virtual const Error * Ain32GetUnsignedDelta (uint8 id, int32 &value)
  Get the analog input unsigned delta value as a 32-bit integer.
- virtual const Error * Ain32SetUnsignedDelta (uint8 id, int32 value)
  Set the analog input unsigned delta value as a 32-bit integer.
- virtual const Error * Ain32GetNegativeDelta (uint8 id, int32 &value)
  Get the analog input negative delta value as a 32-bit integer.
- virtual const Error * Ain32SetNegativeDelta (uint8 id, int32 value)
  Set the analog input negative delta value as a 32-bit integer.
- virtual const Error * Ain32GetPositiveDelta (uint8 id, int32 &value)
  Get the analog input positive delta value as a 32-bit integer.
- virtual const Error * Ain32SetPositiveDelta (uint8 id, int32 value)
  Set the analog input positive delta value as a 32-bit integer.
- virtual const Error * AinFltGetUpperLimit (uint8 id, float &value)
  Get the analog input upper limit value as a floating point value.
- virtual const Error * AinFltSetUpperLimit (uint8 id, float value)
  Set the analog input upper limit value as a floating point value.
- virtual const Error * AinFltGetLowerLimit (uint8 id, float &value)
  Get the analog input lower limit value as a floating point value.
- virtual const Error * AinFltSetLowerLimit (uint8 id, float value)
  Set the analog input lower limit value as a floating point value.
- virtual const Error * AinFltGetUnsignedDelta (uint8 id, float &value)
  Get the analog input unsigned delta value as a floating point value.
- virtual const Error * AinFltSetUnsignedDelta (uint8 id, float value)
  Set the analog input unsigned delta value as a floating point value.
- virtual const Error * AinFltGetNegativeDelta (uint8 id, float &value)

Get the analog input negative delta value as a floating point value.

- virtual const Error * AinFltSetNegativeDelta (uint8 id, float value)

  Set the analog input negative delta value as a floating point value.
- virtual const Error * AinFltGetPositiveDelta (uint8 id, float &value)

  Get the analog input positive delta value as a floating point value.
- virtual const Error * AinFltSetPositiveDelta (uint8 id, float value)

  Set the analog input positive delta value as a floating point value.

Analog output control

If the module contains analog outputs, these methods may be used to configure and write to those outputs.

Most manufacturers support 16-bit access to analog inputs. Other input sizes are optional in the spec. and may or may not be available.

- virtual const Error * Aout8GetCt (uint8 &ct)

  Return the number of 8-bit analog outputs available on this device.
- virtual const Error * Aout8Write (uint8 id, int8 value)

  Write to an 8-bit analog output.
- virtual const Error * Aout16GetCt (uint8 &ct)

  Return the number of 16-bit analog outputs available on this device.
- virtual const Error * Aout16Write (uint8 id, int16 value, bool viaSDO=false)

  Write to a 16-bit analog output.
- virtual const Error * Aout32GetCt (uint8 &ct)

  Return the number of 32-bit analog outputs available on this device.
- virtual const Error * Aout32Write (uint8 id, int32 value)

  Write to a 32-bit analog output.
- virtual const Error * AoutFltGetCt (uint8 &ct)

  Return the number of floating point analog outputs available on this device.
- virtual const Error * AoutFltWrite (uint8 id, float value)

  Write to a floating point analog output.
- virtual const Error * Aout32GetOffset (uint8 id, int32 &value)

  Get the analog output offset value as a 32-bit integer.
- virtual const Error * Aout32SetOffset (uint8 id, int32 value)

  Set the analog output offset value as a 32-bit integer.
- virtual const Error * Aout32GetScaling (uint8 id, int32 &value)

  Get the analog output scaling factor as a 32-bit integer.
- virtual const Error * Aout32SetScaling (uint8 id, int32 value)

  Set the analog output scaling factor as a 32-bit integer.
- virtual const Error * AoutFltGetOffset (uint8 id, float &value)

  Get the analog output offset value as a floating point value.
- virtual const Error * AoutFltSetOffset (uint8 id, float value)

  Set the analog output offset value as a floating point value.
- virtual const Error * AoutFltGetScaling (uint8 id, float &value)

  Get the analog output scaling factor as a floating point value.
- virtual const Error * AoutFltSetScaling (uint8 id, float value)

  Set the analog output scaling factor as a floating point value.
- virtual const Error * AoutGetErrMode (uint8 id, bool &value)

  Get the analog output error mode.
- virtual const Error * AoutSetErrMode (uint8 id, bool value)

  Set the analog output error mode.
- virtual const Error * Aout32GetErrValue (uint8 id, int32 &value)

  Get the analog output error value as a 32-bit integer.
- virtual const Error * Aout32SetErrValue (uint8 id, int32 value)

  Set the analog output error value as a 32-bit integer.
- virtual const Error * Aout16GetErrValue (uint8 id, int16 &value)

Get the analog output error value as a 16-bit integer.

- virtual const Error * Aout16SetErrValue (uint8 id, int16 value)

  Set the analog output error value as a 32-bit integer.
- virtual const Error * AoutFltGetErrValue (uint8 id, float &value)

  Get the analog output error value as a floating point value.
- virtual const Error * AoutFltSetErrValue (uint8 id, float value)

  Set the analog output error value as a floating point value.

Protected Member Functions

- const Error * BitUpld (uint16 base, uint16 id, bool &value)

  Upload a setting for a single digital I/O pin.
- const Error * BitDnld (uint16 base, uint16 id, bool value)

  Download a setting for a single digital I/O pin.
- const Error * BitCount (uint16 base, uint16 &ct)

  Count the number of individual I/O pins available on the device.
- virtual void PostIOEvent (IOMODULE_EVENTS event)

  Post an event condition to the I/O module's event map.

Protected Attributes

- DigOutPDO doutPDO

  Default PDO used to transmit digital output info.

Additional Inherited Members

5.68.1 Detailed Description

Standard CANopen I/O module.

This class represents and I/O module device conforming to the DS401 CANopen specification. The class may be extended to provide additional manufacturer specific features.

Note that the CANopen standard defines a very large number of parameters that may be used with a standard I/O module. Of these, only a small subset are requred by the spec. In practice, it seems that most of the major manufacturers of CANopen I/O modules only implement the minimum required by the spec. The result is that many of the optional functions have not been tested with real hardware due to the lack of availability. Please contact Copley Controls if you believe that you have found a problem with any of these functions.

For the typical I/O module, you can expect the following functionality to be supported based on the type of I/O the module supports:

Digital Inputs: Reading the inputs via PDO or SDO in groups of 8 should be supported. Other features are optional.

Digital Outputs: Writing to the outputs via PDO or SDO in groups of 8 should be supported. Other features are optional.

Analog Inputs: Reading 16-bit analog inputs via PDO or SDO is normally supported. Other input sizes and features are optional.

Analog Outputs: Writing 16-bit analog outputs via PDO or SDO is normally supported. Other output sizes and features are optional.

5.68.2 Constructor & Destructor Documentation

5.68.2.1 IOModule() [1/3]

```
IOModule (
            void  )
```

Default constructor for an I/O module.

Any object created using this constructor must be initialized by a call to IOModule::Init before it is used.

5.68.2.2 IOModule() [2/3]

```
IOModule (
            Network & net,
            int16 nodeID )
```

Construct an IOModule object and initialize it using default settings.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.

5.68.2.3 IOModule() [3/3]

```
IOModule (
            Network & net,
            int16 nodeID,
            IOModuleSettings & settings )
```

Construct an IOModule object and initialize it using custom settings.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.
settings	The settings to use when configuring the module

5.68.3 Member Function Documentation

5.68.3.1 Ain16GetCt()

```
virtual const Error* Ain16GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 16-bit analog inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.2 Ain16GetLowerLimit()

```
virtual const Error* Ain16GetLowerLimit (
            uint8 id,
            int16 & value )  [inline], [virtual]
```

Get the analog input lower limit value as a 16-bit integer.

The lower limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.3 Ain16GetNegativeDelta()

```
virtual const Error* Ain16GetNegativeDelta (
            uint8 id,
            int16 & value )  [inline], [virtual]
```

Get the analog input negative delta value as a 16-bit integer.

The negative delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.4 Ain16GetPositiveDelta()

```
virtual const Error* Ain16GetPositiveDelta (
            uint8 id,
            int16 & value )  [inline], [virtual]
```

Get the analog input positive delta value as a 16-bit integer.

The positive delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.5 Ain16GetUnsignedDelta()

```
virtual const Error* Ain16GetUnsignedDelta (
            uint8 id,
            int16 & value )  [inline], [virtual]
```

Get the analog input unsigned delta value as a 16-bit integer.

The unsigned delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.6 Ain16GetUpperLimit()

```
virtual const Error* Ain16GetUpperLimit (
            uint8 id,
            int16 & value )  [inline], [virtual]
```

Get the analog input upper limit value as a 16-bit integer.

The upper limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.7 Ain16Read()

```
const Error * Ain16Read (
            uint8 id,
            int16 & value,
            bool viaSDO = false )  [virtual]
```

Read a 16-bit analog input.

Parameters

id	The analog input channel ID
value	The analog input value
viaSDO	If true, read the input using SDO transfers. If false (default) use the most recently received PDO data if this input is mapped to a transmit PDO and the PDO is active.

Returns

A pointer to an error object, or NULL on success

5.68.3.8 Ain16SetLowerLimit()

```
virtual const Error* Ain16SetLowerLimit (
            uint8 id,
            int16 value )  [inline], [virtual]
```

Set the analog input lower limit value as a 16-bit integer.

The lower limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.9 Ain16SetNegativeDelta()

```
virtual const Error* Ain16SetNegativeDelta (
            uint8 id,
            int16 value )  [inline], [virtual]
```

Set the analog input negative delta value as a 16-bit integer.

The negative delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.10 Ain16SetPositiveDelta()

```
virtual const Error* Ain16SetPositiveDelta (
            uint8 id,
            int16 value )  [inline], [virtual]
```

Set the analog input positive delta value as a 16-bit integer.

The positive delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.11 Ain16SetUnsignedDelta()

```
virtual const Error* Ain16SetUnsignedDelta (
            uint8 id,
            int16 value )  [inline], [virtual]
```

Set the analog input unsigned delta value as a 16-bit integer.

The unsigned delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.12 Ain16SetUpperLimit()

```
virtual const Error* Ain16SetUpperLimit (
            uint8 id,
            int16 value )  [inline], [virtual]
```

Set the analog input upper limit value as a 16-bit integer.

The upper limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.13 Ain32GetCt()

```
virtual const Error* Ain32GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 32-bit analog inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.14 Ain32GetLowerLimit()

```
virtual const Error* Ain32GetLowerLimit (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input lower limit value as a 32-bit integer.

The lower limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.15 Ain32GetNegativeDelta()

```
virtual const Error* Ain32GetNegativeDelta (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input negative delta value as a 32-bit integer.

The negative delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.16 Ain32GetOffset()

```
virtual const Error* Ain32GetOffset (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input offset value as a 32-bit integer.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.17 Ain32GetPositiveDelta()

```
virtual const Error* Ain32GetPositiveDelta (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input positive delta value as a 32-bit integer.

The positive delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.18 Ain32GetScaling()

```
virtual const Error* Ain32GetScaling (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input scaling factor as a 32-bit integer.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.19 Ain32GetUnsignedDelta()

```
virtual const Error* Ain32GetUnsignedDelta (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input unsigned delta value as a 32-bit integer.

The unsigned delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.20 Ain32GetUpperLimit()

```
virtual const Error* Ain32GetUpperLimit (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog input upper limit value as a 32-bit integer.

The upper limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.21 Ain32Read()

```
virtual const Error* Ain32Read (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Read a 32-bit analog input.

Parameters

id	The analog input channel ID
value	The analog input value

Returns

A pointer to an error object, or NULL on success

5.68.3.22 Ain32SetLowerLimit()

```
virtual const Error* Ain32SetLowerLimit (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input lower limit value as a 32-bit integer.

The lower limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.23 Ain32SetNegativeDelta()

```
virtual const Error* Ain32SetNegativeDelta (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input negative delta value as a 32-bit integer.

The negative delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.24 Ain32SetOffset()

```
virtual const Error* Ain32SetOffset (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input offset value as a 32-bit integer.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.25 Ain32SetPositiveDelta()

```
virtual const Error* Ain32SetPositiveDelta (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input positive delta value as a 32-bit integer.

The positive delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.26 Ain32SetScaling()

```
virtual const Error* Ain32SetScaling (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input scaling factor as a 32-bit integer.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.27 Ain32SetUnsignedDelta()

```
virtual const Error* Ain32SetUnsignedDelta (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input unsigned delta value as a 32-bit integer.

The unsigned delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.28 Ain32SetUpperLimit()

```
virtual const Error* Ain32SetUpperLimit (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog input upper limit value as a 32-bit integer.

The upper limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.29 Ain8GetCt()

```
virtual const Error* Ain8GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 8-bit analog inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.30 Ain8Read()

```
virtual const Error* Ain8Read (
            uint8 id,
            int8 & value )  [inline], [virtual]
```

Read an 8-bit analog input.

Parameters

id	The analog input channel ID
value	The analog input value

Returns

A pointer to an error object, or NULL on success

5.68.3.31 AinFltGetCt()

```
virtual const Error* AinFltGetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of floating point analog inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.32 AinFltGetLowerLimit()

```
virtual const Error* AinFltGetLowerLimit (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input lower limit value as a floating point value.

The lower limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.33 AinFltGetNegativeDelta()

```
virtual const Error* AinFltGetNegativeDelta (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input negative delta value as a floating point value.

The negative delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.34 AinFltGetOffset()

```
virtual const Error* AinFltGetOffset (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input offset value as a floating point value.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.35 AinFltGetPositiveDelta()

```
virtual const Error* AinFltGetPositiveDelta (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input positive delta value as a floating point value.

The positive delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.36 AinFltGetScaling()

```
virtual const Error* AinFltGetScaling (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input scaling factor as a floating point value.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.37 AinFltGetUnsignedDelta()

```
virtual const Error* AinFltGetUnsignedDelta (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input unsigned delta value as a floating point value.

The unsigned delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.38 AinFltGetUpperLimit()

```
virtual const Error* AinFltGetUpperLimit (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog input upper limit value as a floating point value.

The upper limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.39 AinFltRead()

```
virtual const Error* AinFltRead (
            uint8 id,
            float & value )  [inline], [virtual]
```

Read a floating point analog input.

Parameters

id	The analog input channel ID
value	The analog input value

Returns

A pointer to an error object, or NULL on success

5.68.3.40 AinFltSetLowerLimit()

```
virtual const Error* AinFltSetLowerLimit (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input lower limit value as a floating point value.

The lower limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.41 AinFltSetNegativeDelta()

```
virtual const Error* AinFltSetNegativeDelta (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input negative delta value as a floating point value.

The negative delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.42 AinFltSetOffset()

```
virtual const Error* AinFltSetOffset (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input offset value as a floating point value.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.43 AinFltSetPositiveDelta()

```
virtual const Error* AinFltSetPositiveDelta (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input positive delta value as a floating point value.

The positive delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.44 AinFltSetScaling()

```
virtual const Error* AinFltSetScaling (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input scaling factor as a floating point value.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.45 AinFltSetUnsignedDelta()

```
virtual const Error* AinFltSetUnsignedDelta (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input unsigned delta value as a floating point value.

The unsigned delta defines the amount of change at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.46 AinFltSetUpperLimit()

```
virtual const Error* AinFltSetUpperLimit (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog input upper limit value as a floating point value.

The upper limit defines the value at which an interrupt will be generated if it is enabled.

Parameters

id	The analog input channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.47 AinGetIntEna()

```
virtual const Error* AinGetIntEna (
            bool & value )  [inline], [virtual]
```

Get the current setting of the global interrupt enable for analog inputs.

A return value of true indicates that interrupts are enabled, false disabled.

Parameters

value	The current interrupt enable setting is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.48 AinGetIntSource()

```
virtual const Error* AinGetIntSource (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the analog input interrupt source.

This variable may be used to determine which analog input has produced an interrupt. There are eight banks of interrupt source registers, each of which covers 32 analog inputs in it's 32 bits. Bank 0 identifies analog inputs 0 to 31, Bank 1 identifies analog inputs 32 to 63, etc. The bit associated with the analog input generating the latest interrupt will be set in the value returned by this read. Reading this variable causes all it's bit to be automatically reset.

Parameters

id	The bank number to read (0 to 7)
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.49 AinGetTrigType()

```
virtual const Error* AinGetTrigType (
            uint8 id,
            IO_AIN_TRIG_TYPE & value )  [inline], [virtual]
```

Get the analog input trigger type associated with the input channel.

Parameters

id	The analog input channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.50 AinSetIntEna()

```
virtual const Error* AinSetIntEna (
            bool value )  [inline], [virtual]
```

Set the current setting of the global interrupt enable for analog inputs.

Setting this parameter to true enables interrupts, false disables.

Parameters

value	The interrupt enable setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.51 AinSetTrigType()

```
virtual const Error* AinSetTrigType (
            uint8 id,
            IO_AIN_TRIG_TYPE value )  [inline], [virtual]
```

Set the analog input trigger type associated with the input channel.

Parameters

id	The analog input channel ID
value	The value to set

Returns

A pointer to an error object, or NULL on success

5.68.3.52 Aout16GetCt()

```
virtual const Error* Aout16GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 16-bit analog outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.53 Aout16GetErrValue()

```
virtual const Error* Aout16GetErrValue (
            uint8 id,
            int16 & value )  [inline], [virtual]
```

Get the analog output error value as a 16-bit integer.

The error value is the value that the analog output will assume on device error, if it's error mode is set to true.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.54 Aout16SetErrValue()

```
virtual const Error* Aout16SetErrValue (
            uint8 id,
            int16 value )  [inline], [virtual]
```

Set the analog output error value as a 32-bit integer.

The error value is the value that the analog output will assume on device error, if it's error mode is set to true.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.55 Aout16Write()

```
const Error * Aout16Write (
            uint8 id,
            int16 value,
            bool viaSDO = false )  [virtual]
```

Write to a 16-bit analog output.

Since 16-bit outputs are mapped to the default PDOs of the I/O module, these outputs may be written using either PDOs or SDOs.

Parameters

id	The analog input channel ID
value	The value to write.
viaSDO	If true, the outputs will be written using SDO messages. If false (default), then a PDO will be used if possible.

Returns

A pointer to an error object, or NULL on success

5.68.3.56 Aout32GetCt()

```
virtual const Error* Aout32GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 32-bit analog outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.57 Aout32GetErrValue()

```
virtual const Error* Aout32GetErrValue (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog output error value as a 32-bit integer.

The error value is the value that the analog output will assume on device error, if it's error mode is set to true.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.58 Aout32GetOffset()

```
virtual const Error* Aout32GetOffset (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog output offset value as a 32-bit integer.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.59 Aout32GetScaling()

```
virtual const Error* Aout32GetScaling (
            uint8 id,
            int32 & value )  [inline], [virtual]
```

Get the analog output scaling factor as a 32-bit integer.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.60 Aout32SetErrValue()

```
virtual const Error* Aout32SetErrValue (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog output error value as a 32-bit integer.

The error value is the value that the analog output will assume on device error, if it's error mode is set to true.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.61 Aout32SetOffset()

```
virtual const Error* Aout32SetOffset (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog output offset value as a 32-bit integer.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.62 Aout32SetScaling()

```
virtual const Error* Aout32SetScaling (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Set the analog output scaling factor as a 32-bit integer.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.63 Aout32Write()

```
virtual const Error* Aout32Write (
            uint8 id,
            int32 value )  [inline], [virtual]
```

Write to a 32-bit analog output.

Parameters

id	The analog input channel ID
value	The value to write.

Returns

A pointer to an error object, or NULL on success

5.68.3.64 Aout8GetCt()

```
virtual const Error* Aout8GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 8-bit analog outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.65 Aout8Write()

```
virtual const Error* Aout8Write (
            uint8 id,
            int8 value )  [inline], [virtual]
```

Write to an 8-bit analog output.

Parameters

id	The analog input channel ID
value	The value to write.

Returns

A pointer to an error object, or NULL on success

5.68.3.66 AoutFltGetCt()

```
virtual const Error* AoutFltGetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of floating point analog outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.67 AoutFltGetErrValue()

```
virtual const Error* AoutFltGetErrValue (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog output error value as a floating point value.

The error value is the value that the analog output will assume on device error, if it's error mode is set to true.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.68 AoutFltGetOffset()

```
virtual const Error* AoutFltGetOffset (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog output offset value as a floating point value.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.69 AoutFltGetScaling()

```
virtual const Error* AoutFltGetScaling (
            uint8 id,
            float & value )  [inline], [virtual]
```

Get the analog output scaling factor as a floating point value.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.70 AoutFltSetErrValue()

```
virtual const Error* AoutFltSetErrValue (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog output error value as a floating point value.

The error value is the value that the analog output will assume on device error, if it's error mode is set to true.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.71 AoutFltSetOffset()

```
virtual const Error* AoutFltSetOffset (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog output offset value as a floating point value.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.72 AoutFltSetScaling()

```
virtual const Error* AoutFltSetScaling (
            uint8 id,
            float value )  [inline], [virtual]
```

Set the analog output scaling factor as a floating point value.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.73 AoutFltWrite()

```
virtual const Error* AoutFltWrite (
            uint8 id,
            float value )  [inline], [virtual]
```

Write to a floating point analog output.

Parameters

id	The analog input channel ID
value	The value to write.

Returns

A pointer to an error object, or NULL on success

5.68.3.74 AoutGetErrMode()

```
virtual const Error* AoutGetErrMode (
            uint8 id,
            bool & value )  [inline], [virtual]
```

Get the analog output error mode.

If the error mode is true, then the analog output will change it's value to the programmed error value in the case of a device failure. If false, a device failure will not cause a change in the analog output value.

Parameters

id	The analog output channel ID
value	The value is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.75 AoutSetErrMode()

```
virtual const Error* AoutSetErrMode (
            uint8 id,
            bool value )  [inline], [virtual]
```

Set the analog output error mode.

If the error mode is true, then the analog output will change it's value to the programmed error value in the case of a device failure. If false, a device failure will not cause a change in the analog output value.

Parameters

id	The analog output channel ID
value	The value to be set.

Returns

A pointer to an error object, or NULL on success

5.68.3.76 BitCount()

```
const Error* BitCount (
            uint16 base,
            uint16 & ct )  [inline], [protected]
```

Count the number of individual I/O pins available on the device.

Parameters

base	The base index for the object dictionary
ct	The count is returned here

Returns

A pointer to an error object, or NULL on success

5.68.3.77 BitDnld()

```
const Error* BitDnld (
            uint16 base,
            uint16 id,
            bool value )  [inline], [protected]
```

Download a setting for a single digital I/O pin.

The object dictionary index/sub-index is calculated from the pin ID and a passed base address.

Parameters

base	The object dictionary base index for this parameter
id	The I/O pin ID. Must range from 0 to 1024
value	The boolean value is passed here.

Returns

A pointer to an error object, or NULL on success

5.68.3.78 BitUpld()

```
const Error* BitUpld (
            uint16 base,
            uint16 id,
            bool & value )  [inline], [protected]
```

Upload a setting for a single digital I/O pin.

The object dictionary entry is calculated based on the pin ID and the base object dictionary ID passed.

Parameters

base	The object dictionary base index for this parameter
id	The I/O pin ID. Must range from 0 to 1024
value	The boolean value is returned here

Returns

A pointer to an error object, or NULL on success

5.68.3.79 Din16GetCt()

```
virtual const Error* Din16GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 16-bit groups of inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.80 Din16GetFilt()

```
virtual const Error* Din16GetFilt (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the current filter constant settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of inputs to read.
value	The current filter setting of the input lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.81 Din16GetMaskAny()

```
virtual const Error* Din16GetMaskAny (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the 'any transition' interrupt mask settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables interrupts on any change, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of inputs to read.
value	The current interrupt mask setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.82 Din16GetMaskHigh2Low()

```
virtual const Error* Din16GetMaskHigh2Low (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the 'high to low' interrupt mask settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables interrupts on a high to low transition, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of inputs to read.
value	The current interrupt mask setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.83 Din16GetMaskLow2High()

```
virtual const Error* Din16GetMaskLow2High (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the 'low to high' interrupt mask settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables interrupts on a low to high transition, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of inputs to read.
value	The current interrupt mask setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.84 Din16GetPol()

```
virtual const Error* Din16GetPol (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the current polarity settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of inputs to read.
value	The current polarity setting of the input lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.85 Din16Read()

```
const Error * Din16Read (
            uint8 id,
            uint16 & value,
            bool viaSDO = false )  [virtual]
```

Read a group of 16 digital inputs.

Parameters

id	Identifies which group of 16 inputs to read.
value	The value of the 16 input lines is returned here.
viaSDO	If true, read the inputs using SDO transfers. If false (default) use the most recently received PDO data if this input group is mapped to a transmit PDO and the PDO is active.

Returns

A pointer to an error object, or NULL on success

5.68.3.86 Din16SetFilt()

```
virtual const Error* Din16SetFilt (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the current filter constant setting for a group of 16 digital inputs.

For each input in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new filter setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.87 Din16SetMaskAny()

```
virtual const Error* Din16SetMaskAny (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the 'any transition' interrupt mask settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables interrupts on any transition, and a value of 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.88 Din16SetMaskHigh2Low()

```
virtual const Error* Din16SetMaskHigh2Low (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the 'high to low' interrupt mask settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables interrupts on high to low transitions, and a value of 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.89 Din16SetMaskLow2High()

```
virtual const Error* Din16SetMaskLow2High (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the 'low to high' interrupt mask settings for a group of 16 digital inputs.

For each input in the group, a value of 1 enables interrupts on low to high transitions, and a value of 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.90 Din16SetPol()

```
virtual const Error* Din16SetPol (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the current polarity setting for a group of 16 digital inputs.

For each input in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new polarity setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.91 Din32GetCt()

```
virtual const Error* Din32GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 32-bit groups of inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.92 Din32GetFilt()

```
virtual const Error* Din32GetFilt (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the current filter constant settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of inputs to read.
value	The current filter setting of the input lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.93 Din32GetMaskAny()

```
virtual const Error* Din32GetMaskAny (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the 'any transition' interrupt mask settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables interrupts on any change, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of inputs to read.
value	The current interrupt mask setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.94 Din32GetMaskHigh2Low()

```
virtual const Error* Din32GetMaskHigh2Low (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the 'high to low' interrupt mask settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables interrupts on a high to low transition, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of inputs to read.
value	The current interrupt mask setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.95 Din32GetMaskLow2High()

```
virtual const Error* Din32GetMaskLow2High (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the 'low to high' interrupt mask settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables interrupts on a low to high transition, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of inputs to read.
value	The current interrupt mask setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.96 Din32GetPol()

```
virtual const Error* Din32GetPol (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the current polarity settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of inputs to read.
value	The current polarity setting of the input lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.97 Din32Read()

```
const Error * Din32Read (
            uint8 id,
            uint32 & value,
            bool viaSDO = false )  [virtual]
```

Read a group of 32 digital inputs.

Parameters

id	Identifies which group of 32 inputs to read.
value	The value of the 32 input lines is returned here.
viaSDO	If true, read the inputs using SDO transfers. If false (default) use the most recently received PDO data if this input group is mapped to a transmit PDO and the PDO is active.

Returns

A pointer to an error object, or NULL on success

5.68.3.98 Din32SetFilt()

```
virtual const Error* Din32SetFilt (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the current filter constant setting for a group of 32 digital inputs.

For each input in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new filter setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.99 Din32SetMaskAny()

```
virtual const Error* Din32SetMaskAny (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the 'any transition' interrupt mask settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables interrupts on any transition, and a value of 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.100 Din32SetMaskHigh2Low()

```
virtual const Error* Din32SetMaskHigh2Low (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the 'high to low' interrupt mask settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables interrupts on high to low transitions, and a value of 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.101 Din32SetMaskLow2High()

```
virtual const Error* Din32SetMaskLow2High (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the 'low to high' interrupt mask settings for a group of 32 digital inputs.

For each input in the group, a value of 1 enables interrupts on low to high transitions, and a value of 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.102 Din32SetPol()

```
virtual const Error* Din32SetPol (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the current polarity setting for a group of 32 digital inputs.

For each input in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of inputs to effect.
value	The new polarity setting of the input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.103 Din8GetCt()

```
virtual const Error* Din8GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 8-bit groups of inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.104 Din8GetFilt()

```
virtual const Error* Din8GetFilt (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the current filter constant settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of 8 inputs to read.
value	The current filter setting of the 8 input lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.105 Din8GetMaskAny()

```
virtual const Error* Din8GetMaskAny (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the 'any transition' interrupt mask settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables interrupts on any change, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of 8 inputs to read.
value	The current interrupt mask setting of the 8 input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.106 Din8GetMaskHigh2Low()

```
virtual const Error* Din8GetMaskHigh2Low (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the 'high to low' interrupt mask settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables interrupts on a high to low transition, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of 8 inputs to read.
value	The current interrupt mask setting of the 8 input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.107 Din8GetMaskLow2High()

```
virtual const Error* Din8GetMaskLow2High (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the 'low to high' interrupt mask settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables interrupts on a low to high transition, and a value of 0 disables the interrupt.

Parameters

id	Identifies which group of 8 inputs to read.
value	The current interrupt mask setting of the 8 input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.108 Din8GetPol()

```
virtual const Error* Din8GetPol (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the current polarity settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of 8 inputs to read.
value	The current polarity setting of the 8 input lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.109 Din8Read()

```
const Error * Din8Read (
            uint8 id,
            uint8 & value,
            bool viaSDO = false )  [virtual]
```

Read a group of 8 digital inputs.

Parameters

id	Identifies which group of 8 inputs to read.
value	The value of the 8 input lines is returned here.
viaSDO	If true, read the inputs using SDO transfers. If false (default) use the most recently received PDO data if this input group is mapped to a transmit PDO and the PDO is active.

Returns

A pointer to an error object, or NULL on success

5.68.3.110 Din8SetFilt()

```
virtual const Error* Din8SetFilt (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the current filter constant setting for a group of 8 digital inputs.

For each input in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of 8 inputs to effect.
value	The new filter setting of the 8 input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.111 Din8SetMaskAny()

```
virtual const Error* Din8SetMaskAny (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the 'any transition' interrupt mask settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables interrupts on any transition, and a value of 0 disables.

Parameters

id	Identifies which group of 8 inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.112 Din8SetMaskHigh2Low()

```
virtual const Error* Din8SetMaskHigh2Low (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the 'high to low' interrupt mask settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables interrupts on high to low transitions, and a value of 0 disables.

Parameters

id	Identifies which group of 8 inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.113 Din8SetMaskLow2High()

```
virtual const Error* Din8SetMaskLow2High (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the 'low to high' interrupt mask settings for a group of 8 digital inputs.

For each input in the group, a value of 1 enables interrupts on low to high transitions, and a value of 0 disables.

Parameters

id	Identifies which group of 8 inputs to effect.
value	The new interrupt mask value.

Returns

A pointer to an error object, or NULL on success

5.68.3.114 Din8SetPol()

```
virtual const Error* Din8SetPol (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the current polarity setting for a group of 8 digital inputs.

For each input in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of 8 inputs to effect.
value	The new polarity setting of the 8 input lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.115 DinGetCt()

```
virtual const Error* DinGetCt (
            uint16 & ct )  [inline], [virtual]
```

Return the number of individual inputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.116 DinGetFilt()

```
virtual const Error* DinGetFilt (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the current filter constant setting for a digital input.

The filter constant is enabled if true, disabled if false.

Parameters

id	Identifies the digital input.
value	The current filter setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.117 DinGetIntEna()

```
virtual const Error* DinGetIntEna (
            bool & value )  [inline], [virtual]
```

Get the current setting of the global interrupt enable for digital inputs.

A return value of true indicates that interrupts are enabled, false disabled.

Parameters

value	The current interrupt enable setting is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.118 DinGetMaskAny()

```
virtual const Error* DinGetMaskAny (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the 'any transition' interrupt mask settings for a digital input.

If true, any transition on the input will generate an interrupt.

Parameters

id	Identifies the digital input.
value	The current interrupt mask setting

Returns

A pointer to an error object, or NULL on success

5.68.3.119 DinGetMaskHigh2Low()

```
virtual const Error* DinGetMaskHigh2Low (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the 'high to low' interrupt mask settings for a digital input.

If true, a high to low transition on the input will generate an interrupt.

Parameters

id	Identifies the digital input.
value	The current interrupt mask setting

Returns

A pointer to an error object, or NULL on success

5.68.3.120 DinGetMaskLow2High()

```
virtual const Error* DinGetMaskLow2High (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the 'low to high' interrupt mask settings for a digital input.

If true, a low to high transition on the input will generate an interrupt.

Parameters

id	Identifies the digital input.
value	The current interrupt mask setting

Returns

A pointer to an error object, or NULL on success

5.68.3.121 DinGetPol()

```
virtual const Error* DinGetPol (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the current polarity settings for a digital input.

Polarity inversion is enabled if true, disabled if false.

Parameters

id	Identifies the digital input.
value	The current polarity setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.122 DinRead()

```
const Error * DinRead (
            uint16 id,
            bool & value,
            bool viaSDO = false )  [virtual]
```

Read a single digital input.

Parameters

id	Identifies the digital input to read.
value	The value of the input.
viaSDO	If true, an SDO will be used to read the input pin. If false (default), the latest value returned via PDO will be returned, if available.

Returns

A pointer to an error object, or NULL on success

5.68.3.123 DinSetFilt()

```
virtual const Error* DinSetFilt (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the current filter constant setting for a digital input.

The filter constant is enabled if true, disabled if false.

Parameters

id	Identifies the digital input.
value	The new filter setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.124 DinSetIntEna()

```
virtual const Error* DinSetIntEna (
            bool value )  [inline], [virtual]
```

Set the current setting of the global interrupt enable for digital inputs.

Setting this parameter to true enables interrupts, false disables.

Parameters

value	The interrupt enable setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.125 DinSetMaskAny()

```
virtual const Error* DinSetMaskAny (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the 'any transition' interrupt mask settings for a digital input.

If true, any transition on the input will generate an interrupt.

Parameters

id	Identifies the digital input.
value	The new interrupt mask setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.126 DinSetMaskHigh2Low()

```
virtual const Error* DinSetMaskHigh2Low (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the 'high to low' interrupt mask settings for a digital input.

If true, a high to low transition on the input will generate an interrupt.

Parameters

id	Identifies the digital input.
value	The new interrupt mask setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.127 DinSetMaskLow2High()

```
virtual const Error* DinSetMaskLow2High (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the 'low to high' interrupt mask settings for a digital input.

If true, a low to high transition on the input will generate an interrupt.

Parameters

id	Identifies the digital input.
value	The new interrupt mask setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.128 DinSetPol()

```
virtual const Error* DinSetPol (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the current polarity setting for a digital input.

Polarity inversion is enabled if true, disabled if false.

Parameters

id	Identifies the digital input.
value	The new polarity setting.

Returns

A pointer to an error object, or NULL on success

5.68.3.129 Dout16GetCt()

```
virtual const Error* Dout16GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 16-bit groups of outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.130 Dout16GetErrMode()

```
virtual const Error* Dout16GetErrMode (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the current error mode settings for a group of 16 digital outputs.

For each output in the group, a value of 1 will cause the output to take it's programmed error value on a device failure. Setting the mode to 0 will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which group of outputs to read.
value	The current error mode setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.131 Dout16GetErrValue()

```
virtual const Error* Dout16GetErrValue (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the current error value settings for a group of 16 digital outputs.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to 1. Those with error mode set to zero will not be changed by a device failure.

Parameters

id	Identifies which group of outputs to read.
value	The current error value setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.132 Dout16GetFilt()

```
virtual const Error* Dout16GetFilt (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the current filter constant settings for a group of 16 digital outputs.

For each output in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of outputs to read.
value	The current filter setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.133 Dout16GetPol()

```
virtual const Error* Dout16GetPol (
            uint8 id,
            uint16 & value )  [inline], [virtual]
```

Get the current polarity settings for a group of 16 digital outputs.

For each output in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of outputs to read.
value	The current polarity setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.134 Dout16Read()

```
const Error* Dout16Read (
            uint8 id,
            uint16 & value )  [inline]
```

Read back the last value written to this bank of 16 digital outputs.

Parameters

id	Identifies which group of outputs to read.
value	The current state of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.135 Dout16SetErrMode()

```
virtual const Error* Dout16SetErrMode (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the current error mode settings for a group of 16 digital outputs.

For each output in the group, a value of 1 will cause the output to take it's programmed error value on a device failure. Setting the mode to 0 will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which group of outputs to effect.
value	The new error mode setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.136 Dout16SetErrValue()

```
virtual const Error* Dout16SetErrValue (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the current error value settings for a group of 16 digital outputs.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to 1. Those with error mode set to zero will not be changed by a device failure.

Parameters

id	Identifies which group of outputs to effect.
value	The new error value setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.137 Dout16SetFilt()

```
virtual const Error* Dout16SetFilt (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the current filter constant setting for a group of 16 digital outputs.

For each output in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of outputs to effect.
value	The new filter setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.138 Dout16SetPol()

```
virtual const Error* Dout16SetPol (
            uint8 id,
            uint16 value )  [inline], [virtual]
```

Set the current polarity setting for a group of 16 digital outputs.

For each output in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of outputs to effect.
value	The new polarity setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.139 Dout16Write()

```
const Error * Dout16Write (
            uint8 id,
            uint16 value,
            bool viaSDO = false )  [virtual]
```

Write a group of 16 digital outputs.

The outputs may be written either by SDO or by PDO. The PDO method is faster since it only requires a single message to be sent. SDO transfers additionally require a response from the module.

If a PDO transfer is requested, but is not possible because the module is not in an operational state, or because the output block isn't mapped to the PDO, then an SDO transfer will be used.

Parameters

id	Identifies which group of outputs to write.
value	The new value of the output lines.
viaSDO	If true, the outputs will be written using SDO messages. If false (default), then a PDO will be used if possible.

Returns

A pointer to an error object, or NULL on success

5.68.3.140 Dout32GetCt()

```
virtual const Error* Dout32GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 32-bit groups of outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.141 Dout32GetErrMode()

```
virtual const Error* Dout32GetErrMode (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the current error mode settings for a group of 32 digital outputs.

For each output in the group, a value of 1 will cause the output to take it's programmed error value on a device failure. Setting the mode to 0 will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which group of outputs to read.
value	The current error mode setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.142 Dout32GetErrValue()

```
virtual const Error* Dout32GetErrValue (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the current error value settings for a group of 32 digital outputs.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to 1. Those with error mode set to zero will not be changed by a device failure.

Parameters

id	Identifies which group of outputs to read.
value	The current error value setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.143 Dout32GetFilt()

```
virtual const Error* Dout32GetFilt (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the current filter constant settings for a group of 32 digital outputs.

For each output in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of outputs to read.
value	The current filter setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.144 Dout32GetPol()

```
virtual const Error* Dout32GetPol (
            uint8 id,
            uint32 & value )  [inline], [virtual]
```

Get the current polarity settings for a group of 32 digital outputs.

For each output in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of outputs to read.
value	The current polarity setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.145 Dout32Read()

```
const Error* Dout32Read (
            uint8 id,
            uint32 & value )  [inline]
```

Read back the last value written to this bank of 32 digital outputs.

Parameters

id	Identifies which group of outputs to read.
value	The current state of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.146 Dout32SetErrMode()

```
virtual const Error* Dout32SetErrMode (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the current error mode settings for a group of 32 digital outputs.

For each output in the group, a value of 1 will cause the output to take it's programmed error value on a device failure. Setting the mode to 0 will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which group of outputs to effect.
value	The new error mode setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.147 Dout32SetErrValue()

```
virtual const Error* Dout32SetErrValue (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the current error value settings for a group of 32 digital outputs.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to 1. Those with error mode set to zero will not be changed by a device failure.

Parameters

id	Identifies which group of outputs to effect.
value	The new error value setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.148 Dout32SetFilt()

```
virtual const Error* Dout32SetFilt (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the current filter constant setting for a group of 32 digital outputs.

For each output in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of outputs to effect.
value	The new filter setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.149 Dout32SetPol()

```
virtual const Error* Dout32SetPol (
            uint8 id,
            uint32 value )  [inline], [virtual]
```

Set the current polarity setting for a group of 32 digital outputs.

For each output in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of outputs to effect.
value	The new polarity setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.150 Dout32Write()

```
const Error * Dout32Write (
            uint8 id,
            uint32 value,
            bool viaSDO = false )  [virtual]
```

Write a group of 32 digital outputs.

The outputs may be written either by SDO or by PDO. The PDO method is faster since it only requires a single message to be sent. SDO transfers additionally require a response from the module.

If a PDO transfer is requested, but is not possible because the module is not in an operational state, or because the output block isn't mapped to the PDO, then an SDO transfer will be used.

Parameters

id	Identifies which group of outputs to write.
value	The new value of the output lines.
viaSDO	If true, the outputs will be written using SDO messages. If false (default), then a PDO will be used if possible.

Returns

A pointer to an error object, or NULL on success

5.68.3.151 Dout8GetCt()

```
virtual const Error* Dout8GetCt (
            uint8 & ct )  [inline], [virtual]
```

Return the number of 8-bit groups of outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.152 Dout8GetErrMode()

```
virtual const Error* Dout8GetErrMode (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the current error mode settings for a group of 8 digital outputs.

For each output in the group, a value of 1 will cause the output to take it's programmed error value on a device failure. Setting the mode to 0 will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which group of outputs to read.
value	The current error mode setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.153 Dout8GetErrValue()

```
virtual const Error* Dout8GetErrValue (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the current error value settings for a group of 8 digital outputs.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to 1. Those with error mode set to zero will not be changed by a device failure.

Parameters

id	Identifies which group of outputs to read.
value	The current error value setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.154 Dout8GetFilt()

```
virtual const Error* Dout8GetFilt (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the current filter constant settings for a group of 8 digital outputs.

For each output in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of outputs to read.
value	The current filter setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.155 Dout8GetPol()

```
virtual const Error* Dout8GetPol (
            uint8 id,
            uint8 & value )  [inline], [virtual]
```

Get the current polarity settings for a group of 8 digital outputs.

For each output in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of outputs to read.
value	The current polarity setting of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.156 Dout8Read()

```
const Error* Dout8Read (
            uint8 id,
            uint8 & value )  [inline]
```

Read back the last value written to this bank of 8 digital outputs.

Parameters

id	Identifies which group of outputs to read.
value	The current state of the output lines is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.157 Dout8SetErrMode()

```
virtual const Error* Dout8SetErrMode (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the current error mode settings for a group of 8 digital outputs.

For each output in the group, a value of 1 will cause the output to take it's programmed error value on a device failure. Setting the mode to 0 will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which group of outputs to effect.
value	The new error mode setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.158 Dout8SetErrValue()

```
virtual const Error* Dout8SetErrValue (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the current error value settings for a group of 8 digital outputs.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to 1. Those with error mode set to zero will not be changed by a device failure.

Parameters

id	Identifies which group of outputs to effect.
value	The new error value setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.159 Dout8SetFilt()

```
virtual const Error* Dout8SetFilt (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the current filter constant setting for a group of 8 digital outputs.

For each output in the group, a value of 1 enables the filter, 0 disables.

Parameters

id	Identifies which group of outputs to effect.
value	The new filter setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.160 Dout8SetPol()

```
virtual const Error* Dout8SetPol (
            uint8 id,
            uint8 value )  [inline], [virtual]
```

Set the current polarity setting for a group of 8 digital outputs.

For each output in the group, a value of 1 enables inversion and 0 disables.

Parameters

id	Identifies which group of outputs to effect.
value	The new polarity setting of the output lines.

Returns

A pointer to an error object, or NULL on success

5.68.3.161 Dout8Write()

```
const Error * Dout8Write (
            uint8 id,
            uint8 value,
            bool viaSDO = false )
```

Write a group of 8 digital outputs.

The outputs may be written either by SDO or by PDO. The PDO method is faster since it only requires a single message to be sent. SDO transfers additionally require a response from the module.

If a PDO transfer is requested, but is not possible because the module is not in an operational state, or because the output block isn't mapped to the PDO, then an SDO transfer will be used.

Parameters

id	Identifies which group of outputs to write.
value	The new value of the output lines.
viaSDO	If true, the outputs will be written using SDO messages. If false (default), then a PDO will be used if possible.

Returns

A pointer to an error object, or NULL on success

5.68.3.162 DoutGetCt()

```
virtual const Error* DoutGetCt (
            uint16 & ct )  [inline], [virtual]
```

Return the number of individual outputs available on this device.

Parameters

ct	The count is returned here. Zero is returned on error.

Returns

A pointer to an error object, or NULL on success

5.68.3.163 DoutGetErrMode()

```
virtual const Error* DoutGetErrMode (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the current error mode setting for an individual digital output.

A value of true will cause the output to take it's programmed error value on a device failure. Setting the mode to false will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies the output to read.
value	The current error mode setting of the output line is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.164 DoutGetErrValue()

```
virtual const Error* DoutGetErrValue (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the current error value setting for an individual digital output.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to true. Those with error mode set to false will not be changed by a device failure.

Parameters

id	Identifies the output to read.
value	The current error value setting of the output line is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.165 DoutGetFilt()

```
virtual const Error* DoutGetFilt (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the current filter constant setting for an individual digital output.

A value of true enables the filter, false disables.

Parameters

id	Identifies the output to read.
value	The current filter setting of the output line is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.166 DoutGetPol()

```
virtual const Error* DoutGetPol (
            uint16 id,
            bool & value )  [inline], [virtual]
```

Get the current polarity setting for an individual digital output.

A value of true enables inversion and false disables.

Parameters

id	Identifies the output to read.
value	The current polarity setting of the output line is returned here.

Returns

A pointer to an error object, or NULL on success

5.68.3.167 DoutSetErrMode()

```
virtual const Error* DoutSetErrMode (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the current error mode setting for an individual digital output.

A value of true will cause the output to take it's programmed error value on a device failure. Setting the mode to false will cause the output to hold it's programmed value on failure.

Parameters

id	Identifies which digital output to effect.
value	The new error mode setting of the output line.

Returns

A pointer to an error object, or NULL on success

5.68.3.168 DoutSetErrValue()

```
virtual const Error* DoutSetErrValue (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the current error value setting for an individual digital output.

Error values define the state of the output if a device failure occurs. The error value will only be set for those output pins which have an error mode set to true. Those with error mode set to false will not be changed by a device failure.

Parameters

id	Identifies which digital output to effect.
value	The new error value setting of the output line.

Returns

A pointer to an error object, or NULL on success

5.68.3.169 DoutSetFilt()

```
virtual const Error* DoutSetFilt (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the current filter constant setting for an individual digital output.

A value of true enables the filter, false disables.

Parameters

id	Identifies which digital output to effect.
value	The new filter setting of the output line.

Returns

A pointer to an error object, or NULL on success

5.68.3.170 DoutSetPol()

```
virtual const Error* DoutSetPol (
            uint16 id,
            bool value )  [inline], [virtual]
```

Set the current polarity setting for an individual digital output.

A value of true enables inversion and false disables.

Parameters

id	Identifies which digital output to effect.
value	The new polarity setting of the output line.

Returns

A pointer to an error object, or NULL on success

5.68.3.171 DoutWrite()

```
const Error * DoutWrite (
            uint16 id,
```

```
            bool value,
            bool viaSDO = false )  [virtual]
```

Write an individual digital output.

The output may be written either by SDO or by PDO. The PDO method is faster since it only requires a single message to be sent. SDO transfers additionally require a response from the module.

If a PDO transfer is requested, but is not possible because the module is not in an operational state, or because the output isn't mapped to the PDO, then an SDO transfer will be used.

Parameters

id	Identifies which output to write.
value	The new value of the output line.
viaSDO	If true, the outputs will be written using SDO messages. If false (default), then a PDO will be used if possible.

Returns

A pointer to an error object, or NULL on success

5.68.3.172 Init() [1/2]

```
const Error * Init (
            Network & net,
            int16 nodeID )  [virtual]
```

Initialize an I/O module using default settings.

This function associates the object with the CANopen network it will be used on.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.

Returns

A pointer to an error object, or NULL on success

Reimplemented from Node.

Reimplemented in CopleyIO.

5.68.3.173 Init() [2/2]

```
const Error * Init (
            Network & net,
            int16 nodeID,
            IOModuleSettings & settings )  [virtual]
```

Initialize an I/O module using custom settings.

This function associates the object with the CANopen network it will be used on.

Parameters

net	The Network object that this module is associated with.
nodeID	The node ID of the module on the network.
settings	The settings to use when configuring the module

Returns

A pointer to an error object, or NULL on success

Reimplemented in CopleyIO.

5.68.3.174 PostIOEvent()

```
virtual void PostIOEvent (
            IOMODULE_EVENTS event )  [inline], [protected], [virtual]
```

Post an event condition to the I/O module's event map.

This method is used internally by the various standard transmit PDOs when a new PDO message is received.

Parameters

event	The event bit(s) to post

5.68.3.175 WaitIOEvent() [1/2]

```
const Error * WaitIOEvent (
            IOMODULE_EVENTS event,
            Timeout timeout = -1 )  [virtual]
```

Wait on an event associated with this I/O module.

The standard events are used to indicate that a new transmit PDO has been received. A thread may wait on such an event by calling this function.

Parameters

event	The event(s) to wait on. Multiple events may be ORed together and in this case this function will return when any of them occur.
timeout	The timeout for the wait (milliseconds). Negative values indicate that no timeout should be used (wait forever). The default value is -1.

Returns

A pointer to an error object, or NULL on success

5.68.3.176 WaitIOEvent() [2/2]

```
const Error * WaitIOEvent (
            Event & e,
            Timeout timeout,
            IOMODULE_EVENTS & match )  [virtual]
```

Wait for an event associated with this I/O module.

This function can be used to wait on any generic event associated with the I/O module.

Parameters

e	The event to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever.
match	Returns the matching event condition.

Returns

A pointer to an error object, or NULL on success.

The documentation for this class was generated from the following files:

- CML_IO.h
- IOmodule.cpp

5.69 IOModuleSettings Struct Reference

Standard CANopen I/O module settings.

Public Attributes

- uint16 heartbeatPeriod

  The CANopen heartbeat protocol is one of two standard methods used to constantly watch for network or device problems.
- uint16 heartbeatTimeout

  Additional time to wait before generating a heartbeat error (milliseconds) If the heartbeat protocol is used, then this value, combined with the heartbeatTime will determine how long the network master waits for the node's heartbeat message before it generates a heartbeat error.
- uint16 guardTime

  Node guarding guard time (milliseconds)
- uint8 lifeFactor

  Node guarding life time factor.
- bool useStandardDinPDO

  Use the standard digital input PDO object.
- bool useStandardDoutPDO

  Use the standard digital output PDO object.
- bool useStandardAinPDO

  Use the standard analog input PDO objects.
- bool useStandardAoutPDO

  Use the standard analog output PDO objects.

5.69.1 Detailed Description

Standard CANopen I/O module settings.

This structure may be passed to an I/O module object during initialization. It allows custom settings to be assigned to the module.

5.69.2 Member Data Documentation

5.69.2.1 guardTime

```
uint16 guardTime
```

Node guarding guard time (milliseconds)

The CANopen node guarding protocol is a second method (the first being the heartbeat protocol) for devices on the network to watch for network problems. In this protocol, the master controller sends a request message out to the slave device at a specified interval. The slave device responds to this request with a message indicating it's state.

The main difference between this protocol and the heartbeat protocol is that both the slave node and the master are able to recognize network errors. With the heartbeat protocol only the network master is able to identify network problems.

Note that only one of these two protocols can be active in a node device at any time. If the heartbeat period is non-zero, then the heartbeat protocol will be used.

This parameter gives the node guarding period for use with this node. This is the period between node guarding request messages sent by the master controller.

Note that both this parameter, and the life time factor must be non-zero for node guarding to be used.

Default 0 (ms)

5.69.2.2 heartbeatPeriod

```
uint16 heartbeatPeriod
```

The CANopen heartbeat protocol is one of two standard methods used to constantly watch for network or device problems.

When the heartbeat protocol is used, each device on the CANopen network transmits a 'heartbeat' message at a specified interval. The network master watches for these messages, and is able to detect a device error if it's heartbeat message is not received within the expected time.

This parameter configures the heartbeat period (milliseconds) that will be used by this module to transmit it's heartbeat message.

If this parameter is set to zero, then the heartbeat protocol is disabled on this node.

Default: zero (not used)

5.69.2.3 heartbeatTimeout

```
uint16 heartbeatTimeout
```

Additional time to wait before generating a heartbeat error (milliseconds) If the heartbeat protocol is used, then this value, combined with the heartbeatTime will determine how long the network master waits for the node's heartbeat message before it generates a heartbeat error.

Note that setting this to zero does not disable the heartbeat protocol. set the heartbeatPeriod value to zero to disable heartbeat.

Default 100 (ms)

5.69.2.4 lifeFactor

```
uint8 lifeFactor
```

Node guarding life time factor.

When the node guarding protocol is used, this parameter is used by the slave device to determine how long to wait for a node guarding request from the master controller before signaling an error condition.

The life time factor is treated as a multiple of the guard time.

If this parameter and the node guard time are both non-zero, and the heartbeatTime is zero, then node guarding will be setup for the amplifier.

Default 3 (multiples of the guard time)

5.69.2.5 useStandardAinPDO

```
bool useStandardAinPDO
```

Use the standard analog input PDO objects.

If true (default) then up to three standard PDO objects will be configured to read the first 12 16-bit analog input pins when they generate events. If false, then these PDOs will not be configured.

5.69.2.6 useStandardAoutPDO

```
bool useStandardAoutPDO
```

Use the standard analog output PDO objects.

If true (default) then up to three standard PDO objects will be configured to transmit the analog output data for up to 12 16-bit analog outputs. If false, then these PDOs will not be configured.

5.69.2.7 useStandardDinPDO

```
bool useStandardDinPDO
```

Use the standard digital input PDO object.

If true (default) then a standard PDO object will be configured to read up to 64 digital inputs any time one of them changes. If false, then this PDO will not be configured.

5.69.2.8 useStandardDoutPDO

```
bool useStandardDoutPDO
```

Use the standard digital output PDO object.

If true (default) then a standard PDO object will be configured to transmit updated settings for the first 64 digital output pins when any of them are changed. If false, no such PDO will be configured.

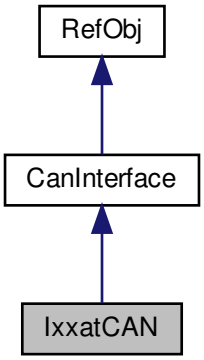
The documentation for this struct was generated from the following file:

- CML_IO.h

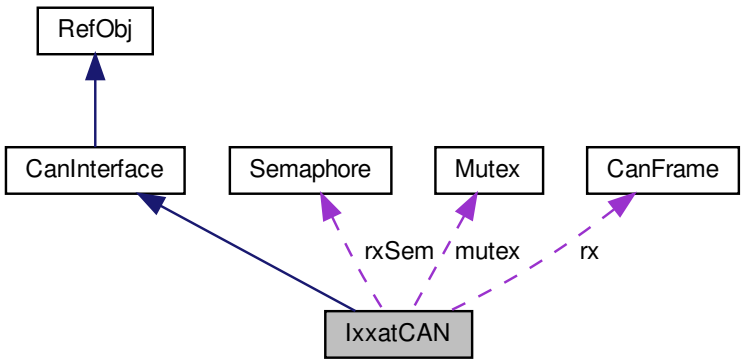
5.70 IxxatCAN Class Reference

Ixxat specific CAN interface.

Inheritance diagram for IxxatCAN:



Collaboration diagram for IxxatCAN:



Public Member Functions

- IxxatCAN (void)

  Construct a new Ixxat CAN interface object.

- IxxatCAN (const char *port)

  Construct a new Ixxat CAN interface object for the specified port.
- virtual ~IxxatCAN (void)

  Destructor for Ixxat card.
- const Error * Open (void)

  Open the Ixxat CAN card.
- const Error * Close (void)

  Close the CAN interface.
- const Error * SetBaud (int32 baud)

  Set the CAN interface baud rate.
- void rxInt (int16 ct, void *frame)

  Receive interrupt handler.

Protected Member Functions

- const Error * RecvFrame (CanFrame &frame, Timeout timeout)

  Receive the next CAN frame.
- const Error * XmitFrame (CanFrame &frame, Timeout timeout)

  Write a CAN frame to the CAN network.
- const Error * ConvertError (int err)

  Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Protected Attributes

- int open

  tracks the state of the interface as open or closed.
- uint8 channel

  Which CAN channel to use (on multi-channel boards).
- int32 baud

  Holds a copy of the last baud rate set.
- int handle

  File handle used to access the CAN channel.

Additional Inherited Members

5.70.1 Detailed Description

Ixxat specific CAN interface.

This class extends the generic CanInterface class into a working interface for the Ixxat can device driver.

5.70.2 Constructor & Destructor Documentation

5.70.2.1 IxxatCAN() [1/2]

```
IxxatCAN (
            void  )
```

Construct a new Ixxat CAN interface object.

This simply sets the default baud rate and marks the card as not open.

5.70.2.2 IxxatCAN() [2/2]

```
IxxatCAN (
            const char * port )
```

Construct a new Ixxat CAN interface object for the specified port.

The port name should be of the form CANx or IXXATx where x is the port number. The port numbers start at 0, so the first port would be identified by the port name CAN0.

Parameters

port	The port name string identifying the CAN device.

5.70.2.3 ~IxxatCAN()

```
~IxxatCAN (
            void  )  [virtual]
```

Destructor for Ixxat card.

Closes the interface and unloads the library.

5.70.3 Member Function Documentation

5.70.3.1 Close()

```
const Error * Close (
            void  )  [virtual]
```

Close the CAN interface.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.70.3.2 ConvertError()

```
const Error * ConvertError (
            int err )  [protected]
```

Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Parameters

err	The Vector style status code

Returns

A pointer to an error object, or NULL if no error is indicated

5.70.3.3 Open()

```
const Error * Open (
            void  )  [virtual]
```

Open the Ixxat CAN card.

The card should have been identified by setting it's name either in the constructor, or by using the method CanInterface::SetName.

If no port name was set, then the default Ixxat card will be used.

If the port name is set to "select", then a dialog box will be shown allowing the card to be selected from any installed Ixxat cards.

Otherwise, the port name should be of the form "CANx" where x is the Ixxat hardware key number (i.e. CAN1 for hardware key 1).

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.70.3.4 RecvFrame()

```
const Error * RecvFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Receive the next CAN frame.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.70.3.5 rxInt()

```
void rxInt (
            int16 ct,
            void * ptr )
```

Receive interrupt handler.

This is an internal function that should not be called except by the driver. It's used to add one or more CAN frames to the receive buffer when they are received.

Parameters

ct	The number of frames to add
ptr	Points to an array of ct VCI_CAN_OBJ structures.

5.70.3.6 SetBaud()

```
const Error * SetBaud (
            int32 b )  [virtual]
```

Set the CAN interface baud rate.

Parameters

b	The baud rate to set.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.70.3.7 XmitFrame()

```
const Error * XmitFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Write a CAN frame to the CAN network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.70.4 Member Data Documentation

5.70.4.1 channel

```
uint8 channel  [protected]
```

Which CAN channel to use (on multi-channel boards).

For the moment this is always set to zero.

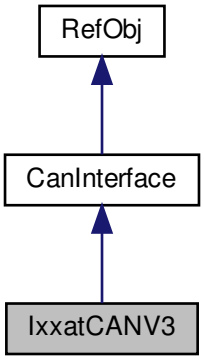
The documentation for this class was generated from the following files:

- can_ixxat.h
- can_ixxat.cpp

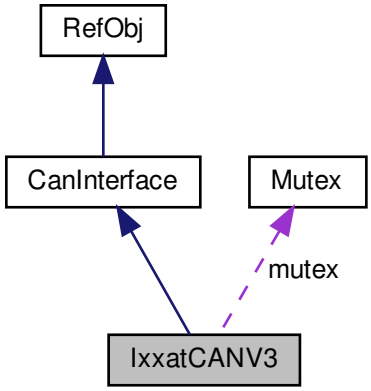
5.71 IxxatCANV3 Class Reference

Ixxat specific CAN interface.

Inheritance diagram for IxxatCANV3:



Collaboration diagram for IxxatCANV3:



Public Member Functions

- const Error * Open (void)

Open the CAN interface.

- const Error * Close (void)

  Close the CAN interface.

- const Error * SetBaud (int32 baud)

  Set the CAN interface baud rate.

Protected Member Functions

- const Error * RecvFrame (CanFrame &frame, Timeout timeout)

  Receive the next CAN frame.

- const Error * XmitFrame (CanFrame &frame, Timeout timeout)

  Write a CAN frame to the CAN network.

- const Error * ConvertError (int err)

  Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Protected Attributes

- int open

  tracks the state of the interface as open or closed.

- uint8 channel

  Which CAN channel to use (on multi-channel boards).

- int32 baud

  Holds a copy of the last baud rate set.

- int handle

  File handle used to access the CAN channel.

Additional Inherited Members

5.71.1 Detailed Description

Ixxat specific CAN interface.

This class extends the generic CanInterface class into a working interface for the Ixxat can device driver.

5.71.2 Member Function Documentation

5.71.2.1 Close()

```
const Error * Close (
            void  )  [virtual]
```

Close the CAN interface.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.71.2.2 ConvertError()

```
const Error * ConvertError (
            int err )  [protected]
```

Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Parameters

err	The Vector style status code

Returns

A pointer to an error object, or NULL if no error is indicated

5.71.2.3 Open()

```
const Error * Open (
            void  )  [virtual]
```

Open the CAN interface.

Returns

A valid CAN error object

Reimplemented from CanInterface.

5.71.2.4 RecvFrame()

```
const Error * RecvFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Receive the next CAN frame.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.71.2.5 SetBaud()

```
const Error * SetBaud (
            int32 b )  [virtual]
```

Set the CAN interface baud rate.

Parameters

b	The baud rate to set.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.71.2.6 XmitFrame()

```
const Error * XmitFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Write a CAN frame to the CAN network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A pointer to an error object on failure, or NULL on success.

Reimplemented from CanInterface.

5.71.3 Member Data Documentation

5.71.3.1 channel

```
uint8 channel  [protected]
```

Which CAN channel to use (on multi-channel boards).

For the moment this is always set to zero.

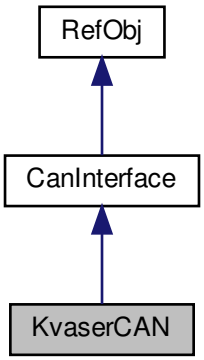
The documentation for this class was generated from the following files:

- can_ixxat_v3.h
- can_ixxat_v3.cpp

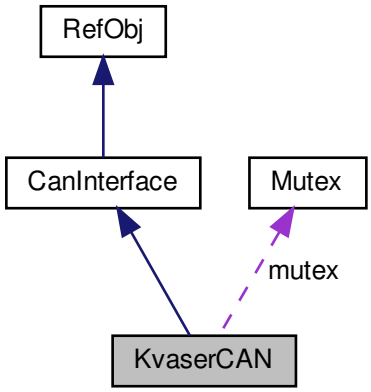
5.72 KvaserCAN Class Reference

Kvaser specific CAN interface.

Inheritance diagram for KvaserCAN:



Collaboration diagram for KvaserCAN:



Public Member Functions

- KvaserCAN (void)

  Construct a default CAN object.
- KvaserCAN (const char *port)

  Construct a CAN object with a specified port name.
- virtual ~KvaserCAN (void)

  Destructor.
- const Error * Open (void)

  Open the Kvaser CAN port.
- const Error * Close (void)

  Close the CAN interface.
- const Error * SetBaud (int32 baud)

  Set the CAN interface baud rate.

Protected Member Functions

- const Error * RecvFrame (CanFrame &frame, Timeout timeout)

  Receive the next CAN frame.
- const Error * XmitFrame (CanFrame &frame, Timeout timeout)

  Write a CAN frame to the CAN network.
- const Error * ConvertError (int err)

  Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Protected Attributes

- int open

  tracks the state of the interface as open or closed.
- int32 baud

  Holds a copy of the last baud rate set.
- int kvBaud

  Holds a value the Kvaser driver uses to identify bit rate.
- int Handle_Rd

  File handle used to configure and read from the CAN channel.
- int Handle_Wr

  File handle used to write to the CAN channel.

Additional Inherited Members

5.72.1 Detailed Description

Kvaser specific CAN interface.

This class extends the generic CanInterface class into a working interface for the Kvaser can device driver.

5.72.2 Constructor & Destructor Documentation

5.72.2.1 KvaserCAN() [1/2]

```
KvaserCAN (
            void  )
```

Construct a default CAN object.

The CAN interface is closed initially, and no port name is selected.

5.72.2.2 KvaserCAN() [2/2]

```
KvaserCAN (
            const char * port )
```

Construct a CAN object with a specified port name.

The port name should be of the form CANx or KVASERx where x is the port number. The port numbers start at 0, so the first port would be identified by the port name CAN0.

Parameters

port	The port name string identifying the CAN device.

5.72.2.3 ~KvaserCAN()

```
~KvaserCAN (
            void  )  [virtual]
```

Destructor.

This closes the CAN port and frees the .dll

5.72.3 Member Function Documentation

5.72.3.1 Close()

```
const Error * Close (
            void  )  [virtual]
```

Close the CAN interface.

Returns

A pointer to an error object on failure, NULL on success.

Reimplemented from CanInterface.

5.72.3.2 ConvertError()

```
const Error * ConvertError (
            int err )  [protected]
```

Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Parameters

err	The Vector style status code

Returns

A pointer to an error object on failure, NULL on success.

5.72.3.3 Open()

```
const Error * Open (
            void  )  [virtual]
```

Open the Kvaser CAN port.

Before Open is called, the desired baud rate must have been specified by calling SetBaud, and the port name must have been set. If the baud was not specified, it will default to 1,000,000 BPS. If the port name is not set, it will default to CAN0.

Returns

A pointer to an error object on failure, NULL on success.

Reimplemented from CanInterface.

5.72.3.4 RecvFrame()

```
const Error * RecvFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Receive the next CAN frame.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A pointer to an error object on failure, NULL on success.

Reimplemented from CanInterface.

5.72.3.5 SetBaud()

```
const Error * SetBaud (
            int32 b )  [virtual]
```

Set the CAN interface baud rate.

Parameters

b	The baud rate to set.

Returns

A pointer to an error object on failure, NULL on success.

Reimplemented from CanInterface.

5.72.3.6 XmitFrame()

```
const Error * XmitFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Write a CAN frame to the CAN network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A pointer to an error object on failure, NULL on success.

Reimplemented from CanInterface.

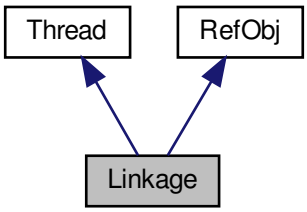
The documentation for this class was generated from the following files:

- can_kvaser.h
- can_kvaser.cpp

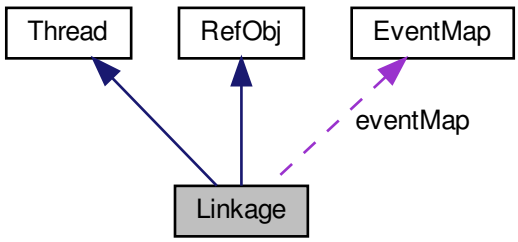
5.73 Linkage Class Reference

Linkage object, used for controlling a group of coordinated amplifiers.

Inheritance diagram for Linkage:



Collaboration diagram for Linkage:



Public Member Functions

- Linkage ()
  Default constructor.
- virtual ~Linkage ()
  Linkage object destructor.
- const Error * Init (uint16 ct, Amp a[])
  Initialize a new linkage object.
- const Error * Init (uint16 ct, Amp *a[])
  Initialize a new linkage object.
- const Error * Configure (LinkSettings &settings)

Configure a linkage.

- Amp & GetAmp (uint16 i)

  Get a reference to the amplifier object at the specified location in the linkage.
- uint32 GetAmpRef (uint16 i)

  Return a CML reference number for the amp object at the specified location in the linkage.
- const Error * MoveTo (PointN &p, uunit vel, uunit acc, uunit dec, uunit jrk, bool start=true)

  Move to a point in space.
- const Error * MoveToRel (PointN &p, uunit vel, uunit acc, uunit dec, uunit jrk, bool start=true)

  Move to a point in space using relative distances.
- const Error * SetMoveLimits (uunit vel, uunit acc, uunit dec, uunit jrk)

  Set limits used for multi-axis point-to-point moves.
- const Error * GetMoveLimits (uunit &vel, uunit &acc, uunit &dec, uunit &jrk)

  Return the move limits currently set for this linkage.
- const Error * MoveTo (PointN &p, bool start=true)

  Move to a specified position.
- const Error * MoveToRel (PointN &p, bool start=true)

  Move to a point in space using relative distances.
- const Error * StartMove (void)

  Start the moves that have already been programmed into all axes of this linkage.
- const Error * WaitMoveDone (Timeout timeout=-1)

  Wait for the currently running move to finish, or for an error to occur.
- const Error * WaitEvent (Event &e, Timeout timeout, LINK_EVENT &match, EventMap &map)

  Wait for a linkage event condition.
- const Error * WaitEvent (Event &e, Timeout timeout, LINK_EVENT &match)

  Wait for a linkage event condition.
- const Error * WaitEvent (Event &e, Timeout timeout=-1)

  Wait for a linkage event condition.
- const Error * HaltMove (void)

  Halt the current move.
- const Error * SendTrajectory (LinkTrajectory &trj, bool start=true)

  Upload a multi-axis PVT move trajectory to the linkage and optionally start the move.
- const Error * GetLatchedError (int &amp)

  Return any latched error codes held by the linkage object.
- void ClearLatchedError (void)

  Clear any latched errors.
- Amp & operator[ ] (uint16 i)

  Return a reference to the specified amplifier object in this linkage.
- uint16 GetAmpCount (void)

  Return the number of amplifiers associated with this linkage.
- virtual uint16 GetAxesCount (void)

  Return the number of independent axes associated with this linkage.
- const Error * GetPositionCommand (PointN &p)

  Get the current commanded position of the linkage.
- virtual const Error * ConvertAmpToAxisPos (uunit pos[ ])

  Convert the linkage position from the amplifier frame to the axis frame.
- virtual const Error * ConvertAxisToAmpPos (uunit pos[ ])

  Convert the linkage position from the axis frame to the amplifier frame.

- virtual const Error * ConvertAmpToAxis (uunit pos[ ], uunit vel[ ])

  Convert position & velocity information from the amplifier frame to the axis frame.
- virtual const Error * ConvertAxisToAmp (uunit pos[ ], uunit vel[ ])

  Convert position & velocity information from the axis frame to the amplifier frame.
- uint32 GetNetworkRef (void)

  Return a CML reference number for the Network this linkage is associated with.
- NetworkType GetNetworkType (void)

  Return the network type for the network this linkage is associated with.

Public Attributes

- EventMap eventMap

  This is an event map that is used to track linkage events and state changes.

Friends

- class **Amp**

Additional Inherited Members

5.73.1 Detailed Description

Linkage object, used for controlling a group of coordinated amplifiers.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 Linkage()

```
Linkage (
            void  )
```

Default constructor.

Linkage::Init must be called before this linkage object may be used.

5.73.3 Member Function Documentation

5.73.3.1 ClearLatchedError()

```
void ClearLatchedError (
            void  )  [inline]
```

Clear any latched errors.

This function clears the latched error information returned by Linkage::GetLatchedError(). Latched errors are automatically cleared at the start of a new move. This call may be used to clear latched error information at any other time.

5.73.3.2 Configure()

```
const Error * Configure (
            LinkSettings & settings )
```

Configure a linkage.

The linkage object will be configured to use the various settings passed in the LinkSettings object.

When a new Linkage object is created, it will be configured using a default set of settings. These settings can be modified through this call. Set the documentation of the LinkSettings object for details of the available settings and their default values.

Parameters

settings	The new settings to be used. A local copy of this object will be made by the linkage.

Returns

An error object pointer, or NULL on success.

5.73.3.3 ConvertAmpToAxis()

```
virtual const Error* ConvertAmpToAxis (
            uunit pos[],
            uunit vel[] )  [inline], [virtual]
```

Convert position & velocity information from the amplifier frame to the axis frame.

The passed arrays contain a position and velocity for each amplifer on entry. These values should be converted to axis positions & velocities in this function.

By default, this function doesn't do anything, however it is a virtual function to allow it to be extended in sub-classes.

Parameters

pos	An array of amplifer positions on entry, and axes positions on exit.
vel	An array of amplifier velocities on entry, and axis velocities on exit.

Returns

NULL on success or an Error pointer on failure.

5.73.3.4 ConvertAmpToAxisPos()

```
virtual const Error* ConvertAmpToAxisPos (
            uunit pos[] )  [inline], [virtual]
```

Convert the linkage position from the amplifier frame to the axis frame.

The passed array contains a position for each amplifer on entry. These positions should be converted to axis positions in this function. By default, this function doesn't do anything, however it is a virtual function to allow it to be extended in sub-classes.

Parameters

pos	An array of amplifer positions on entry, and axes positions on exit.

Returns

NULL on success or an Error pointer on failure.

5.73.3.5 ConvertAxisToAmp()

```
virtual const Error* ConvertAxisToAmp (
            uunit pos[],
            uunit vel[] )  [inline], [virtual]
```

Convert position & velocity information from the axis frame to the amplifier frame.

The passed arrays contain a position and velocity for each axis on entry. These values should be converted to amp positions & velocities in this function.

By default, this function doesn't do anything, however it is a virtual function to allow it to be extended in sub-classes.

Parameters

pos	An array of axis positions on entry, and amplifier positions on exit.
vel	An array of axis velocities on entry, and amplifier velocities on exit.

Returns

NULL on success or an Error pointer on failure.

5.73.3.6 ConvertAxisToAmpPos()

```
virtual const Error* ConvertAxisToAmpPos (
            uunit pos[] )  [inline], [virtual]
```

Convert the linkage position from the axis frame to the amplifier frame.

The passed array contains a position for each axis on entry. These positions should be converted to amplifier positions in this function. By default, this function doesn't do anything, however it is a virtual function to allow it to be extended in sub-classes.

Parameters

pos	An array of axis positions on entry, and amp positions on exit.

Returns

NULL on success or an Error pointer on failure.

5.73.3.7 GetAmp()

```
Amp & GetAmp (
            uint16 i )
```

Get a reference to the amplifier object at the specified location in the linkage.

Note that if CML_DEBUG_ASSERT is defined, then the standard C assert function will be used to check for an invalid index.

NOTE: This function is unsafe and has been depreciated. Use Linkage::GetAmpRef() as a safer alternative.

Parameters

i	The index of the amplifier to access.

Returns

A reference to the amplifier object.

5.73.3.8 GetAmpCount()

```
uint16 GetAmpCount (
            void  )  [inline]
```

Return the number of amplifiers associated with this linkage.

Returns

The amplifier count.

5.73.3.9 GetAmpRef()

```
uint32 GetAmpRef (
            uint16 i )
```

Return a CML reference number for the amp object at the specified location in the linkage.

RefObj::LockRef() can then be used to safely obtain a pointer to the actual Amp object.

Parameters

i	The index of the amplifier to access.

Returns

A reference to the amplifier object.

5.73.3.10 GetAxesCount()

```
virtual uint16 GetAxesCount (
            void  )  [inline], [virtual]
```

Return the number of independent axes associated with this linkage.

For a standard Linkage object, this will be the same as the amp count, however, this function is virtual to allow more complex structures to be represented in sub-classes.

Returns

The number of independent axes for this Linkage.

5.73.3.11 GetLatchedError()

```
const Error* GetLatchedError (
             int & amp )  [inline]
```

Return any latched error codes held by the linkage object.

When an error occurs during a move, the linkage latches the first error to occur and the index of the amplifier that caused it.

Note that the latched error information will be reset automatically at the start of any new move.

Parameters

amp	The index of the amplifier producing the latched error will be returned. -1 will be returned if the amplifier is unknown.

Returns

A pointer to the latched error object, or NULL if no error was latched.

5.73.3.12 GetMoveLimits()

```
const Error * GetMoveLimits (
              uunit & vel,
              uunit & acc,
              uunit & dec,
              uunit & jrk )
```

Return the move limits currently set for this linkage.

Parameters

vel	Returns maximum velocity
acc	Returns maximum acceleration
dec	Returns maximum deceleration
jrk	Returns maximum jerk

Returns

An error object pointer, or NULL on success.

5.73.3.13 GetPositionCommand()

```
const Error * GetPositionCommand (
            PointN & p )
```

Get the current commanded position of the linkage.

Note that this function queries the position of each amplifier sequentially and therefore the returned position information will only be accurate if the linkage is at rest when the function is called.

Parameters

p	A point that will be filled in with the current Linkage commanded position.

Returns

An error object pointer, or NULL on success.

5.73.3.14 HaltMove()

```
const Error * HaltMove (
            void  )
```

Halt the current move.

The exact type of halt can be programmed individually for each axis using the Amp::SetHaltMode function.

Returns

An error object pointer, or NULL on success.

5.73.3.15 Init() [1/2]

```
const Error * Init (
            uint16 ct,
            Amp a[] )
```

Initialize a new linkage object.

If the object has already been initialized, this will fail with an error.

All amplifiers attached to a linkage must be initialized, and must share the same network object. Also, amplifiers may only be attached to one linkage at a time, so this function will fail if any of the passed amplifier objects is already attached to a Linkage.

The linkage object will maintain pointers to each of the amplifier objects passed to this function. The amplifiers may not be destroyed until after the linkage object is.

Parameters

ct	The number of amplifiers to be used with this linkage. Note that this must be between 1 and CML_MAX_AMPS_PER_LINK.
a	An array of amplifiers to be assigned to this linkage. There must be at least ct amplifiers in this array.

Returns

An error object pointer, or NULL on success.

5.73.3.16 Init() [2/2]

```
const Error * Init (
            uint16 ct,
            Amp * a[] )
```

Initialize a new linkage object.

If the object has already been initialized, this will fail with an error.

All amplifiers attached to a linkage must be initialized, and must share the same network object. Also, amplifiers may only be attached to one linkage at a time, so this function will fail if any of the passed amplifier objects is already attached to a Linkage.

The linkage object will maintain pointers to each of the amplifier objects passed to this function. The amplifiers may not be destroyed until after the linkage object is.

Parameters

ct	The number of amplifiers to be used with this linkage. Note that this must be between 1 and CML_MAX_AMPS_PER_LINK.
a	An array of pointer to amplifier objects to be assigned to this linkage. There must be at least ct pointers in this array.

Returns

An error object pointer, or NULL on success.

5.73.3.17 MoveTo() [1/2]

```
const Error * MoveTo (
            PointN & p,
            uunit vel,
            uunit acc,
            uunit dec,
            uunit jrk,
            bool start = true )
```

Move to a point in space.

The number of dimensions of the point must equal the number of axes controlled by the Linkage (as returned by Linkage::GetAxesCount).

This method causes the linkage to perform a straight line move in N space from the present position to the specified point. The move will be limited in velocity, acceleration & jerk to the passed values.

The linkage is assumed to be at rest when this method is called. If this isn't the case, then an error will result.

Note that this function causes a trajectory to be calculated and passed to the amplifiers as a series of PVT points. This calculation requires floating point math, so this function is not available if floating point support has not been enabled in CML_Settings.h.

Parameters

p	The point in N space to move to.
vel	Maximum velocity
acc	Maximum acceleration
dec	Maximum deceleration
jrk	Maximum jerk
start	If true (the default), the profile will be started by this call. If false, the profile will be uploaded, but not started. In that case the move may be later started by a call to Linkage::StartMove.

Returns

An error object pointer, or NULL on success.

5.73.3.18 MoveTo() [2/2]

```
const Error * MoveTo (
            PointN & p,
            bool start = true )
```

Move to a specified position.

This move uses the limits previously set using Linkage::SetMoveLimits.

Parameters

p	The point to move to.
start	If true (the default), the profile will be started by this call. If false, the profile will be uploaded, but not started. In that case the move may be later started by a call to Linkage::StartMove.

Returns

An error object pointer, or NULL on success.

5.73.3.19 MoveToRel() [1/2]

```
const Error * MoveToRel (
            PointN & p,
            uunit vel,
            uunit acc,
            uunit dec,
            uunit jrk,
            bool start = true )
```

Move to a point in space using relative distances.

The number of dimensions of the point must equal the number of axes controlled by the Linkage (as returned by Linkage::GetAxesCount).

This method causes the linkage to perform a straight line move in N space from the present position to the specified point using relative distances. The move will be limited in velocity, acceleration & jerk to the passed values.

The linkage is assumed to be at rest when this method is called. If this isn't the case, then an error will result.

Note that this function causes a trajectory to be calculated and passed to the amplifiers as a series of PVT points. This calculation requires floating point math, so this function is not available if floating point support has not been enabled in CML_Settings.h.

Parameters

p	The relative distances to the desired point in N space.
vel	Maximum velocity
acc	Maximum acceleration
dec	Maximum deceleration
jrk	Maximum jerk
start	If true (the default), the profile will be started by this call. If false, the profile will be uploaded, but not started. In that case the move may be later started by a call to Linkage::StartMove.

Returns

An error object pointer, or NULL on success.

5.73.3.20 MoveToRel() [2/2]

```
const Error * MoveToRel (
            PointN & p,
            bool start = true )
```

Move to a point in space using relative distances.

This move uses the limits previously set using Linkage::SetMoveLimits.

Parameters

p	The relative distances to the desired point in N space.
start	If true (the default), the profile will be started by this call. If false, the profile will be uploaded, but not started. In that case the move may be later started by a call to Linkage::StartMove.

Returns

An error object pointer, or NULL on success.

5.73.3.21 operator[]()

```
Amp& operator[] (
            uint16 i )  [inline]
```

Return a reference to the specified amplifier object in this linkage.

This is the same as Linkage::GetAmp

NOTE: This function is unsafe and has been depreciated. Use Linkage::GetAmpRef() as a safer alternative.

Parameters

i	The amplifier index location

Returns

A reference to the amp object

5.73.3.22 SendTrajectory()

```
const Error * SendTrajectory (
            LinkTrajectory & trj,
            bool start = true )
```

Upload a multi-axis PVT move trajectory to the linkage and optionally start the move.

Parameters

trj	Reference to the linkage trajectory to be used. A local reference to this trajectory will be stored if the entire profile will not fit in the amplifiers on-board buffer. This pointer will be kept until the entire profile has been uploaded to the linkage. It is therefore important to ensure that the trajectory object passed here will remain valid (i.e. not be deallocated) until the linkage has called the LinkTrajectory.Finish() method.
start	If true (the default), the profile will be started by this call. If false, the profile will be uploaded, but not started. In that case the move may be later started by a call to Linkage::StartMove.

Returns

An error object.

5.73.3.23 SetMoveLimits()

```
const Error * SetMoveLimits (
            uunit vel,
            uunit acc,
            uunit dec,
            uunit jrk )
```

Set limits used for multi-axis point-to-point moves.

Parameters

vel	Maximum velocity
acc	Maximum acceleration
dec	Maximum deceleration
jrk	Maximum jerk

Returns

An error object pointer, or NULL on success.

5.73.3.24 StartMove()

```
const Error * StartMove (
            void  )
```

Start the moves that have already been programmed into all axes of this linkage.

Returns

An error object pointer, or NULL on success.

5.73.3.25 WaitEvent() [1/3]

```
const Error * WaitEvent (
            Event & e,
            Timeout timeout,
            LINK_EVENT & match,
            EventMap & map )
```

Wait for a linkage event condition.

This function additionally polls the drive status while waiting to ensure a lost message won't cause a timeout

Parameters

e	The event to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever.
match	Returns the matching event condition.
map	Event map to wait on

Returns

A pointer to an error object, or NULL on success.

5.73.3.26 WaitEvent() [2/3]

```
const Error * WaitEvent (
            Event & e,
            Timeout timeout,
            LINK_EVENT & match )
```

Wait for a linkage event condition.

This function can be used to wait on any generic event associated with the linkage.

Parameters

e	The event to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever.
match	Returns the matching event condition.

Returns

A pointer to an error object, or NULL on success.

5.73.3.27 WaitEvent() [3/3]

```
const Error * WaitEvent (
            Event & e,
            Timeout timeout = -1 )
```

Wait for a linkage event condition.

This function can be used to wait on any generic event associated with the linkage.

Parameters

e	The event to wait on.
timeout	The timeout for the wait (milliseconds). If < 0, then wait forever (default).

Returns

A pointer to an error object, or NULL on success.

5.73.3.28 WaitMoveDone()

```
const Error * WaitMoveDone (
            Timeout timeout = -1 )
```

Wait for the currently running move to finish, or for an error to occur.

Parameters

timeout	The maximum time to wait (milliseconds). Default is -1 (forever).

Returns

A pointer to an error object, or NULL on success.

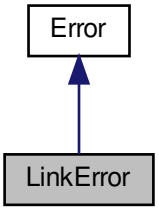
The documentation for this class was generated from the following files:

- CML_Linkage.h
- Linkage.cpp

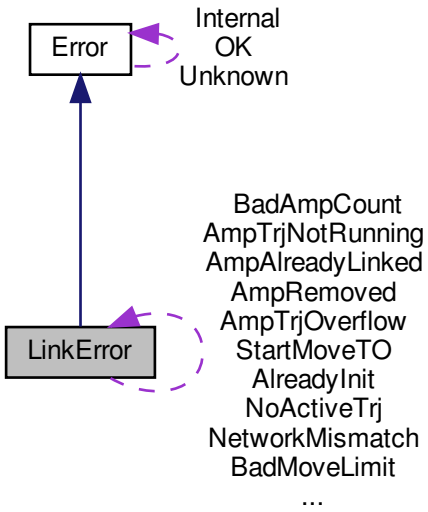
5.74 LinkError Class Reference

This class represents error conditions that can occur in the Linkage class.

Inheritance diagram for LinkError:



Collaboration diagram for LinkError:



Static Public Attributes

- static const LinkError NetworkMismatch

  The amplifier objects used to init the linkage are not all attached to the same CanOpen network.
- static const LinkError BadAmpCount

  An illegal number of amplifiers was passed to Linkage::Init.
- static const LinkError AlreadyInit

  Init was called on a Linkage that is already initialized.
- static const LinkError AmpAlreadyLinked

  The passed amplifier object is already assigned to a linkage.
- static const LinkError AxisCount

  The point dimension doesn't match the number of linkage axes.
- static const LinkError AmpTrjOverflow

  Amplifier trajectory structure overflow.
- static const LinkError AmpTrjInUse

  Amplifier trajectory already in use.
- static const LinkError AmpTrjNotRunning

  Amplifier trajectory not presently in use.
- static const LinkError NoActiveTrj

  No linkage trajectory is active.
- static const LinkError BadMoveLimit

  A zero or negative move limit was detected.
- static const LinkError UnknownAmpErr

Unknown amplifier error.

- static const LinkError StartMoveTO

  Timeout waiting on amplifier to respond to start move command.

- static const LinkError NotSupported

  Returned if Linkage::MoveTo is called on a system where floating point math was not enabled at compile time.

- static const LinkError AmpRemoved

  An amp object referenced by the linkage is no longer valid.

Protected Member Functions

- LinkError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.74.1 Detailed Description

This class represents error conditions that can occur in the Linkage class.

5.74.2 Member Data Documentation

5.74.2.1 NetworkMismatch

```
const LinkError NetworkMismatch  [static]
```

The amplifier objects used to init the linkage are not all attached to the same CanOpen network.

5.74.2.2 NotSupported

```
const LinkError NotSupported  [static]
```

Returned if Linkage::MoveTo is called on a system where floating point math was not enabled at compile time.

The documentation for this class was generated from the following file:

- CML_Linkage.h

5.75 LinkSettings Class Reference

Linkage object settings.

Public Member Functions

- LinkSettings ()

  Default constructor.

Public Attributes

- Timeout moveAckTimeout

  This setting gives the amount of time (milliseconds) to wait for all amplifiers to acknowedge the start of a new move before reporting an error.
- bool haltOnPosWarn

  If this setting is set to true, then the linkage object will automatically issue a halt to all axes if any of them reports a position warning window condition during a move.
- bool haltOnVelWin

  If this setting is set to true, then the linkage object will automatically issue a halt to all axes if any of them reports a velocity tracking window condition during a move.

5.75.1 Detailed Description

Linkage object settings.

An object of this type may be passed to the Linkage::Configure function to define the settings used by that linkage.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 LinkSettings()

```
LinkSettings ( )
```

Default constructor.

All settings are set to their default values at construction time.

5.75.3 Member Data Documentation

5.75.3.1 haltOnPosWarn

```
bool haltOnPosWarn
```

If this setting is set to true, then the linkage object will automatically issue a halt to all axes if any of them reports a position warning window condition during a move.

Default: false

5.75.3.2 haltOnVelWin

```
bool haltOnVelWin
```

If this setting is set to true, then the linkage object will automatically issue a halt to all axes if any of them reports a velocity tracking window condition during a move.

Default: false

5.75.3.3 moveAckTimeout

```
Timeout moveAckTimeout
```

This setting gives the amount of time (milliseconds) to wait for all amplifiers to acknowedge the start of a new move before reporting an error.

When a new move is started on the linkage, each amplifier will respond with an acknowledgment. If all of these responses are not received in this amount of time then an error will be reported.

Default: 200 ms

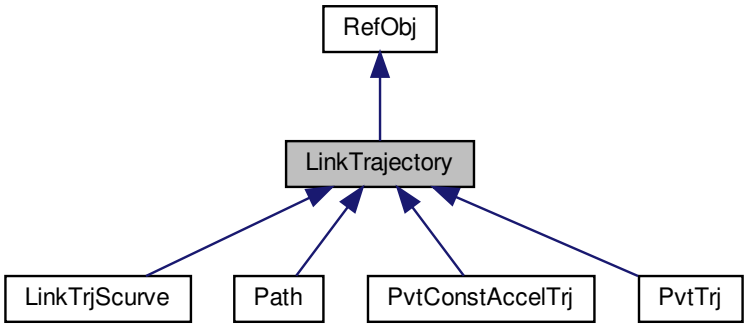
The documentation for this class was generated from the following files:

- CML_Linkage.h
- Linkage.cpp

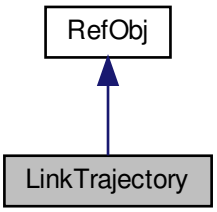
5.76 LinkTrajectory Class Reference

Linkage trajectory.

Inheritance diagram for LinkTrajectory:



Collaboration diagram for LinkTrajectory:



Public Member Functions

- LinkTrajectory (void)

  LinkTrajectory default constructor.
- virtual ~LinkTrajectory ()

  Virtual destructor.
- virtual const Error * StartNew (void)

  Start a new trajectory.
- virtual void Finish (void)

Trajectory finished.

- virtual int GetDim (void)=0

  Get the dimension of the trajectory.

- virtual bool UseVelocityInfo (void)

  This function indicates whether the velocity information returned by NextSegment should be used.

- virtual int MaximumBufferPointsToUse (void)

  This function allows a trajectory object to effectively reduce the size of the amplifier's internal trajectory buffer.

- virtual const Error * NextSegment (uunit pos[ ], uunit vel[ ], uint8 &time)=0

  Get the next segment of position, velocity & time info.

Additional Inherited Members

5.76.1 Detailed Description

Linkage trajectory.

This class is similar to the Trajectory class, except that it is used to pass multi-axis trajectory information to a linkage rather then single axis trajectory information to a single amplifier.

Like the base Trajectory class, the base LinkTrajectory class is pure virtual. This class should be extended by actual trajectory implementations.

5.76.2 Member Function Documentation

5.76.2.1 Finish()

```
virtual void Finish (
            void  )  [inline], [virtual]
```

Trajectory finished.

This function is called by the Linkage object when it is finished with the trajectory and no longer holding a reference to it. Typically, this will happen after the LinkTrajectory::NextSegment function has returned a zero time value, although it can also occur when some external event causes the trajectory to be aborted.

Once the Linkage object calls LinkTrajectory::Finish it will clear it's reference to the trajectory object. No further access to the trajectory object will be made after Finish is called.

Reimplemented in LinkTrjScurve.

5.76.2.2 GetDim()

```
virtual int GetDim (
            void  )  [pure virtual]
```

Get the dimension of the trajectory.

The trajectory dimension gives the number of axes defined for the trajectory. This can not change from the time Start↩ New() is called to the time Finish() is called. The position and velocity arrays passed to NextSegment will be of at least this size.

Returns

The dimension of the trajectory.

Implemented in Path, LinkTrjScurve, PvtTrj, and PvtConstAccelTrj.

5.76.2.3 MaximumBufferPointsToUse()

```
virtual int MaximumBufferPointsToUse (
            void  )  [inline], [virtual]
```

This function allows a trajectory object to effectively reduce the size of the amplifier's internal trajectory buffer.

Normally it's desirable to download as many points as possible to the amplifier at once. For some applicaitons however, the trajectory information is calculated in real time and the amplifier's buffer causes unacceptable latency. For such applications this function may be used to reduce the delay between calculating trajectory points and the amplifier's acting on them.

Note that the amplifier requires some buffering of points in order to interpolate between them. This function should never return a value less then 2 or a trajectory underflow will certainly occur.

Returns

The maximum number of trajectory points that should be stored in the amplifier at any time. By default this returns a very large number which ensures that the amplifier's full buffer will be used.

5.76.2.4 NextSegment()

```
virtual const Error* NextSegment (
            uunit pos[],
            uunit vel[],
            uint8 & time )  [pure virtual]
```

Get the next segment of position, velocity & time info.

Note that this function will be called from the high priority CANopen receiver task. Therefore, no lengthy processing should be done here.

Parameters

pos	An array where the position values will be returned. This array will be at least D elements long, where D is the trajectory dimension as returned by LinkTrajectory::GetDim()
vel	An array where the velocity values will be returned. These values are ignored if the function UseVelocityInfo() returns false.
time	The segment time is returned here. This is in milliseconds and ranges from 1 to 255. If zero is returned, this is the last frame in the profile.

Returns

A pointer to an error object on failure, or NULL on success.

Implemented in Path, and LinkTrjScurve.

5.76.2.5 StartNew()

```
virtual const Error* StartNew (
            void  )  [inline], [virtual]
```

Start a new trajectory.

This function is called before the first call to LinkTrajectory::NextSegment. It gives the trajectory object a chance to return an error indicating that it isn't ready to be sent.

Returns

An error pointer if the trajectory object is not available, or NULL if it is ready to be sent.

Reimplemented in Path, and LinkTrjScurve.

5.76.2.6 UseVelocityInfo()

```
virtual bool UseVelocityInfo (
            void  )  [inline], [virtual]
```

This function indicates whether the velocity information returned by NextSegment should be used.

If this returns true, then the amplifier will operate in PVT mode and use cubic polynomial interpolation between trajectory segments. If this returns false, then the velocity returned by NextSegment will be ignored, and the amplifier will run in PT mode using linear interpolation between sets of points.

Returns

true if velocity information should be used (default), or false if velocities should be ignored.

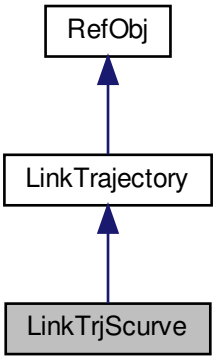
The documentation for this class was generated from the following file:

- CML_Trajectory.h

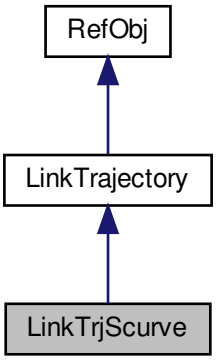
5.77 LinkTrjScurve Class Reference

Multi-axis s-curve profile.

Inheritance diagram for LinkTrjScurve:



Collaboration diagram for LinkTrjScurve:



Public Member Functions

- LinkTrjScurve ()

Default constructor for multi-axis s-curve trajectory.

- const Error * Calculate (PointN &start, PointN &end, uunit vel, uunit acc, uunit dec, uunit jrk)

  Calculate a multi-axis s-curve trajectory.
- int GetDim (void)

  Get the dimension of the trajectory.
- const Error * StartNew (void)

  Start a new move using this trajectory.
- void Finish (void)

  Finish this trajectory.
- const Error * NextSegment (uunit pos[ ], uunit vel[ ], uint8 &time)

  Retrieve the next segment of this trajectory.

Additional Inherited Members

5.77.1 Detailed Description

Multi-axis s-curve profile.

This extends the single axis TrjScurve object for use in multi-axis linkage moves.

5.77.2 Member Function Documentation

5.77.2.1 Calculate()

```
const Error * Calculate (
            PointN & startingPosArg,
            PointN & endingPosArg,
            uunit maxVelArgument,
            uunit maxAccelArgument,
            uunit maxDecelArgument,
            uunit maxJerkArgument )
```

Calculate a multi-axis s-curve trajectory.

This function calculates the straight line move between the two passed positions.

Parameters

s	The starting position
e	The ending position
vel	The max velocity
acc	The max acceleration
dec	The max deceleration
jrk	The max jerk (rate of change of velocity)

Returns

A pointer to an error object, or NULL on success.

5.77.2.2 GetDim()

```
int GetDim (
            void  )  [inline], [virtual]
```

Get the dimension of the trajectory.

The trajectory dimension gives the number of axes defined for the trajectory. This can not change from the time StartNew() is called to the time Finish() is called. The position and velocity arrays passed to NextSegment will be of at least this size.

Returns

The dimension of the trajectory.

Implements LinkTrajectory.

5.77.2.3 NextSegment()

```
const Error * NextSegment (
              uunit pos[],
              uunit vel[],
              uint8 & time )  [virtual]
```

Retrieve the next segment of this trajectory.

The positions & velocities for all axes are returned in the passed arrays.

Parameters

pos	An array which will be filled with position information.
vel	An array which will be filled with velocity information.
time	A reference to a variable where the time (milliseconds) will be returned.

Returns

A pointer to an error object, or NULL on success.

Implements LinkTrajectory.

5.77.2.4 StartNew()

```
const Error * StartNew (
            void  )  [virtual]
```

Start a new move using this trajectory.

The trajectory must have already been calculated when this function is called.

Returns

A pointer to an error object, or NULL on success.

Reimplemented from LinkTrajectory.

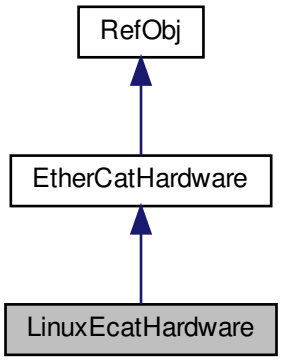
The documentation for this class was generated from the following files:

- CML_TrjScurve.h
- TrjScurve.cpp

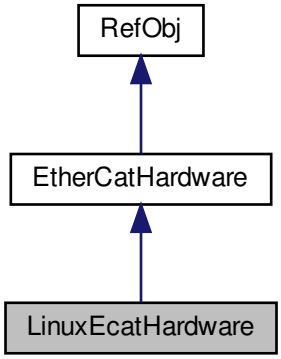
5.78 LinuxEcatHardware Class Reference

This class provides an interface to the Ethernet ports on a linux system.

Inheritance diagram for LinuxEcatHardware:



Collaboration diagram for LinuxEcatHardware:



Additional Inherited Members

5.78.1 Detailed Description

This class provides an interface to the Ethernet ports on a linux system.

It can be used to send and received formatted EtherCAT packets.

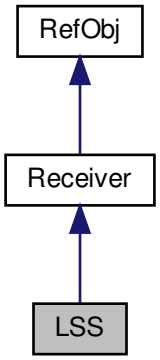
The documentation for this class was generated from the following files:

- ecat_linux.h
- ecat_linux.cpp

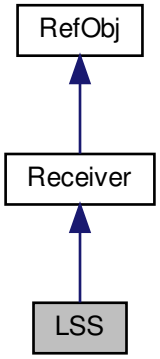
5.79 LSS Class Reference

CANopen Layer Setting Services object.

Inheritance diagram for LSS:



Collaboration diagram for LSS:



Public Member Functions

- LSS (CanOpen &co)

  Default constructor for the LSS object.
- int FindAmplifiers (int max, uint32 serial[ ])

  Search the CANopen network for Copley amplifiers.
- void setTimeout (Timeout to)

  Set the timeout value used by the LSS protocol.

- Timeout getTimeout (void)

  Get the current timeout value used by the LSS protocol.
- const Error * SwitchModeGlobal (bool config)

  Set all devices on the network into either LSS configuration mode or LSS operational mode.
- const Error * GetAmpNodeID (uint32 serial, byte &nodeID)

  Get the current CANopen node ID of the specified amplifier.
- const Error * SetAmpNodeID (uint32 serial, byte nodeID)

  Set the CANopen node ID of the specified amplifier.
- const Error * SetNodeID (byte nodeID)

  Set the CANopen node ID of the currently selected device.
- const Error * EnquireInfo (uint32 *vid, uint32 *prodID, uint32 *revNum, uint32 *serial, byte *nodeID)

  Read identification info from the selected device using LSS.
- const Error * SelectAmp (uint32 serial)

  Put the specified amplifier into LSS configure mode.
- const Error * SetBitRate (byte tableID, byte index)

  Send an LSS command to program the selected devices CAN bit rate given the table number and index within the table.
- const Error * SetBitRate (uint32 rate)

  Send an LSS command to program the selected devices CAN bit rate given the table number and index within the table.
- const Error * ActivateBitRate (uint16 delay)

  Activate the new bit rate previously set on all devices using the LSS::SetBitRate function.
- const Error * StoreConfig (void)

  Save the current node ID and bit rate info to non-volatile memory on the selected drive.

Protected Member Functions

- uint32 FindAmpSerial (uint32 low, uint32 high)

  Find the serial number of the first amplifier in the passed range.
- int NewFrame (CanFrame &frame)

  This method is called by the main CAN network listener when a new LSS response frame is received.
- const Error * Xmit (byte cs, uint32 data=0)

  Transmit a LSS CAN frame.
- const Error * Enquire (int cs)

  This internal function is used to enquire one field from the single selected drive using LSS.

Additional Inherited Members

5.79.1 Detailed Description

CANopen Layer Setting Services object.

The Layer Setting Services (LSS) protocol is part of the CANopen network standard. The intent of LSS is to allow low level network settings, such as the network bit rate and device node ID numbers to be configured over the network.

The CANopen protocol requires each device on the network to have a unique node ID number in the range 1 to 127. In general, it's not possible to communicate with a device using CANopen if it doesn't have a unique node ID in this range.

The LSS protocol allows some limited communication with any device on the network even if it doesn't have a node ID set. This allows node ID numbers to be assigned to devices over the network.

This object implements the LSS protocol and allows devices on the network to be queried and configured.

For more detailed information on the LSS protocol please see the CANopen standard document DSP305.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 LSS()

```
LSS (
            CanOpen & co )
```

Default constructor for the LSS object.

Parameters

co	A reference to the CANopen network object over which this protocol will run.

5.79.3 Member Function Documentation

5.79.3.1 ActivateBitRate()

```
const Error * ActivateBitRate (
            uint16 delay )
```

Activate the new bit rate previously set on all devices using the LSS::SetBitRate function.

Note that this function doesn't change the bit rate of the local CAN port, it simply returns after requesting the new rate on the LSS slave devices. It's the responsibility of the calling function to change the local bit rate.

After calling this function the master should stop transmitting on the CAN bus. After a delay of the specified time, the master should change it's bit rate. After a second delay of the specified time, the master can start transmitting on the bus at the new bit rate.

Parameters

delay	Delay in milliseconds which the LSS devices will use to ensure that they all switch their bit rates at a time when no device is transmitting. This delay will be used twice, once before the change and once after.

Returns

A pointer to an error object, or NULL on success.

5.79.3.2 Enquire()

```
const Error * Enquire (
            int cs )  [protected]
```

This internal function is used to enquire one field from the single selected drive using LSS.

Parameters

cs	The LSS code of the desired field

Returns

A pointer to an error object, or NULL on success.

5.79.3.3 EnquireInfo()

```
const Error * EnquireInfo (
            uint32 * vid,
            uint32 * prodID,
            uint32 * revNum,
            uint32 * serial,
            byte * nodeID )
```

Read identification info from the selected device using LSS.

Only one device should be selected when this is called.

Parameters

vid	If not NULL, the vendor ID will be returned here
prodID	If not NULL, the product ID will be returned here
revNum	If not NULL, the revision number will be returned here
serial	If not NULL, the serial number will be returned here
nodeID	If not NULL, the node ID will be returned here

Returns

A pointer to an error object, or NULL on success.

5.79.3.4 FindAmplifiers()

```
int FindAmplifiers (
            int max,
            uint32 serial[] )
```

Search the CANopen network for Copley amplifiers.

This function uses the CANopen Layer Setting Services (LSS) protocol to find amplifiers on the network. All Copley amplifiers on the CANopen network can be identified using this protocol, even if they do not have a valid CANopen node ID number configured.

On return from this function, the passed array will have been filled with the serial numbers of all the amplifiers found. These serial numbers may then be passed to LSS::SetAmpNodeID to assign a node ID number to the amplifier.

Note that firmware support for the LSS protocol was added starting with version 4.04. Any amplifier on the network with earlier firmware will not be discovered using this technique.

Parameters

max	The maximum number of amplifier serial numbers to be returned.
serial	An array where the amplifier serial numbers will be returned. This array must be at least max elements long.

Returns

The number of amplifiers actually found. This is not limited to the max parameter value

5.79.3.5 FindAmpSerial()

```
uint32 FindAmpSerial (
            uint32 low,
            uint32 high )  [protected]
```

Find the serial number of the first amplifier in the passed range.

Parameters

low	The lower limit of the range
high	The upper limit of the range

Returns

the serial number, or zero if no amp found.

5.79.3.6 GetAmpNodeID()

```
const Error * GetAmpNodeID (
            uint32 serial,
            byte & nodeID )
```

Get the current CANopen node ID of the specified amplifier.

Parameters

serial	The serial number of the amplifier to query.
nodeID	The node ID will be returned here.

Returns

A pointer to an error object, or NULL on success.

5.79.3.7 getTimeout()

```
Timeout getTimeout (
            void  )  [inline]
```

Get the current timeout value used by the LSS protocol.

Returns

The current timeout in milliseconds.

5.79.3.8 NewFrame()

```
int NewFrame (
            CanFrame & frame )  [protected], [virtual]
```

This method is called by the main CAN network listener when a new LSS response frame is received.

Parameters

frame	A reference to the CAN fame that was received

Returns

Non-zero if the frame was handled.

Reimplemented from Receiver.

5.79.3.9 SelectAmp()

```
const Error * SelectAmp (
            uint32 serial )
```

Put the specified amplifier into LSS configure mode.

All other amplifiers on the network are switched into LSS operational mode.

Parameters

serial	The serial number of the device to configure

Returns

A pointer to an error object, or NULL on success.

5.79.3.10 SetAmpNodeID()

```
const Error * SetAmpNodeID (
            uint32 serial,
            byte nodeID )
```

Set the CANopen node ID of the specified amplifier.

Parameters

serial	The serial number of the amplifier to update.
nodeID	The CANopen node ID to assign to this amplifier.

Returns

A pointer to an error object, or NULL on success.

5.79.3.11 SetBitRate() [1/2]

```
const Error * SetBitRate (
            byte tableID,
            byte index )
```

Send an LSS command to program the selected devices CAN bit rate given the table number and index within the table.

When this is called, only one device should be in LSS config mode.

Note that the new bit rate doesn't become active immedately, it must be activated first by calling LSS::ActivateBitRate.

Parameters

tableID	Specifies which table of bit rates to use (0 for standard table)
index	Specifies the index within the table of the desired bit rate.

Returns

A pointer to an error object, or NULL on success.

5.79.3.12 SetBitRate() [2/2]

```
const Error * SetBitRate (
            uint32 rate )
```

Send an LSS command to program the selected devices CAN bit rate given the table number and index within the table.

When this is called, only one device should be in LSS config mode.

Note that the new bit rate doesn't become active immedately, it must be activated first by calling LSS::ActivateBitRate.

Parameters

rate	The new bit rate in bits/sec

Returns

A pointer to an error object, or NULL on success.

5.79.3.13 SetNodeID()

```
const Error * SetNodeID (
            byte nodeID )
```

Set the CANopen node ID of the currently selected device.

When this is called, exactly one deivce on the network should be in LSS configuration mode.

Parameters

nodeID	The CANopen node ID to assign to the selected device.

Returns

A pointer to an error object, or NULL on success.

5.79.3.14 setTimeout()

```
void setTimeout (
            Timeout to )  [inline]
```

Set the timeout value used by the LSS protocol.

Parameters

to	The new timeout (milliseconds)

5.79.3.15 StoreConfig()

```
const Error * StoreConfig (
            void  )
```

Save the current node ID and bit rate info to non-volatile memory on the selected drive.

When this is called, exactly one drive should be in LSS configuration mode.

Returns

A pointer to an error object, or NULL on success.

5.79.3.16 SwitchModeGlobal()

```
const Error * SwitchModeGlobal (
            bool config )
```

Set all devices on the network into either LSS configuration mode or LSS operational mode.

Parameters

config	If true, put all devices into configuration mode. Otherwise, put everything into operational mode.

Returns

A pointer to an error object, or NULL on success.

5.79.3.17 Xmit()

```
const Error * Xmit (
            byte cs,
            uint32 data = 0 )  [protected]
```

Transmit a LSS CAN frame.

Parameters

cs	The command specifier for this frame.
data	The data passed with the frame.

Returns

A pointer to an error object, or NULL on success.

The documentation for this class was generated from the following files:

- CML_CanOpen.h
- LSS.cpp

5.80 MtrInfo Struct Reference

Motor information structure.

Public Member Functions

- MtrInfo (void)

  Motor info structure default constructor.

Public Attributes

- uint16 type

  Motor type.
- char mfgName [COPLEY_MAX_STRING]

  Name of the motor manufacturer.
- char model [COPLEY_MAX_STRING]

  Motor model number.
- int16 poles

  Number of pole pairs (i.e.
- uint16 resistance

  Motor resistance (10 milliohm units)
- uint16 inductance

  Motor inductance (10 microhenry units)
- uint32 trqPeak

  Peak torque (0.01 Newton millimeters)
- uint32 trqCont

  Continuous torque (0.01 Newton millimeters)
- uint32 trqConst

  Torque constant (0.01 Newton millimeters / Amp)
- uunit velMax

  Max velocity.
- uint32 backEMF

  Back EMF constant (10 millivolts / KRPM)
- uint32 inertia

  Inertia.
- bool tempSensor

  Motor has a temperature sensor (true/false)
- bool mtrReverse

  Reverse motor wiring if true.
- bool encReverse

  Reverse encoder direction if true.
- int16 hallType

  Type of hall sensors on motor. See documentation for details.
- int16 hallWiring

  Hall wiring code, see documentation for details.
- int16 hallOffset

  Hall offset (degrees)
- bool hasBrake

  Motor has a brake if true.
- int16 stopTime

  Delay (milliseconds) between disabling amp & applying brake During this time the amp will attempt to actively stop motor.
- int16 brakeDelay

  Delay (milliseconds) between applying brake & disabling PWM.
- uunit brakeVel

  Velocity below which brake will be applied.
- int16 encType

Encoder type. See documentation for details.

- int32 ctsPerRev

  Encoder counts / revolution (rotory motors only)
- int16 encUnits

  Encoder units (linear motor only)
- int16 encRes

  Encoder resolution (encoder units / count) - linear motors only.
- int32 eleDist

  Motor electrical distance (encoder units / electrical phase) - linear only.
- int16 mtrUnits

  Motor units (used by CME program)
- int32 stepsPerRev

  Microsteps / motor rotation (used for Stepnet amplifiers)
- int16 encShift

  Analog Encoder shift value (used only with Analog encoders)
- int32 ndxDist

  Index mark distance (reserved for future use)
- int16 loadEncType

  Load encoder type (0 for none).
- int32 loadEncRes

  Load encoder resolution.
- bool loadEncReverse

  Reverse load encoder if true.
- int32 gearRatio

  Load encoder gear ratio.
- uint16 resolverCycles

  Resolver cycles / rev.
- int16 hallVelShift

  Hall velocity shift value.
- uint32 mtrEncOptions

  Motor Encoder options.
- uint32 loadEncOptions

  Load Encoder options.

5.80.1 Detailed Description

Motor information structure.

This structure holds information about the motor connected to the amplifier.

The amplifier uses the information in this structure when controlling the motor. It is very important that the information provided to the amplifier be as accurate as possible for proper motor control.

Use the methods Amp::GetMtrInfo and Amp::SetMtrInfo to upload / download the information contained in this structure.

Note that unlike many amplifier parameters, motor parameters are always stored in non-volatile flash memory.

5.80.2 Constructor & Destructor Documentation

5.80.2.1 MtrInfo()

```
MtrInfo (
            void  )
```

Motor info structure default constructor.

This simply initializes all members to legal default values.

5.80.3 Member Data Documentation

5.80.3.1 gearRatio

```
int32 gearRatio
```

Load encoder gear ratio.

This parameter is used by the CME software and gives a ratio of motor encoder counts to load encoder counts.

5.80.3.2 hallVelShift

```
int16 hallVelShift
```

Hall velocity shift value.

This parameter is only used on servo systems where there is no encoder and digital hall sensors are used for velocity feedback. In that case, this shift value can be used to scale up the calculated velocity.

5.80.3.3 loadEncOptions

```
uint32 loadEncOptions
```

Load Encoder options.

This bitmapped parameter is used to specify various configuration options for the load encoder. See documentation.

5.80.3.4 loadEncRes

```
int32 loadEncRes
```

Load encoder resolution.

This is encoder counts/rev for rotory encoders, or nanometers/count for linear encoders.

5.80.3.5 loadEncType

```
int16 loadEncType
```

Load encoder type (0 for none).

See amplifier documentation for possible values.

5.80.3.6 mtrEncOptions

```
uint32 mtrEncOptions
```

Motor Encoder options.

This bitmapped parameter is used to specify various configuration options for the motor encoder. See documentation.

5.80.3.7 poles

```
int16 poles
```

Number of pole pairs (i.e.

number of electrical phases) per rotation. Used for rotory motors only.

5.80.3.8 resolverCycles

```
uint16 resolverCycles
```

Resolver cycles / rev.

This parameter gives the number of resolver cycles / motor rev. It's only used on systems that use a resolver for position feedback. Default is 1 cycle/rev.

The documentation for this struct was generated from the following files:

- CML_AmpStruct.h
- AmpStruct.cpp

5.81 Mutex Class Reference

This class represents an object that can be used by multiple threads to gain safe access to a shared resource.

Public Member Functions

- Mutex (void)

  Create a new mutex object.
- virtual ~Mutex ()

  Free any system resources associated with the mutex.
- const Error * Lock (void)

  Lock the mutex.
- const Error * Unlock (void)

  Unlock the mutex.

5.81.1 Detailed Description

This class represents an object that can be used by multiple threads to gain safe access to a shared resource.

If an attempt is made to lock a mutex that is currently locked by another thread, the thread attempting the lock will be suspended until the thread holding the lock releases it. Mutex objects are not required to allow recursive access.

5.81.2 Member Function Documentation

5.81.2.1 Lock()

```
const Error* Lock (
             void  )
```

Lock the mutex.

This function causes the calling function to gain exclusive access to the mutex object. If some other thread has the mutex locked when this method is called, the calling thread will block until the mutex is unlocked.

Returns

An error object.

5.81.2.2 Unlock()

```
const Error* Unlock (
             void  )
```

Unlock the mutex.

This function causes the calling thread to give up it's lock on the mutex. A task switch may occur before this call returns if a high priority task is currently trying to lock the mutex.

Returns

An error object.

The documentation for this class was generated from the following file:

- CML_Threads.h

5.82 MutexLocker Class Reference

This is a utility class that locks a mutex in it's constructor, and unlocks it in it's destructor.

Public Member Functions

- MutexLocker (void)
  Default constructor No mutex is locked by default.
- MutexLocker (Mutex &m)
  Lock the passed mutex.
- void Lock (Mutex &m)
  Lock the passed mutex and keep a pointer to it.
- void Unlock (void)
  Unlock and forget about the previously locked Mutex.
- ~MutexLocker ()
  If a mutex is currently being tracked it will be unlocked.

5.82.1 Detailed Description

This is a utility class that locks a mutex in it's constructor, and unlocks it in it's destructor.

It can be used to ensure that a mutex is properly unlocked when a function returns. Just create a temporary MutexLocker object and pass it the mutex to lock in it's constructor. The mutex will be automatically unlocked when the function returns and the MutexLocker is deleted.

5.82.2 Constructor & Destructor Documentation

5.82.2.1 MutexLocker()

```
MutexLocker (
             Mutex & m )  [inline]
```

Lock the passed mutex.

Parameters

m	The mutex to lock.

5.82.3 Member Function Documentation

5.82.3.1 Lock()

```
void Lock (
            Mutex & m )  [inline]
```

Lock the passed mutex and keep a pointer to it.

The mutex will be unlocked in the destructor if not explicitely unlocked first. If another mutex was previously locked then it will be forgotten.

Parameters

m	The mutex to lock.

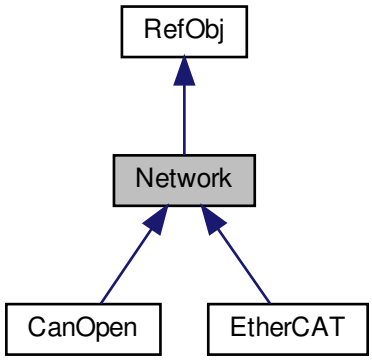
The documentation for this class was generated from the following file:

- CML_Threads.h

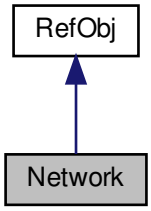
5.83 Network Class Reference

Abstract network class.

Inheritance diagram for Network:



Collaboration diagram for Network:



Public Member Functions

- virtual int32 maxSdoToNode (Node ∗n)

  Return the maximum number of bytes that can be sent in an SDO message.
- virtual int32 maxSdoFromNode (Node ∗n)

  Return the maximum number of bytes that can be received in an SDO message.

Static Public Member Functions

- static const char ∗ DescribeState (NodeState state)

  Return a string that describes the passed node state.

Static Protected Member Functions

- static NetworkNodeInfo * GetNodeInfo (Node *n)

  Return a pointer to the network information union embedded in this node.
- static void SetNodeInfo (Node *n, NetworkNodeInfo *ni)

  Set the network's node information for this node.

Additional Inherited Members

5.83.1 Detailed Description

Abstract network class.

This class forms the root of all the different networks that are supported by CML. Every device managed by CML is associated with exactly one network object. The network object manages some high level aspects of the communications with the device.

5.83.2 Member Function Documentation

5.83.2.1 DescribeState()

```
const char * DescribeState (
            NodeState state )  [static]
```

Return a string that describes the passed node state.

Parameters

state	The state to describe

Returns

A string describing the passed state

5.83.2.2 GetNodeInfo()

```
NetworkNodeInfo * GetNodeInfo (
            Node * n )  [static], [protected]
```

Return a pointer to the network information union embedded in this node.

This union contains data related to the node that is owned by the network layer object.

Parameters

n	Pointer to the node object

Returns

Pointer to the node information

5.83.2.3 maxSdoFromNode()

```
int32 maxSdoFromNode (
            Node * n )  [virtual]
```

Return the maximum number of bytes that can be received in an SDO message.

For CANopen this is always 8 (the max size of a CAN frame). For EtherCAT it's the size of the mailbox buffer, and is node specific

Parameters

n	The node to query

Returns

The maximum number of bytes in an SDO receive message

Reimplemented in EtherCAT.

5.83.2.4 maxSdoToNode()

```
int32 maxSdoToNode (
            Node * n )  [virtual]
```

Return the maximum number of bytes that can be sent in an SDO message.

For CANopen this is always 8 (the max size of a CAN frame). For EtherCAT it's the size of the mailbox buffer, and is node specific

Parameters

n	The node to query

Returns

The maximum number of bytes in an SDO transmit message

Reimplemented in EtherCAT.

5.83.2.5 SetNodeInfo()

```
void SetNodeInfo (
            Node * n,
            NetworkNodeInfo * ni )  [static], [protected]
```

Set the network's node information for this node.

This function is used internally by the network classes.

Parameters

n	Pointer to the node to update
ni	Pointer to the node information

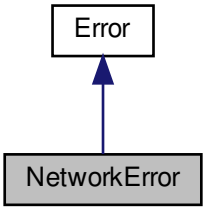
The documentation for this class was generated from the following files:

- CML_Network.h
- Network.cpp

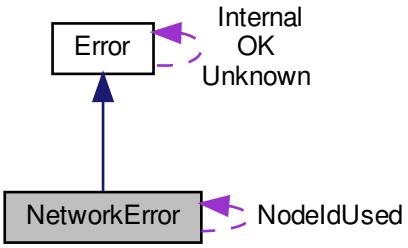
5.84 NetworkError Class Reference

This class holds the error codes that describe various Netowrk error conditions.

Inheritance diagram for NetworkError:



Collaboration diagram for NetworkError:



Static Public Attributes

- static const NetworkError NodeIdUsed

  A node with the specified ID is already present on the network.

Additional Inherited Members

5.84.1 Detailed Description

This class holds the error codes that describe various Netowrk error conditions.

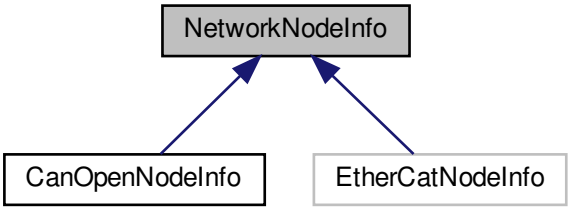
The documentation for this class was generated from the following file:

- CML_Network.h

5.85 NetworkNodeInfo Class Reference

Private data owned by the network object attached to every node.

Inheritance diagram for NetworkNodeInfo:



5.85.1 Detailed Description

Private data owned by the network object attached to every node.

The documentation for this class was generated from the following file:

- CML_Network.h

5.86 NetworkOptions Struct Reference

Configuration structure used to configure the amplifiers network support.

Public Member Functions

- NetworkOptions (void)
  Default constructor.

Public Attributes

- uint16 canBusConfig
  Network Options. The details fo this parameter depend on the type.

5.86.1 Detailed Description

Configuration structure used to configure the amplifiers network support.

These settings may be up/download from the amplifier using the functions Amp::SetNetworkOptions and Amp::Get↩ NetworkOptions.

5.86.2 Constructor & Destructor Documentation

5.86.2.1 NetworkOptions()

```
NetworkOptions (
            void  )  [inline]
```

Default constructor.

Initialize all structure elements to zero.

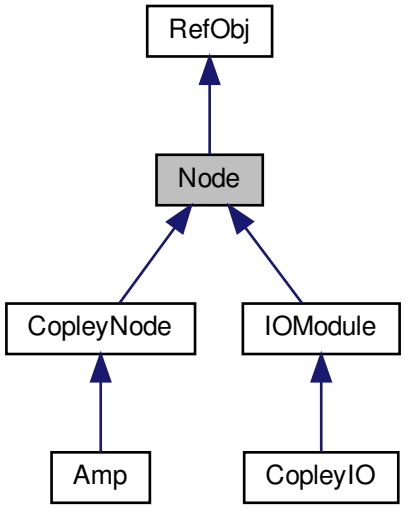
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

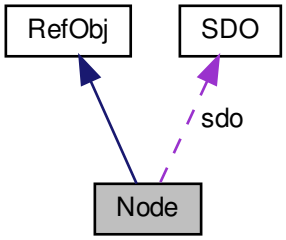
5.87 Node Class Reference

Node class.

Inheritance diagram for Node:



Collaboration diagram for Node:



Public Member Functions

- Node ()

Default CANopen node object constructor.

- Node (Network &net, int16 nodeID)

  Initialize the Node object.

- virtual ~Node ()

  CANopen node destructor.

- virtual const Error * StopGuarding (void)

  Disable node guarding & heartbeat monitoring.

- virtual const Error * StartHeartbeat (uint16 period, uint16 timeout)

  Enable heartbeat messages from this node, and start a thread to monitor them.

- virtual const Error * StartNodeGuard (uint16 guardTime, byte lifeFactor)

  Enable node guarding on this node.

- virtual const Error * Init (Network &co, int16 nodeID)

  Initialize the CANopen Node object.

- virtual const Error * UnInit (void)

  Un-initialize the Node object.

- virtual NetworkType GetNetworkType (void)

  Return a value that identifies the type of network the node is currently attached to.

- virtual uint32 GetNetworkRef (void)

  Return a reference ID to the network that this node is attached to.

- virtual const Error * PdoSet (uint16 n, PDO &pdo, bool enable=true)

  Associate the passed PDO object with this node.

- virtual const Error * PdoEnable (uint16 n, PDO &pdo)

  Enable the passed PDO object.

- virtual const Error * PdoDisable (uint16 n, PDO &pdo)

  Disable the passed PDO object.

- virtual const Error * RpdoDisable (uint16 n)

  Disable the specified receive PDO.

- virtual const Error * TpdoDisable (uint16 n)

  Disable the specified transmit PDO.

- virtual const Error * SavePDOMapping (bool saveProfile)

  Save the current PDO mapping to flash.

- virtual const Error * StartNode (void)

  Start this node.

- virtual const Error * StopNode (void)

  Stop this node.

- virtual const Error * PreOpNode (void)

  Put this node in pre-operational state.

- virtual const Error * ResetNode (void)

  Reset this node.

- virtual const Error * ResetComm (void)

  Reset this node's communications.

- virtual NodeState GetState (void)

  Returns the present state of this node.

- virtual const Error * GetDeviceType (uint32 &devType)

  Read the device type from the object dictionary.

- virtual const Error * GetErrorRegister (byte &err)

  Read the error register from the object dictionary.

- virtual const Error * GetMfgStatus (uint32 &stat)

  Read the manufacturer status register from the object dictionary.
- virtual const Error * GetErrorHistory (uint16 &ct, uint32 *array)

  Get the error history array (CANopen object 0x1003).
- virtual const Error * ClearErrorHistory (void)

  Clear the error history (object 0x1003) array for this node.
- virtual const Error * GetMfgDeviceName (int32 &len, char *str)

  Read the manufacturer's device name string from the object dictionary.
- virtual const Error * GetMfgHardwareVer (int32 &len, char *str)

  Read the manufacturer's Hardware version string from the object dictionary.
- virtual const Error * GetMfgSoftwareVer (int32 &len, char *str)

  Read the manufacturer's software version string from the object dictionary.
- virtual const Error * GetIdentity (NodeIdentity &id)

  Get the CANopen identity object for this node (object dictionary entry 0x1018).
- virtual const Error * SetSynchId (uint32 id)

  Set the COB-ID of the synch message.
- virtual const Error * GetSynchId (uint32 &id)

  Return the COB-ID of the synch message.
- virtual const Error * SetSynchPeriod (uint32 per)

  Set the SYNC message interval in microseconds.
- virtual const Error * GetSynchPeriod (uint32 &per)

  Get the SYNC message interval in microseconds.
- virtual const Error * SynchStart (void)

  Start producing SYNC messages on this node.
- virtual const Error * SynchStop (void)

  Stop producing SYNC messages on this node.
- virtual int16 GetNodeID (void)

  Return the node ID associated with this node.
- bool IsInitialized (void)

  Return true if this node object has been initialized.
- virtual int32 maxSdoToNode (void)

  Return the maximum number of bytes that can be sent in an SDO message.
- virtual int32 maxSdoFromNode (void)

  Return the maximum number of bytes that can be received in an SDO message.

Public Attributes

- SDO sdo

  This SDO may be used to get/set values in the node's object dictionary.

Protected Member Functions

- virtual void HandleEmergency (CanFrame &frame)

  Overload this function to handle emergency objects sent by this node.
- virtual void HandleStateChange (NodeState from, NodeState to)

  Overload this function to handle changes to the nodes state.

Friends

- class **Network**
- class **CanOpen**
- class **EtherCAT**

Additional Inherited Members

5.87.1 Detailed Description

Node class.

Objects of this class represent individual nodes on the CANopen or EtherCAT network.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 Node() [1/2]

```
Node ( )
```

Default CANopen node object constructor.

This constructor simple marks the object as uninitialized. The Init() function must be called before this object can be used.

5.87.2.2 Node() [2/2]

```
Node (
            Network & net,
            int16 nodeID )
```

Initialize the Node object.

Parameters

net	The network object that this node is associated with.
nodeID	The node's ID.

5.87.3 Member Function Documentation

5.87.3.1 ClearErrorHistory()

```
virtual const Error* ClearErrorHistory (
            void  )  [inline], [virtual]
```

Clear the error history (object 0x1003) array for this node.

Returns

An error object.

5.87.3.2 GetDeviceType()

```
virtual const Error* GetDeviceType (
            uint32 & devType )  [inline], [virtual]
```

Read the device type from the object dictionary.

Parameters

devType	Where the device type is returned

Returns

An error object

5.87.3.3 GetErrorHistory()

```
const Error * GetErrorHistory (
            uint16 & ct,
            uint32 * array )  [virtual]
```

Get the error history array (CANopen object 0x1003).

Parameters

ct	When the function is first called, this variable holds the maximum number of errors that can be stored in the err array (i.e. the length of the array). On return, the actual number of errors uploaded will be stored here.
array	An array of 32-bit integers that will be used to return the list of errors.

Returns

A pointer to an error object, or NULL on success

5.87.3.4 GetErrorRegister()

```
virtual const Error* GetErrorRegister (
            byte & err )  [inline], [virtual]
```

Read the error register from the object dictionary.

Parameters

err	Reference to where the error should be returned.

Returns

An error object

5.87.3.5 GetIdentity()

```
const Error * GetIdentity (
            NodeIdentity & id )  [virtual]
```

Get the CANopen identity object for this node (object dictionary entry 0x1018).

Note that only the VendorID field is mandatory. Any unsupported fields will be returned as zero.

Parameters

id	The identity object to be filled in by this call

Returns

A pointer to an error object, or NULL on success

5.87.3.6 GetMfgDeviceName()

```
virtual const Error* GetMfgDeviceName (
            int32 & len,
            char * str )  [inline], [virtual]
```

Read the manufacturer's device name string from the object dictionary.

Parameters

len	Holds the size of the buffer on entry, and the length of the downloaded data on return.
str	An array of characters used to upload the string.

Returns

An error object

5.87.3.7 GetMfgHardwareVer()

```
virtual const Error* GetMfgHardwareVer (
            int32 & len,
            char * str )  [inline], [virtual]
```

Read the manufacturer's Hardware version string from the object dictionary.

Parameters

len	Holds the size of the buffer on entry, and the length of the downloaded data on return.
str	An array of characters used to upload the string.

Returns

An error object

5.87.3.8 GetMfgSoftwareVer()

```
virtual const Error* GetMfgSoftwareVer (
            int32 & len,
            char * str )  [inline], [virtual]
```

Read the manufacturer's software version string from the object dictionary.

Parameters

len	Holds the size of the buffer on entry, and the length of the downloaded data on return.
str	An array of characters used to upload the string.

Returns

An error object

5.87.3.9 GetMfgStatus()

```
virtual const Error* GetMfgStatus (
            uint32 & stat )  [inline], [virtual]
```

Read the manufacturer status register from the object dictionary.

Parameters

stat	Reference to the int32 where the status will be returned

Returns

An error object

5.87.3.10 GetNetworkRef()

```
uint32 GetNetworkRef (
            void  )  [virtual]
```

Return a reference ID to the network that this node is attached to.

Returns

The reference ID or 0 if the node isn't attached to any network.

Reimplemented in Amp.

5.87.3.11 GetNetworkType()

```
NetworkType GetNetworkType (
            void  )  [virtual]
```

Return a value that identifies the type of network the node is currently attached to.

Returns

A network type value, or NET_TYPE_INVALID if the node isn't attached to any network.

5.87.3.12 GetNodeID()

```
virtual int16 GetNodeID (
            void  )  [inline], [virtual]
```

Return the node ID associated with this node.

Returns

The node ID

5.87.3.13 GetState()

```
virtual NodeState GetState (
            void  )  [inline], [virtual]
```

Returns the present state of this node.

Note that this requires node guarding or heartbeats to be enabled.

Returns

The present node state.

Reimplemented in Amp.

5.87.3.14 GetSynchId()

```
virtual const Error* GetSynchId (
            uint32 & id )  [inline], [virtual]
```

Return the COB-ID of the synch message.

Note that if this node is producing the synch message, bit 30 will be set.

Parameters

id	Where the COB-ID is returned

Returns

An error object.

5.87.3.15 GetSynchPeriod()

```
virtual const Error* GetSynchPeriod (
            uint32 & per )  [inline], [virtual]
```

Get the SYNC message interval in microseconds.

Parameters

per	Period will be returned here

Returns

An error object.

5.87.3.16 HandleEmergency()

```
virtual void HandleEmergency (
            CanFrame & frame )  [inline], [protected], [virtual]
```

Overload this function to handle emergency objects sent by this node.

Parameters

frame	Reference to the CAN frame holding the emergency data.

5.87.3.17 HandleStateChange()

```
virtual void HandleStateChange (
```

```
            NodeState from,
            NodeState to )  [inline], [protected], [virtual]
```

Overload this function to handle changes to the nodes state.

Note that the state member variable will have been changed to the new state before this function is called.

Parameters

from	Previous node state before the change
to	New node state

Reimplemented in Amp.

5.87.3.18 Init()

```
const Error * Init (
            Network & network,
            int16 nodeID )  [virtual]
```

Initialize the CANopen Node object.

Note that a CANopen node object must be initialized once and only once. This function should be used to initialize the object if it was created using the default constructor.

Parameters

network	The network object that this node is associated with.
nodeID	The node's ID. A value that identifies this node on the network.

Returns

A pointer to an error object, or NULL on success

Reimplemented in Amp, IOModule, and CopleyIO.

5.87.3.19 maxSdoFromNode()

```
int32 maxSdoFromNode (
            void  )  [virtual]
```

Return the maximum number of bytes that can be received in an SDO message.

For CANopen this is always 8 (the max size of a CAN frame). For EtherCAT it's the size of the mailbox buffer, and is node specific

Returns

The maximum number of bytes in an SDO receive message, or 0 on error

5.87.3.20 maxSdoToNode()

```
int32 maxSdoToNode (
            void  )  [virtual]
```

Return the maximum number of bytes that can be sent in an SDO message.

For CANopen this is always 8 (the max size of a CAN frame). For EtherCAT it's the size of the mailbox buffer, and is node specific

Returns

The maximum number of bytes in an SDO transmit message, or 0 on error

5.87.3.21 PdoDisable()

```
const Error * PdoDisable (
            uint16 n,
            PDO & pdo )  [virtual]
```

Disable the passed PDO object.

Parameters

n	The slot number of the PDO
pdo	The PDO mapped to that slot

Returns

An error object

5.87.3.22 PdoEnable()

```
const Error * PdoEnable (
            uint16 n,
            PDO & pdo )  [virtual]
```

Enable the passed PDO object.

Parameters

n	The slot number of the PDO
pdo	The PDO mapped to that slot

Returns

An error object

5.87.3.23 PdoSet()

```
const Error * PdoSet (
            uint16 slot,
            PDO & pdo,
            bool enable = true )  [virtual]
```

Associate the passed PDO object with this node.

The PDO will be setup as this node's nth PDO.

Parameters

slot	Which PDO slot to assign this PDO to.
pdo	The PDO object.
enable	If true, the PDO will be enabled after being setup (default). If false, the PDO will be setup but not enabled.

Returns

A pointer to an error object, or NULL on success

5.87.3.24 PreOpNode()

```
const Error * PreOpNode (
            void  )  [virtual]
```

Put this node in pre-operational state.

Returns

An error object or null on success.

5.87.3.25 ResetComm()

```
const Error * ResetComm (
            void  )  [virtual]
```

Reset this node's communications.

Returns

An error object or null on success.

5.87.3.26 ResetNode()

```
const Error * ResetNode (
            void  )  [virtual]
```

Reset this node.

Returns

An error object

5.87.3.27 RpdoDisable()

```
const Error * RpdoDisable (
            uint16 n )  [virtual]
```

Disable the specified receive PDO.

Parameters

n	The slot number of the PDO

Returns

An error object

5.87.3.28 SavePDOMapping()

```
const Error * SavePDOMapping (
            bool saveProfile )  [virtual]
```

Save the current PDO mapping to flash.

Parameters

saveProfile	If true, the device profile parameters will be saved. If false, the device profile parameters will not be saved.

Returns

An error object

5.87.3.29 SetSynchId()

```
virtual const Error* SetSynchId (
            uint32 id )  [inline], [virtual]
```

Set the COB-ID of the synch message.

If bit 30 of the ID is set, then this node will be the synch producer.

Parameters

id	COB-ID to set

Returns

An error object

5.87.3.30 SetSynchPeriod()

```
virtual const Error* SetSynchPeriod (
            uint32 per )  [inline], [virtual]
```

Set the SYNC message interval in microseconds.

Parameters

per	The period in microseconds.

Returns

An error object.

5.87.3.31 StartHeartbeat()

```
const Error * StartHeartbeat (
            uint16 period,
            uint16 timeout )  [virtual]
```

Enable heartbeat messages from this node, and start a thread to monitor them.

Parameters

period	The producer timeout value (milliseconds). The node will be configured to produce a heartbeat message at this interval.
timeout	The additional number of milliseconds that the monitor thread will wait before indicating an error. Thus, the consumer heartbeat interval will be (period + timeout).

Returns

A pointer to an error object, or NULL on success

5.87.3.32 StartNode()

```
const Error * StartNode (
            void  )  [virtual]
```

Start this node.

Returns

An error object or null on success.

5.87.3.33 StartNodeGuard()

```
const Error * StartNodeGuard (
            uint16 guardTime,
            byte lifeFactor )  [virtual]
```

Enable node guarding on this node.

When node guarding is enabled, a new thread is created which will send a remote request to this node every (guard↩ Time) milliseconds. The node must respond to this message within the guard time. If the node does not respond then the thread will notify the node of a state change.

Parameters

guardTime	The period in milliseconds of the guard messages sent to the node. It can range from 1 to 65535.
lifeFactor	A multiplier used by the node to determine how long to wait for a node guarding message from the host before indicating a local error. The nodes timeout (life time) is guardTime * lifeFactor. This parameter must be between 0 and 255. If it's zero, then life guarding on the node is disabled.

Returns

A pointer to an error object, or NULL on success

5.87.3.34 StopGuarding()

```
const Error * StopGuarding (
            void  )  [virtual]
```

Disable node guarding & heartbeat monitoring.

Returns

A pointer to an error object, or NULL on success

5.87.3.35 StopNode()

```
const Error * StopNode (
            void  )  [virtual]
```

Stop this node.

Returns

An error object or null on success.

5.87.3.36 SynchStart()

```
const Error * SynchStart (
            void  )  [virtual]
```

Start producing SYNC messages on this node.

Returns

An error object.

5.87.3.37 SynchStop()

```
const Error * SynchStop (
            void  )  [virtual]
```

Stop producing SYNC messages on this node.

Returns

An error object.

5.87.3.38 TpdoDisable()

```
const Error * TpdoDisable (
            uint16 n )  [virtual]
```

Disable the specified transmit PDO.

Parameters

n	The slot number of the PDO

Returns

An error object

5.87.3.39 UnInit()

```
const Error * UnInit (
            void  )  [virtual]
```

Un-initialize the Node object.

This puts the object back to it's default state.

Returns

A pointer to an error object, or NULL on success.

5.87.4 Member Data Documentation

5.87.4.1 sdo

```
SDO sdo
```

This SDO may be used to get/set values in the node's object dictionary.

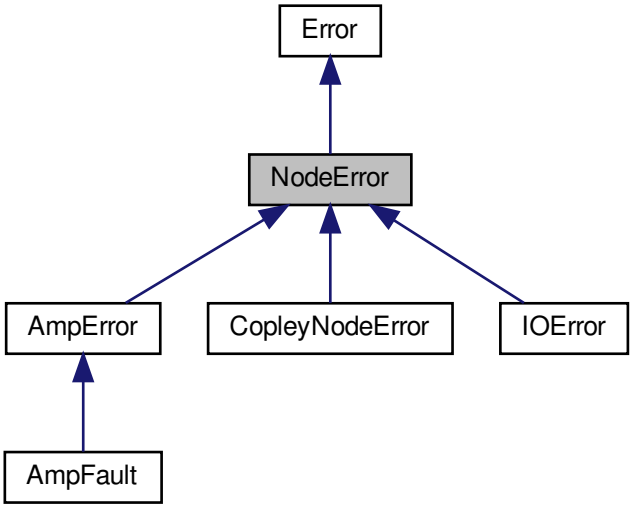
The documentation for this class was generated from the following files:

- CML_Node.h
- Node.cpp

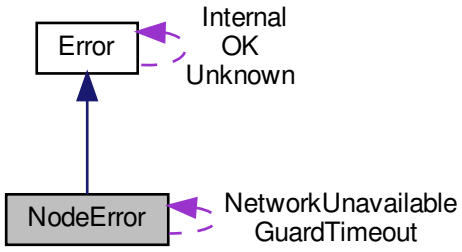
5.88 NodeError Class Reference

This class represents node errors.

Inheritance diagram for NodeError:



Collaboration diagram for NodeError:



Static Public Attributes

- static const NodeError GuardTimeout

  A node guarding or heartbeat timeout occurred.
- static const NodeError NetworkUnavailable

  The network this node is connected to has been deleted.

Protected Member Functions

- NodeError (uint16 id, const char *desc)
  Standard protected constructor.

Additional Inherited Members

5.88.1 Detailed Description

This class represents node errors.

There is one static member for each defined node error.

The documentation for this class was generated from the following file:

- CML_Node.h

5.89 NodeIdentity Struct Reference

CANopen identity object.

Public Attributes

- uint32 vendorID
  A unique vendor ID assigned by CiA (Can in Automation)
- uint32 productCode
  Manufacturer's product code.
- uint32 revision
  Revision number which identifies CANopen functionality.
- uint32 serial
  Product serial number.

5.89.1 Detailed Description

CANopen identity object.

Each node is required to include an identity object on it's object dictionary at location 0x1018. The only required parameter is the vendorID. All others are included at the manufacturer's discretion.

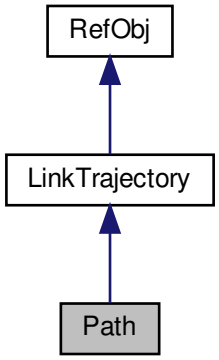
The documentation for this struct was generated from the following file:

- CML_Node.h

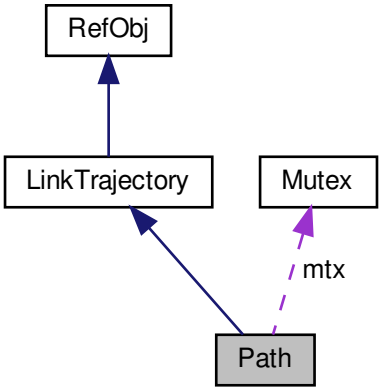
5.90 Path Class Reference

Multi-axis complex trajectory path.

Inheritance diagram for Path:



Collaboration diagram for Path:



Public Member Functions

- Path (uint d)

Path object constructor.

- virtual ~Path (void)

  Destructor for the path object.
- virtual void Reset (void)

  Reset the path to the first position.
- virtual const Error * SetStartPos (PointN &p)

  Set the initial position for the path.
- virtual const Error * SetVel (uunit v)

  Set the velocity limit for the current location.
- virtual const Error * SetAcc (uunit a)

  Set the acceleration limit for the current location.
- virtual const Error * SetDec (uunit d)

  Set the deceleration limit for the current location.
- virtual const Error * SetJrk (uunit j)

  Set the jerk limit for the current location.
- virtual const Error * AddLine (PointN &p)

  Add a line segment from the current position to the specified point.
- virtual const Error * AddLine (uunit length)

  Add a line segment of the specified length.
- virtual const Error * AddArc (double radius, double angle)

  Add an arc with the specified radius and angle (radians).
- virtual const Error * AddArc (PointN &center, double angle)

  Add an arc with the specified center point and angle (radians).
- virtual const Error * Pause (double sec)

  Set the current velocity to 0 and pause for the specified amount of time.
- virtual int GetDim (void)

  Get the dimension (i.e.
- virtual const Error * NextSegment (uunit pos[ ], uunit vel[ ], uint8 &time)

  Get the next trajectory segment.
- virtual const Error * StartNew (void)

  Start a new trajectory.
- bool PlayPath (double timeInc, double pos[ ], double vel[ ])

  Play back path data.

Additional Inherited Members

5.90.1 Detailed Description

Multi-axis complex trajectory path.

The object may be used to construct one or two dimensional trajectories built out of line segments and arcs.

5.90.2 Constructor & Destructor Documentation

5.90.2.1 Path()

```
Path (
            uint d )
```

Path object constructor.

The number of dimensions for the path must be passed. This object currently supports one and two dimensional (i.e. one and two axis) path construction.

Parameters

d	The number of dimensions for the path. Must be either one or two for now.

5.90.3 Member Function Documentation

5.90.3.1 AddArc() [1/2]

```
virtual const Error* AddArc (
            double radius,
            double angle )  [virtual]
```

Add an arc with the specified radius and angle (radians).

The arc will start at the current position and will move in either a clockwise (positive angle), or counter-clockwise (negative angle) direction.

Parameters

radius	The radius of the arc
angle	The number of radians to rotate through. Positive values will result in clockwise rotation.

Returns

An error object or null on success

5.90.3.2 AddArc() [2/2]

```
virtual const Error* AddArc (
            PointN & center,
            double angle )  [virtual]
```

Add an arc with the specified center point and angle (radians).

The arc will start at the current position and will move in clockwise (positive angle), or counter-clockwise (negative angle) direction.

Parameters

center	The center point of the arc.
angle	The number of radians to rotate through. Positive values will result in clockwise rotation.

Returns

An error object or null on success

5.90.3.3 AddLine() [1/2]

```
virtual const Error* AddLine (
            PointN & p )  [virtual]
```

Add a line segment from the current position to the specified point.

The direction of motion required to move from the current position to the given point will be compared to the direction of motion at the end of the last segment. If these directions change then the addition of this new point will require an abrupt change of direction. In this case, the initial velocity will be set to zero.

Parameters

p	The point to move to.

Returns

An error object or null on success

5.90.3.4 AddLine() [2/2]

```
virtual const Error* AddLine (
            uunit length )  [virtual]
```

Add a line segment of the specified length.

The direction of motion will remain the same as it was at the end of the last added segment. If this is the first segment added to the path, then the direction will be positive motion in the first axis.

Parameters

length	The length of the line segment to add.

Returns

An error object or null on success

5.90.3.5 GetDim()

```
virtual int GetDim (
            void  )  [virtual]
```

Get the dimension (i.e.

number of axes) of the path.

Returns

The path dimension

Implements LinkTrajectory.

5.90.3.6 NextSegment()

```
virtual const Error* NextSegment (
            uunit pos[],
            uunit vel[],
            uint8 & time )  [virtual]
```

Get the next trajectory segment.

This method is called by the Linkage object when as it passes the trajectory informatoin up to the amplifiers.

Parameters

pos	An array where the position values will be returned. This array will be at least D elements long, where D is the trajectory dimension as returned by LinkTrajectory::GetDim()
vel	An array where the velocity values will be returned.
time	The segment time is returned here. This is in milliseconds and ranges from 1 to 255. If zero is returned, this is the last frame in the profile.

Returns

A pointer to an error object on failure, or NULL on success.

Implements LinkTrajectory.

5.90.3.7 Pause()

```
virtual const Error* Pause (
            double sec )  [virtual]
```

Set the current velocity to 0 and pause for the specified amount of time.

Parameters

sec	The time to pause (must be >= 0). Time is specified in seconds.

Returns

An error object or null on success

5.90.3.8 PlayPath()

```
bool PlayPath (
            double timeInc,
            double pos[],
            double vel[] )
```

Play back path data.

This method may be used to itterate through a path for display purposes.

Before starting a path playback, the path should be reset using the method Path::Reset.

Each call to this function will return position and velocity information for the current playback position in the path. It will then increment the playback position by the time value passed. When the end of the path is reached, the method will return true.

Parameters

timeInc	The amount of time (seconds) to increment the playback position after reading out the position & velocity values.
pos	An array where the position information will be returned. This array must be long enough to store DIM elements, where DIM is the path dimension.
vel	An array where the velocity information will be returned. This array must be long enough to store DIM elements, where DIM is the path dimension.

Returns

true if the end of the path has been reached, false if not.

5.90.3.9 Reset()

```
virtual void Reset (
            void  )  [virtual]
```

Reset the path to the first position.

This should be called before the path is passed to the Linkage object as a trajectory to run.

5.90.3.10 SetAcc()

```
virtual const Error* SetAcc (
            uunit a )  [virtual]
```

Set the acceleration limit for the current location.

Acceleration limits must be greater then zero.

Parameters

a	The maximum acceleration (position units / second / second)

Returns

An error object or null on success

5.90.3.11 SetDec()

```
virtual const Error* SetDec (
            uunit d )  [virtual]
```

Set the deceleration limit for the current location.

Note that setting the deceleration limit less then or equal to zero will cause the acceleration value to be used for deceleration also.

Parameters

d	The maximum deceleration (position units / second / second)

Returns

An error object or null on success

5.90.3.12 SetJrk()

```
virtual const Error* SetJrk (
            uunit j )  [virtual]
```

Set the jerk limit for the current location.

Note that setting the jerk limit to a value less then or equal to zero will cause the path to be calculated with no jerk limiting.

Parameters

j	The jerk limit (position units / second / second / second)

Returns

An error object or null on success.

5.90.3.13 SetStartPos()

```
virtual const Error* SetStartPos (
            PointN & p )  [virtual]
```

Set the initial position for the path.

This method may be used to start a path at a position other then (0,0) which is the default if no staring position is set.

The starting position may be set at any time, either before or after adding segments to the path. Internally, the segments are all stored as relative positions.

Parameters

p	The starting position for this path.

Returns

An error object or null on success

5.90.3.14 SetVel()

```
virtual const Error* SetVel (
            uunit v )  [virtual]
```

Set the velocity limit for the current location.

Velocity limits must be greater then zero.

Parameters

v	The maximum velocity (position units / second)

Returns

An error object or null on success

5.90.3.15 StartNew()

```
virtual const Error* StartNew (
            void  )  [virtual]
```

Start a new trajectory.

This function is called before the first call to LinkTrajectory::NextSegment. It will result in a call to Path::Reset

Returns

An error pointer if the trajectory object is not available, or NULL if it is ready to be sent.

Reimplemented from LinkTrajectory.

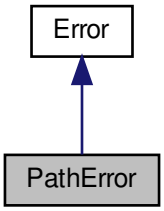
The documentation for this class was generated from the following file:

- CML_Path.h

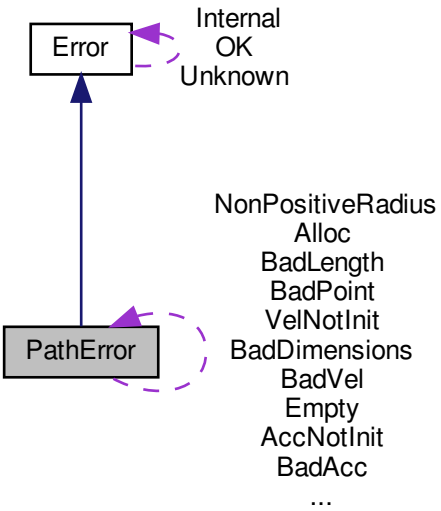
5.91 PathError Class Reference

This class represents errors returned by the path Path object.

Inheritance diagram for PathError:



Collaboration diagram for PathError:



Static Public Attributes

- static const PathError BadVel

Illegal velocity value.

- static const PathError BadAcc

  Illegal acceleration value.

- static const PathError VelNotInit

  Velocity limit not yet set.

- static const PathError AccNotInit

  Acceleration limit not yet set.

- static const PathError BadPoint

  The passed point doesn't match the path.

- static const PathError Alloc

  Unable to allocate memory for path.

- static const PathError BadLength

  An illegal negative length value was passed.

- static const PathError Empty

  Attempt to execute an empty path.

- static const PathError BadDimensions

  Attempt to use a point with bad dimensions.

- static const PathError BadCenterPoint

  The center point must result in a positive radius.

- static const PathError NonPositiveRadius

  The user entered a radius that is not a positive value.

Protected Member Functions

- PathError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.91.1 Detailed Description

This class represents errors returned by the path Path object.

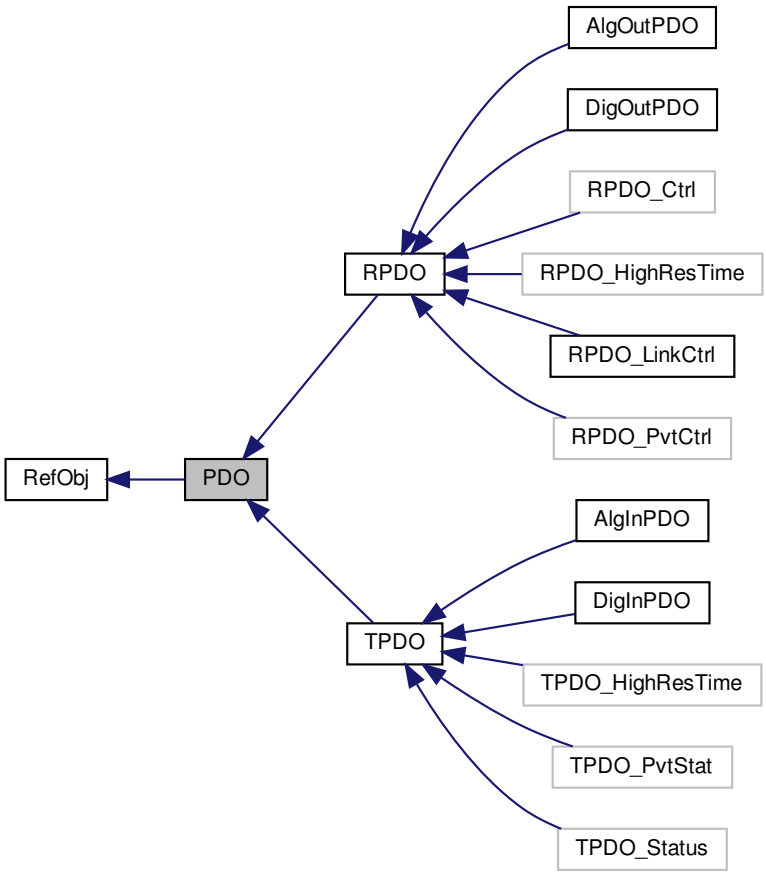
The documentation for this class was generated from the following file:

- CML_Path.h

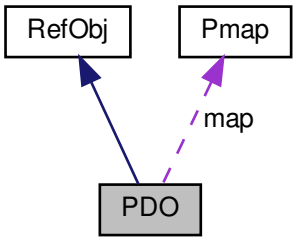
5.92 PDO Class Reference

PDO (Process Data Object) base class.

Inheritance diagram for PDO:



Collaboration diagram for PDO:



Public Member Functions

- PDO (void)

  Default constructor. Simply initializes some variables.
- virtual ~PDO ()

  Virtual destructor.
- virtual bool IsTxPDO (void)=0

  Return true for transmit PDOs and false for receive PDOs.
- virtual const Error * SetID (uint32 i)

  Set the CAN message ID associated with the PDO.
- virtual uint32 GetID (void)

  Get the CAN message ID associated with the PDO.
- virtual const Error * SetType (byte t)

  Set the PDO transmission type code.
- virtual byte GetType (void)

  Return the PDO transmission type associated with this PDO.
- virtual const Error * ClearMap (void)

  Clear the variable map associated with this PDO.
- virtual const Error * AddVar (Pmap &var)

  Add the passed variable to the end of the variable map associated with this PDO.
- virtual int GetRtrOk (void)

  Return non-zero if RTR requests are OK for this PDO.
- virtual int GetMapCodes (uint32 codes[ ])

  Fill the array of 32-bit ints with the PDO mapping codes used by this PDO.
- virtual int GetBitCt (void)

  Return the length (in bits) of the data mapped to this PDO.

Protected Attributes

- byte flags

  Misc flags associated with the PDO.
- byte type

  Transmission type code.
- int mapCt

  Number of elements in the variable map.
- int bitCt

  Number of bits mapped so far.
- Pmap * map [PDO_MAP_LEN]

  Array of pointers to Pmap objects that describe the variables transmitted by this PDO.
- uint32 id

  The CAN message ID associated with this PDO.

Friends

- class **EtherCAT**

Additional Inherited Members

5.92.1 Detailed Description

PDO (Process Data Object) base class.

5.92.2 Member Function Documentation

5.92.2.1 AddVar()

```
const Error * AddVar (
            Pmap & var )  [virtual]
```

Add the passed variable to the end of the variable map associated with this PDO.

Parameters

var	The variable to be added.

Returns

An error object.

5.92.2.2 ClearMap()

```
virtual const Error* ClearMap (
            void  )  [inline], [virtual]
```

Clear the variable map associated with this PDO.

Returns

An error code

5.92.2.3 GetID()

```
virtual uint32 GetID (
            void  )  [inline], [virtual]
```

Get the CAN message ID associated with the PDO.

Returns

The COB-ID of the PDO

5.92.2.4 GetMapCodes()

```
virtual int GetMapCodes (
            uint32 codes[] )  [inline], [virtual]
```

Fill the array of 32-bit ints with the PDO mapping codes used by this PDO.

Parameters

codes	An array of at least PDO_MAP_LEN 32-bit ints. The mapping codes will be stored here

Returns

The number of elements mapped into this PDO.

5.92.2.5 GetRtrOk()

```
virtual int GetRtrOk (
            void  )  [inline], [virtual]
```

Return non-zero if RTR requests are OK for this PDO.

Note that this only really makes sense for transmit PDOs

Returns

zero if RTR not allowed, non-zero if allowed.

5.92.2.6 GetType()

```
virtual byte GetType (
            void  )  [inline], [virtual]
```

Return the PDO transmission type associated with this PDO.

Returns

The 8-bit type code

5.92.2.7 SetID()

```
virtual const Error* SetID (
            uint32 i )  [inline], [virtual]
```

Set the CAN message ID associated with the PDO.

Parameters

i	The ID value.

Returns

An error object or NULL on success

5.92.2.8 SetType()

```
virtual const Error* SetType (
            byte t )  [inline], [virtual]
```

Set the PDO transmission type code.

Parameters

t	Transmission type code

Returns

An error object.

5.92.3 Member Data Documentation

5.92.3.1 map

```
Pmap* map[PDO_MAP_LEN]  [protected]
```

Array of pointers to Pmap objects that describe the variables transmitted by this PDO.

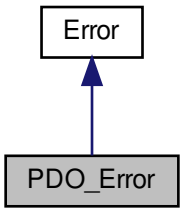
The documentation for this class was generated from the following files:

- CML_PDO.h
- PDO.cpp

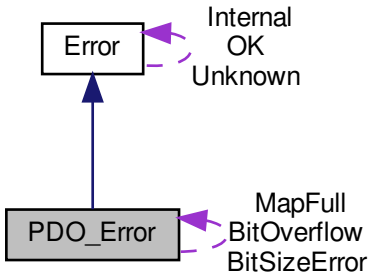
5.93 PDO_Error Class Reference

This class represents error conditions related to PDOs.

Inheritance diagram for PDO_Error:



Collaboration diagram for PDO_Error:



Static Public Attributes

- static const PDO_Error MapFull

  The variable map associated with the PDO is already full.
- static const PDO_Error BitOverflow

  Adding the variable to the map would cause the map to be too long (more then 64 bits).
- static const PDO_Error BitSizeError

  PDO Map variables of the passed bit size are not presently supported.

Protected Member Functions

- PDO_Error (uint16 id, const char *desc)
  Standard protected constructor.

Additional Inherited Members

5.93.1 Detailed Description

This class represents error conditions related to PDOs.

5.93.2 Member Data Documentation

5.93.2.1 BitOverflow

```
const PDO_Error BitOverflow  [static]
```

Adding the variable to the map would cause the map to be too long (more then 64 bits).

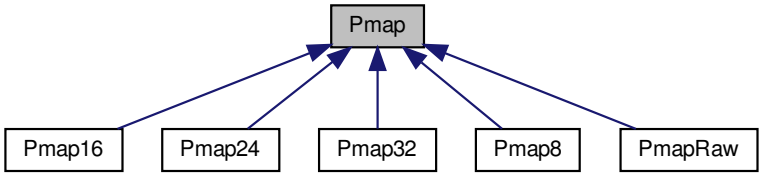
The documentation for this class was generated from the following file:

- CML_PDO.h

5.94 Pmap Class Reference

This class allows variables to be mapped into a PDO.

Inheritance diagram for Pmap:



Public Member Functions

- Pmap ()

  Default constructor for a generic PDO mapping variable.
- Pmap (uint16 index, byte sub, byte bits)

  Construct a generic PDO mapping variable and initialize it's size and object ID.
- virtual ~Pmap ()

  Virtual destructor.
- virtual const Error * Init (uint16 index, byte sub, byte bits)

  Initialize a generic PDO mapping variable.
- virtual void Get (byte *cptr)

  Called when a receive PDO is about to be transmitted.
- virtual void Set (byte *cptr)

  Called when a transmit PDO is received.
- virtual uint32 GetMapCode ()

  Return the 32-bit code used to identify this variable in the CANopen node's PDO mapping block.
- uint16 GetIndex ()

  Get the object index associated with this variable.
- byte GetSub ()

  Get the object sub-index associated with this variable.
- byte GetBits ()

  Get the number of bits in this variable.

Protected Attributes

- uint16 index

  The 16-bit index of the object in the object dictionary.
- byte sub

  The 8-bit sub-index of the object in the object dictionary.
- byte bits

  The number of bits that this object takes up.

5.94.1 Detailed Description

This class allows variables to be mapped into a PDO.

This class can be used directly for transmit PDOs if the received data is not of interest (it will simply be discarded by the Set() function. Using this for receive PDOs is not recommended since the Get() function doesn't add any data to the output stream and therefore the data transmitted to the node will be undefined.

5.94.2 Constructor & Destructor Documentation

5.94.2.1 Pmap()

```
Pmap (
            uint16 index,
            byte sub,
            byte bits )  [inline]
```

Construct a generic PDO mapping variable and initialize it's size and object ID.

Parameters

index	The index of the variable in the object dictionary
sub	The variable's sub-index in the object dictionary
bits	The size of the variable in bits

5.94.3 Member Function Documentation

5.94.3.1 Get()

```
virtual void Get (
            byte * cptr )  [inline], [virtual]
```

Called when a receive PDO is about to be transmitted.

This virtual function does nothing and therefore objects of this generic type shouldn't be used when actually transmitting PDOs

Parameters

cptr	Pointer where the PDO data should be stored

Reimplemented in Pmap8, Pmap16, Pmap24, Pmap32, and PmapRaw.

5.94.3.2 GetBits()

```
byte GetBits ( )  [inline]
```

Get the number of bits in this variable.

Returns

The number of bits.

5.94.3.3 GetIndex()

```
uint16 GetIndex ( )  [inline]
```

Get the object index associated with this variable.

Returns

The 16-bit object index.

5.94.3.4 GetSub()

```
byte GetSub ( )  [inline]
```

Get the object sub-index associated with this variable.

Returns

The 8-bit object sub-index.

5.94.3.5 Init()

```
virtual const Error* Init (
            uint16 index,
            byte sub,
            byte bits )  [inline], [virtual]
```

Initialize a generic PDO mapping variable.

Parameters

index	The index of the variable in the object dictionary
sub	The variable's sub-index in the object dictionary
bits	The size of the variable in bits

Returns

An error object

5.94.3.6 Set()

```
virtual void Set (
            byte * cptr )  [inline], [virtual]
```

Called when a transmit PDO is received.

This virtual function doesn't do anything and therefore objects of this base class should only be used for variables that are not of interest and can therefore be ignored.

Parameters

cptr	Pointer to the received PDO data.

Reimplemented in Pmap8, Pmap16, Pmap24, Pmap32, and PmapRaw.

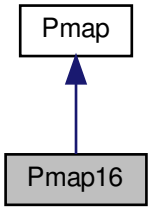
The documentation for this class was generated from the following file:

- CML_PDO.h

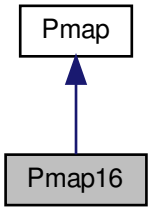
5.95 Pmap16 Class Reference

This is a PDO variable mapping class that extends the virtual Pmap class to handle 16-bit integers.

Inheritance diagram for Pmap16:



Collaboration diagram for Pmap16:



Public Member Functions

- Pmap16 ()
  Default constructor for a 16-bit mapping object.
- Pmap16 (uint16 index, byte sub=0)
  Create a new 16-bit mapping object.
- const Error * Init (uint16 index, byte sub=0)
  Initialize a 16-bit mapping object.
- virtual void Get (byte *cptr)
  Copy the current value of this variable into the passed character array.
- virtual void Set (byte *cptr)
  Update the value of this variable based on the data passed in a character array.
- virtual int16 Read (void)
  Read the current value of this variable.
- virtual void Write (int16 d)
  Write a new value to this variable.

Additional Inherited Members

5.95.1 Detailed Description

This is a PDO variable mapping class that extends the virtual Pmap class to handle 16-bit integers.

5.95.2 Constructor & Destructor Documentation

5.95.2.1 Pmap16()

```
Pmap16 (
            uint16 index,
            byte sub = 0 )  [inline]
```

Create a new 16-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

5.95.3 Member Function Documentation

5.95.3.1 Get()

```
virtual void Get (
            byte * cptr )  [inline], [virtual]
```

Copy the current value of this variable into the passed character array.

This function is called when a receive PDO is about to be transmitted to a node.

Parameters

cptr	A character pointer that references a char array of at least 2 bytes. The current value of this variable will be copied there.

Reimplemented from Pmap.

5.95.3.2 Init()

```
const Error* Init (
            uint16 index,
            byte sub = 0 )  [inline]
```

Initialize a 16-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

Returns

An error object

5.95.3.3 Read()

```
virtual int16 Read (
            void  )  [inline], [virtual]
```

Read the current value of this variable.

Returns

The current value of this variable.

5.95.3.4 Set()

```
virtual void Set (
            byte * cptr )  [inline], [virtual]
```

Update the value of this variable based on the data passed in a character array.

This function is called when a transmit PDO that this variable is mapped to is received.

Parameters

cptr	A character pointer that references a char array of at least 2 bytes. The value of this variable will be updated with the data passed in this array.

Reimplemented from Pmap.

5.95.3.5 Write()

```
virtual void Write (
            int16 d )  [inline], [virtual]
```

Write a new value to this variable.

Parameters

d	The new value to write.

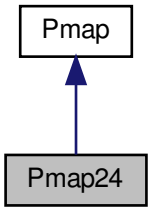
The documentation for this class was generated from the following file:

- CML_PDO.h

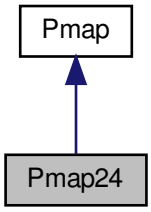
5.96 Pmap24 Class Reference

This is a PDO variable mapping class that extends the virtual Pmap class to handle 24-bit integers.

Inheritance diagram for Pmap24:



Collaboration diagram for Pmap24:



Public Member Functions

- Pmap24 ()

  Default constructor for a 24-bit mapping object.
- Pmap24 (uint16 index, byte sub=0)

  Create a new 24-bit mapping object.
- const Error * Init (uint16 index, byte sub=0)

  Initialize a 24-bit mapping object.
- virtual void Get (byte *cptr)

  Copy the current value of this variable into the passed character array.
- virtual void Set (byte *cptr)

Update the value of this variable based on the data passed in a character array.

- virtual int32 Read (void)

  Read the current value of this variable.

- virtual void Write (int32 d)

  Write a new value to this variable.

Additional Inherited Members

5.96.1 Detailed Description

This is a PDO variable mapping class that extends the virtual Pmap class to handle 24-bit integers.

5.96.2 Constructor & Destructor Documentation

5.96.2.1 Pmap24()

```
Pmap24 (
            uint16 index,
            byte sub = 0 )  [inline]
```

Create a new 24-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

5.96.3 Member Function Documentation

5.96.3.1 Get()

```
virtual void Get (
            byte * cptr )  [inline], [virtual]
```

Copy the current value of this variable into the passed character array.

This function is called when a receive PDO is about to be transmitted to a node.

Parameters

cptr	A character pointer that references a char array of at least 4 bytes. The current value of this variable will be copied there.

Reimplemented from Pmap.

5.96.3.2 Init()

```
const Error* Init (
             uint16 index,
             byte sub = 0 )  [inline]
```

Initialize a 24-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

Returns

An error object

5.96.3.3 Read()

```
virtual int32 Read (
             void  )  [inline], [virtual]
```

Read the current value of this variable.

Returns

The current value of this variable.

5.96.3.4 Set()

```
virtual void Set (
             byte * cptr )  [inline], [virtual]
```

Update the value of this variable based on the data passed in a character array.

This function is called when a transmit PDO that this variable is mapped to is received.

Parameters

cptr	A character pointer that references a char array of at least 4 bytes. The value of this variable will be updated with the data passed in this array.

Reimplemented from Pmap.

5.96.3.5 Write()

```
virtual void Write (
            int32 d )  [inline], [virtual]
```

Write a new value to this variable.

Parameters

d	The new value to write.

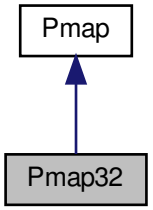
The documentation for this class was generated from the following file:

- CML_PDO.h

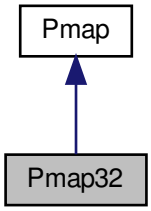
5.97 Pmap32 Class Reference

This is a PDO variable mapping class that extends the virtual Pmap class to handle 32-bit integers.

Inheritance diagram for Pmap32:



Collaboration diagram for Pmap32:



Public Member Functions

- Pmap32 ()
  Default constructor for a 32-bit mapping object.
- Pmap32 (uint16 index, byte sub=0)
  Create a new 32-bit mapping object.
- const Error * Init (uint16 index, byte sub=0)
  Initialize a 32-bit mapping object.
- virtual void Get (byte *cptr)
  Copy the current value of this variable into the passed character array.
- virtual void Set (byte *cptr)
  Update the value of this variable based on the data passed in a character array.
- virtual int32 Read (void)
  Read the current value of this variable.
- virtual void Write (int32 d)
  Write a new value to this variable.

Additional Inherited Members

5.97.1 Detailed Description

This is a PDO variable mapping class that extends the virtual Pmap class to handle 32-bit integers.

5.97.2 Constructor & Destructor Documentation

5.97.2.1 Pmap32()

```
Pmap32 (
            uint16 index,
            byte sub = 0 )  [inline]
```

Create a new 32-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

5.97.3 Member Function Documentation

5.97.3.1 Get()

```
virtual void Get (
            byte * cptr )  [inline], [virtual]
```

Copy the current value of this variable into the passed character array.

This function is called when a receive PDO is about to be transmitted to a node.

Parameters

cptr	A character pointer that references a char array of at least 4 bytes. The current value of this variable will be copied there.

Reimplemented from Pmap.

5.97.3.2 Init()

```
const Error* Init (
            uint16 index,
            byte sub = 0 )  [inline]
```

Initialize a 32-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

Returns

An error object

5.97.3.3 Read()

```
virtual int32 Read (
            void  )  [inline], [virtual]
```

Read the current value of this variable.

Returns

The current value of this variable.

5.97.3.4 Set()

```
virtual void Set (
            byte * cptr )  [inline], [virtual]
```

Update the value of this variable based on the data passed in a character array.

This function is called when a transmit PDO that this variable is mapped to is received.

Parameters

cptr	A character pointer that references a char array of at least 4 bytes. The value of this variable will be updated with the data passed in this array.

Reimplemented from Pmap.

5.97.3.5 Write()

```
virtual void Write (
            int32 d )  [inline], [virtual]
```

Write a new value to this variable.

Parameters

d	The new value to write.

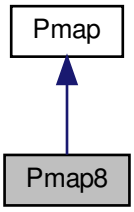
The documentation for this class was generated from the following file:

- CML_PDO.h

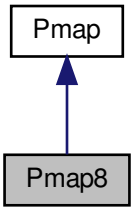
5.98 Pmap8 Class Reference

This is a PDO variable mapping class that extends the virtual Pmap class to handle 8-bit integers.

Inheritance diagram for Pmap8:



Collaboration diagram for Pmap8:



Public Member Functions

- Pmap8 ()

  Default constructor for a 8-bit mapping object.
- Pmap8 (uint16 index, byte sub=0)

  Create a new 8-bit mapping object.
- const Error * Init (uint16 index, byte sub=0)

  Initialize a 8-bit mapping object.
- virtual void Get (byte *cptr)

  Copy the current value of this variable into the passed character array.
- virtual void Set (byte *cptr)

Update the value of this variable based on the data passed in a character array.

- virtual byte Read (void)

  Read the current value of this variable.

- virtual void Write (byte d)

  Write a new value to this variable.

Additional Inherited Members

5.98.1 Detailed Description

This is a PDO variable mapping class that extends the virtual Pmap class to handle 8-bit integers.

5.98.2 Constructor & Destructor Documentation

5.98.2.1 Pmap8()

```
Pmap8 (
            uint16 index,
            byte sub = 0 )  [inline]
```

Create a new 8-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

5.98.3 Member Function Documentation

5.98.3.1 Get()

```
virtual void Get (
            byte * cptr )  [inline], [virtual]
```

Copy the current value of this variable into the passed character array.

This function is called when a receive PDO is about to be transmitted to a node.

Parameters

cptr	A character pointer that references a char array of at least 1 bytes. The current value of this variable will be copied there.

Reimplemented from Pmap.

5.98.3.2 Init()

```
const Error* Init (
             uint16 index,
             byte sub = 0 )  [inline]
```

Initialize a 8-bit mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index (defaults to 0)

Returns

An error object

5.98.3.3 Read()

```
virtual byte Read (
             void  )  [inline], [virtual]
```

Read the current value of this variable.

Returns

The current value of this variable.

5.98.3.4 Set()

```
virtual void Set (
             byte * cptr )  [inline], [virtual]
```

Update the value of this variable based on the data passed in a character array.

This function is called when a transmit PDO that this variable is mapped to is received.

Parameters

cptr	A character pointer that references a char array of at least 1 bytes. The value of this variable will be updated with the data passed in this array.

Reimplemented from Pmap.

5.98.3.5 Write()

```
virtual void Write (
            byte d )  [inline], [virtual]
```

Write a new value to this variable.

Parameters

d	The new value to write.

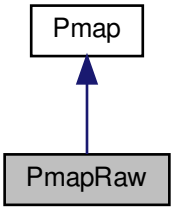
The documentation for this class was generated from the following file:

- CML_PDO.h

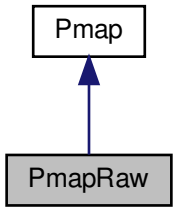
5.99 PmapRaw Class Reference

This is the most generic PDO variable mapping class.

Inheritance diagram for PmapRaw:



Collaboration diagram for PmapRaw:



Public Member Functions

- PmapRaw ()

  Default constructor.
- PmapRaw (uint16 index, byte sub, byte bits)

  Create a new mapping object.
- virtual void Get (byte *cptr)

  Copy the current value of this variable into the passed character array.
- virtual void Set (byte *cptr)

  Update the value of this variable based on the data passed in a character array.

Additional Inherited Members

5.99.1 Detailed Description

This is the most generic PDO variable mapping class.

It doesn't do any special formatting, just holds up to 8 bytes of raw data.

5.99.2 Constructor & Destructor Documentation

5.99.2.1 PmapRaw()

```
PmapRaw (
            uint16 index,
            byte sub,
            byte bits )  [inline]
```

Create a new mapping object.

Parameters

index	Object index associated with this variable
sub	Object sub-index
bits	The size of the variable in bits

5.99.3 Member Function Documentation

5.99.3.1 Get()

```
virtual void Get (
            byte * cptr )  [inline], [virtual]
```

Copy the current value of this variable into the passed character array.

This function is called when a receive PDO is about to be transmitted to a node.

Parameters

cptr	A character pointer that references a char array large enough to hold this objects data.

Reimplemented from Pmap.

5.99.3.2 Set()

```
virtual void Set (
            byte * cptr )  [inline], [virtual]
```

Update the value of this variable based on the data passed in a character array.

This function is called when a transmit PDO that this variable is mapped to is received.

Parameters

cptr	A pointer to the received data.

Reimplemented from Pmap.

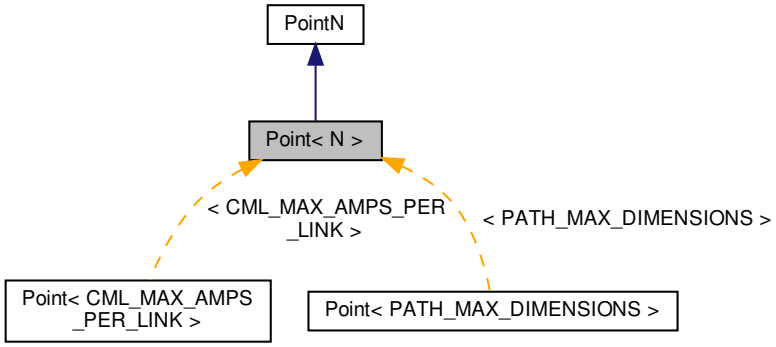
The documentation for this class was generated from the following file:

- CML_PDO.h

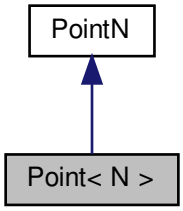
5.100 Point< N > Class Template Reference

Template used for N dimensional objects.

Inheritance diagram for Point< N >:



Collaboration diagram for Point< N >:



Public Member Functions

- int getDim (void) const

  Get the number of dimensions of this point.
- int getMax (void) const

  Get the max dimensions that this point can handle.
- void setDim (int d)

  Set the number of dimensions of this point.

5.100.1 Detailed Description

template<int N>
class Point< N >

Template used for N dimensional objects.

This template may be used to generate point objects for some fixed number of dimensions.

5.100.2 Member Function Documentation

5.100.2.1 getDim()

```
int getDim (
            void  ) const  [inline], [virtual]
```

Get the number of dimensions of this point.

Returns

The point dimension

Implements PointN.

5.100.2.2 getMax()

```
int getMax (
            void  ) const  [inline], [virtual]
```

Get the max dimensions that this point can handle.

Returns

The max value.

Implements PointN.

5.100.2.3 setDim()

```
void setDim (
            int d )  [inline], [virtual]
```

Set the number of dimensions of this point.

Parameters

d	The new point dimension

Implements PointN.

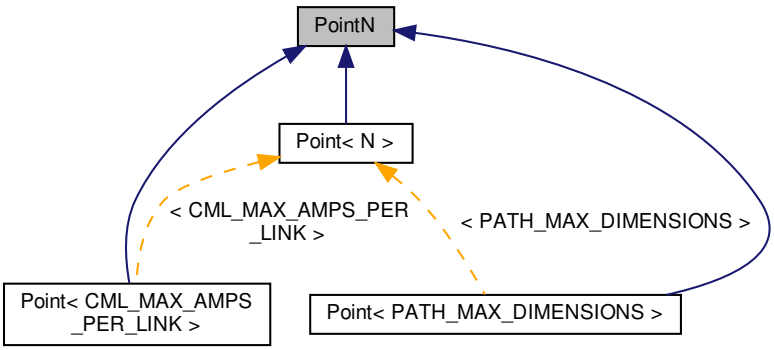
The documentation for this class was generated from the following file:

- CML_Geometry.h

5.101 PointN Class Reference

An N axis point.

Inheritance diagram for PointN:



Public Member Functions

- virtual ~PointN ()

  Virtual destructor.
- virtual int getDim (void) const =0

  Get the number of dimensions of this point.
- virtual void setDim (int d)=0

  Set the number of dimensions of this point.
- virtual int getMax (void) const =0

  Get the max dimensions that this point can handle.

5.101.1 Detailed Description

An N axis point.

This point specifies a position in N dimensions.

This is a pure virtual base class of the more specific Point classes.

5.101.2 Member Function Documentation

5.101.2.1 getDim()

```
virtual int getDim (
            void  ) const  [pure virtual]
```

Get the number of dimensions of this point.

Returns

The point dimension

Implemented in Point< N >, Point< CML_MAX_AMPS_PER_LINK >, and Point< PATH_MAX_DIMENSIONS >.

5.101.2.2 getMax()

```
virtual int getMax (
            void  ) const  [pure virtual]
```

Get the max dimensions that this point can handle.

Returns

The max value.

Implemented in Point< N >, Point< CML_MAX_AMPS_PER_LINK >, and Point< PATH_MAX_DIMENSIONS >.

5.101.2.3 setDim()

```
virtual void setDim (
            int d )  [pure virtual]
```

Set the number of dimensions of this point.

Parameters

d	The new point dimension

Implemented in Point< N >, Point< CML_MAX_AMPS_PER_LINK >, and Point< PATH_MAX_DIMENSIONS >.

The documentation for this class was generated from the following files:

- CML_Geometry.h
- Geometry.cpp

5.102 PosLoopConfig Struct Reference

This structure holds the position loop configuration parameters specific to the Copley amplifier.

Public Member Functions

- PosLoopConfig (void)
  Default constructor.

Public Attributes

- int16 kp
  Proportional gain.
- int16 kvff
  Velocity feed forward.
- int16 kaff
  Acceleration feed forward.
- int16 ki
  Integral gain.
- int16 kd
  Derivative gain.
- int16 scale
  Scaling factor.
- int16 xKd
  cross coupling derivative gain
- int16 xKi
  cross coupling integral gain
- int16 xKp
  cross coupling proportional gain
- int16 kiDrain
  Position loop drain value for integral sum. Set to 0 to disable.

5.102.1 Detailed Description

This structure holds the position loop configuration parameters specific to the Copley amplifier.

The position loop is one of three servo control loops used by the amplifier to control a motor. The configuration parameters used by this control loop allow the servo performance to be 'tuned' for various motors and loads.

The amplifier member functions Amp::GetPosLoopConfig and Amp::SetPosLoopConfig are used to read and write this data to the amplifier.

5.102.2 Constructor & Destructor Documentation

5.102.2.1 PosLoopConfig()

```
PosLoopConfig (
            void  )  [inline]
```

Default constructor.

Simply initializes all servo parameters to zero.

5.102.3 Member Data Documentation

5.102.3.1 scale

```
int16 scale
```

Scaling factor.

This is a multiplier that is applied to the output of the position loop. It's scaled up by 100, so setting the scaling factor to 1234 would multiply the output of the loop by 12.34. This parameter was added in firmware version 3.30. For any earlier version it will default to 100 (scale by 1.0).

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.103 ProfileConfig Struct Reference

Amplifier profile parameters.

Public Member Functions

- ProfileConfig (void)

  Default constructor. Simply set all parameters to zero.

Public Attributes

- PROFILE_TYPE type

  Type of profile to be used.
- uunit pos

  For absolute moves this is an absolute position, for relative moves it's a distance to move.
- uunit vel

  Velocity limit for move.
- uunit acc

  Acceleration limit for move.
- uunit dec

  Deceleration limit for move.
- uunit abort

  Acceleration value to use when aborting a running trajectory.
- uunit jrk

  Jerk limit for move.

5.103.1 Detailed Description

Amplifier profile parameters.

This structure holds all the parameters related to point-to-point profile moves.

5.103.2 Member Data Documentation

5.103.2.1 abort

```
uunit abort
```

Acceleration value to use when aborting a running trajectory.

This is the same as the 'quick stop' deceleration.

5.103.2.2 pos

```
uunit pos
```

For absolute moves this is an absolute position, for relative moves it's a distance to move.

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.104 ProfileConfigScurve Struct Reference

S-curve profile parameters.

Public Member Functions

- ProfileConfigScurve (void)
  Default constructor. Simply set all parameters to zero.

Public Attributes

- uunit pos
  For absolute moves this is an absolute position, for relative moves it's a distance to move.
- uunit vel
  Velocity limit for move.
- uunit acc
  Acceleration limit for move.
- uunit jrk
  Jerk limit for move.

5.104.1 Detailed Description

S-curve profile parameters.

This structure holds all the parameters necessary to perform a s-curve (jerk limited) profile move.

5.104.2 Member Data Documentation

5.104.2.1 acc

`uunit acc`

Acceleration limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.104.2.2 jrk

`uunit jrk`

Jerk limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.104.2.3 pos

`uunit pos`

For absolute moves this is an absolute position, for relative moves it's a distance to move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.104.2.4 vel

`uunit vel`

Velocity limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

The documentation for this struct was generated from the following file:

- CML_Amp.h

5.105 ProfileConfigTrap Struct Reference

Trapezoidal profile parameters.

Public Member Functions

- ProfileConfigTrap (void)
  Default constructor. Simply set all parameters to zero.

Public Attributes

- uunit pos

  For absolute moves this is an absolute position, for relative moves it's a distance to move.
- uunit vel

  Velocity limit for move.
- uunit acc

  Acceleration limit for move.
- uunit dec

  Deceleration limit for move.

5.105.1 Detailed Description

Trapezoidal profile parameters.

This structure holds all the parameters necessary to perform a trapezoidal profile move.

5.105.2 Member Data Documentation

5.105.2.1 acc

```
uunit acc
```

Acceleration limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.105.2.2 dec

```
uunit dec
```

Deceleration limit for move.

Note that if this parameter is not set, then the acceleration value will be used for deceleration.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.105.2.3 pos

```
uunit pos
```

For absolute moves this is an absolute position, for relative moves it's a distance to move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.105.2.4 vel

```
uunit vel
```

Velocity limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

The documentation for this struct was generated from the following file:

- CML_Amp.h

5.106 ProfileConfigVel Struct Reference

Velocity profile parameters.

Public Member Functions

- ProfileConfigVel (void)
  Default constructor. Simply set all parameters to zero.

Public Attributes

- uunit dir
  Direction of motion.
- uunit vel
  Velocity limit for move.
- uunit acc
  Acceleration limit for move.
- uunit dec
  Deceleration limit for move.

5.106.1 Detailed Description

Velocity profile parameters.

This structure holds all the parameters necessary to perform a velocity profile move.

5.106.2 Member Data Documentation

5.106.2.1 acc

`uunit acc`

Acceleration limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.106.2.2 dec

`uunit dec`

Deceleration limit for move.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.106.2.3 dir

`uunit dir`

Direction of motion.

If >= 0, then move in the positive direction If < 0, then move in the negative direction.

5.106.2.4 vel

`uunit vel`

Velocity limit for move.

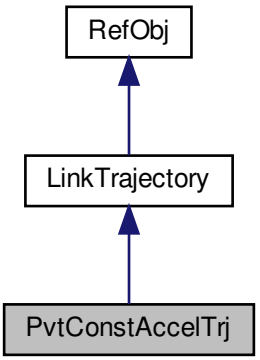
This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

The documentation for this struct was generated from the following file:

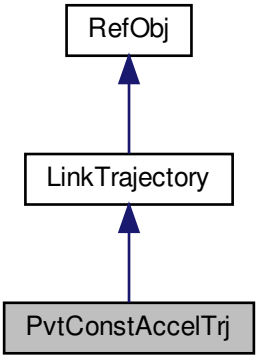
- CML_Amp.h

5.107 PvtConstAccelTrj Class Reference

Inheritance diagram for PvtConstAccelTrj:



Collaboration diagram for PvtConstAccelTrj:



Public Member Functions

- virtual int GetDim (void)

  Get the dimension of the trajectory.

Additional Inherited Members

5.107.1 Member Function Documentation

5.107.1.1 GetDim()

```
int GetDim (
            void  )  [virtual]
```

Get the dimension of the trajectory.

The trajectory dimension gives the number of axes defined for the trajectory. This can not change from the time Start↩ New() is called to the time Finish() is called. The position and velocity arrays passed to NextSegment will be of at least this size.

Returns

The dimension of the trajectory.

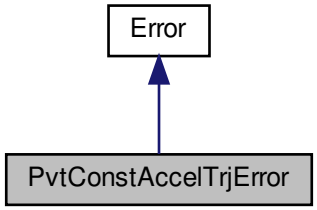
Implements LinkTrajectory.

The documentation for this class was generated from the following files:

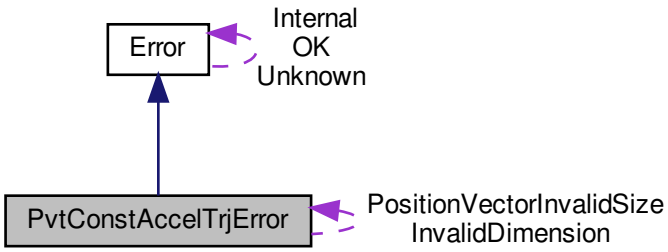
- CML_PvtConstAccelTrj.h
- PvtConstAccelTrj.cpp

5.108 PvtConstAccelTrjError Class Reference

Inheritance diagram for PvtConstAccelTrjError:



Collaboration diagram for PvtConstAccelTrjError:



Static Public Attributes

- static const PvtConstAccelTrjError PositionVectorInvalidSize

  The user input an empty multidimensional position vector.
- static const PvtConstAccelTrjError InvalidDimension

  The user input an empty multidimensional time vector.

Protected Member Functions

- PvtConstAccelTrjError (uint16 id, const char ∗desc)

  Standard protected constructor.

Additional Inherited Members

The documentation for this class was generated from the following file:

- CML_PvtConstAccelTrj.h

5.109 PvtSegCache Class Reference

PVT trajectory segment cache object.

Public Member Functions

- PvtSegCache ()

  Default constructor. Clears the cache.
- void Clear ()

  Clear the cache.
- void AddSegment (uint8 *seg, uint16 id, uunit p)

  Add the passed segment to the cache.
- bool GetSegment (uint8 *seg, uint16 id)

  Get the specified segment from the cache.
- bool GetPosition (uunit *p, uint16 id)

  Get the position corresponding to the specified segment from the cache.

5.109.1 Detailed Description

PVT trajectory segment cache object.

This is used internally by the Amp object to keep track of PVT segments recently sent. It allows the amp object to recover if a segment is lost in transit by resending the missing segments.

5.109.2 Member Function Documentation

5.109.2.1 AddSegment()

```
void AddSegment (
            uint8 * seg,
            uint16 id,
            uunit p )
```

Add the passed segment to the cache.

Segments must be passed in order and with no gaps between ID numbers. If this segment doesn't follow those rules then the cache will be cleared before the segment is added.

Parameters

seg	Points to an array of 8 bytes which make up the segment to be added. The segment data is copied into the cache. No copy of the pointer is kept locally.
id	The ID number of the passed segment.
p	The position corresponding to this segment.

5.109.2.2 GetPosition()

```
bool GetPosition (
            uunit * p,
            uint16 id )
```

Get the position corresponding to the specified segment from the cache.

If the requested position is available, it will be copied to the passed pointer.

Parameters

p	A pointer to where the position information will be copied.
id	The ID number of the segment being requested

Returns

true on success, false if the requested segment isn't available.

5.109.2.3 GetSegment()

```
bool GetSegment (
            uint8 * seg,
            uint16 id )
```

Get the specified segment from the cache.

If the requested segment is available, it's contents will be copied to the passed pointer.

Parameters

seg	A pointer to an array of 8 bytes where the segment data will be copied on success.
id	The ID number of the segment being requested

Returns

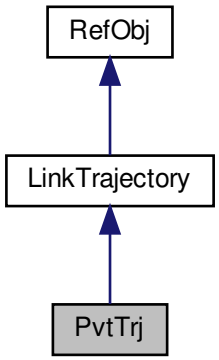
true on success, false if the requested segment isn't available.

The documentation for this class was generated from the following files:

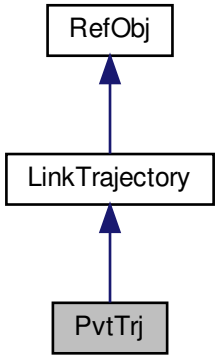
- CML_Amp.h
- AmpPVT.cpp

5.110 PvtTrj Class Reference

Inheritance diagram for PvtTrj:



Collaboration diagram for PvtTrj:



Public Member Functions

- virtual int GetDim (void)

  Get the dimension of the trajectory.

Additional Inherited Members

5.110.1 Member Function Documentation

5.110.1.1 GetDim()

```
int GetDim (
            void  )  [virtual]
```

Get the dimension of the trajectory.

The trajectory dimension gives the number of axes defined for the trajectory. This can not change from the time Start↩ New() is called to the time Finish() is called. The position and velocity arrays passed to NextSegment will be of at least this size.

Returns

The dimension of the trajectory.

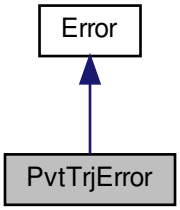
Implements LinkTrajectory.

The documentation for this class was generated from the following files:

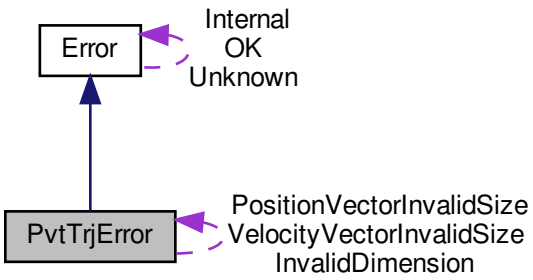
- CML_PvtTrj.h
- PvtTrj.cpp

5.111 PvtTrjError Class Reference

Inheritance diagram for PvtTrjError:



Collaboration diagram for PvtTrjError:



Static Public Attributes

- static const PvtTrjError PositionVectorInvalidSize

  The user input an empty multidimensional position vector.
- static const PvtTrjError VelocityVectorInvalidSize

  The user input an empty multidimensional velocity vector.
- static const PvtTrjError InvalidDimension

  The user input an empty multidimensional time vector.

Protected Member Functions

- PvtTrjError (uint16 id, const char ∗desc)

  Standard protected constructor.

Additional Inherited Members

The documentation for this class was generated from the following file:

- CML_PvtTrj.h

5.112 PwmInConfig Struct Reference

PWM or Pulse/Direction input configuration.

Public Member Functions

- PwmInConfig (void)

  Default constructor. Simply sets all members to zero.

Public Attributes

- int16 cfg

  PWM input pin configuration.
- int32 uvCfg

  PWM input UV configuration.
- int32 scale

  Scaling factor.
- int16 freq

  PWM input frequency.
- int16 deadBand

  PWM input deadband.

5.112.1 Detailed Description

PWM or Pulse/Direction input configuration.

These parameters are used when the amplifier is being controlled through it's PWM inputs (current or velocitiy mode), or pulse/direction input pins (position mode). These parameters have no effect when running in standard CANopen modes of operation.

5.112.2 Member Data Documentation

5.112.2.1 cfg

```
int16 cfg
```

PWM input pin configuration.

See amplifier documentation for detailed information.

5.112.2.2 deadBand

```
int16 deadBand
```

PWM input deadband.

This parameter was added to plus products starting with version 2.75. Previously (and on older products), 0x2322 was used as a deadband value. The range of 0 to 32k equals deadband of 0% to 100%.

5.112.2.3 freq

```
int16 freq
```

PWM input frequency.

This parameter is only used when running in UV current mode. For other PWM or step/dir input modes the PWM frequency is automatically calculated by the amplifier and this parameter is ignored. The frequency is set 10 Hz units. For example, setting this parameter to 100 indicates that the PWM input frequency is 1kHz.

5.112.2.4 scale

```
int32 scale
```

Scaling factor.

Units are dependent on the mode of operation: 0.01 Amp when driving current. 0.1 Encoder counts/second when driving velocity Encoder counts (upper 16 bits) / pulses (lower 16 bits) ratio for position mode.

5.112.2.5 uvCfg

```
int32 uvCfg
```

PWM input UV configuration.

Used to configure the drive when running in UV mode (desired state 5). See documentation

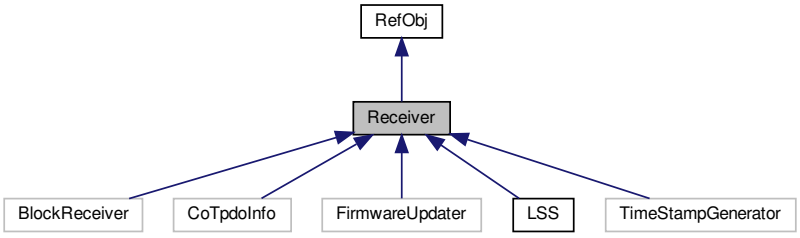
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

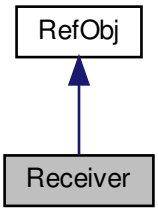
5.113 Receiver Class Reference

CANopen receiver object.

Inheritance diagram for Receiver:



Collaboration diagram for Receiver:



Public Member Functions

- Receiver ()

  Default constructor for a network receiver object.
- virtual ~Receiver ()

  Destructor for network receiver objects.
- virtual int NewFrame (CanFrame &frame)

  Process a new received CAN bus frame.

Additional Inherited Members

5.113.1 Detailed Description

CANopen receiver object.

This class allows the programmer to create routines that are run whenever a CAN frame with a specific ID is received.

To run code when a message is received, create a new class that extends Receiver. The Receiver::Init() function should be called with the CAN message ID of the frames to be received. Whenever this new class is Enabled(), the member function NewFrame() will be called once for every frame received with a matching ID.

5.113.2 Constructor & Destructor Documentation

5.113.2.1 ~Receiver()

```
~Receiver ( )  [virtual]
```

Destructor for network receiver objects.

This destructor ensures that the receiver is disabled before it is destroyed.

5.113.3 Member Function Documentation

5.113.3.1 NewFrame()

```
int NewFrame (
            CanFrame & frame )  [virtual]
```

Process a new received CAN bus frame.

This virtual function is called by the CANopen read thread every time a CAN frame is received over the network with an ID matching the receivers ID if the receiver is enabled.

Note that this function is called from the CANopen receive thread. No other receive frames will be processed until this function returns.

Also note that the map object used to associate message IDs with receive objects is locked when this function is called. The locking is required to prevent a race condition that could occur when a receive object is disabled and it's memory is deallocated. Since the map is locked, it's illegal to Enable() or Disable() any receive object from within this function.

Parameters

frame	The CAN frame to be processed. Note that the memory holding the frame structure may be reused after the call returns. If the frame contents are to be used after the return the a copy of the frame should be made.

Returns

non-zero if the frame was handled, zero if the frame type was unknown.

Reimplemented in LSS.

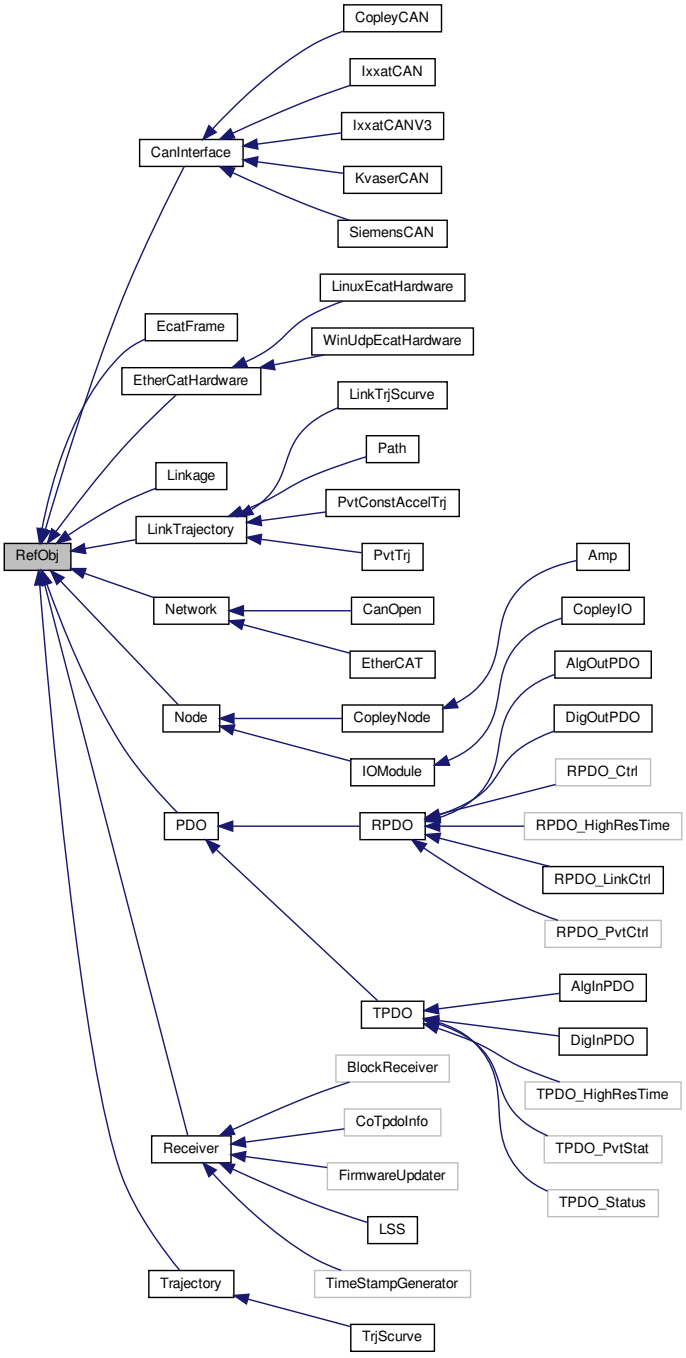
The documentation for this class was generated from the following files:

- CML_CanOpen.h
- CanOpen.cpp

5.114 RefObj Class Reference

This class is used to track object references in the CML library.

Inheritance diagram for RefObj:



Public Member Functions

- RefObj (const char *name=0)

Default constructor for a reference object.

- virtual ~RefObj ()

  Reference object destructor.

- void SetRefName (const char *name)

  Assign a name to this reference.

- uint32 GrabRef (void)

  Grab a reference to this object.

- void setAutoDelete (bool autoDeleteEna)

  This function is used to enable or disable the autodelete function for a reference object.

- void UnlockRef (void)

  Unlock an object that was previously locked using RefObj::Lock.

Static Public Member Functions

- static void ReleaseRef (uint32 id)

  Release the local reference to this object.

- static RefObj * LockRef (uint32 id)

  Find the object associated with the passed reference number and lock it to prevent the object from being destroyed while I'm accessing it.

- static bool GrabRef (uint32 id)

  Grab a reference to the object identified by the reference ID.

- static void LogRefs (void)

  This function is provided for debugging.

Protected Member Functions

- void KillRef (void)

  Destroy this reference.

5.114.1 Detailed Description

This class is used to track object references in the CML library.

Most CML objects use the RefObj class as a base class. This class assignes an integer ID to the object which can be used as a safer alternative to keeping a pointer to the object.

5.114.2 Constructor & Destructor Documentation

5.114.2.1 RefObj()

```
RefObj (
            const char * name = 0 )
```

Default constructor for a reference object.

This allocates a reference ID which will be associated with this object for as long as any references to the object exist. This ID can be safely used to find a pointer to the object.

Parameters

name	An optional string that identifies the reference for debugging purposes. If a string is passed it should persist for the entire life of the reference object.

5.114.2.2 ~RefObj()

```
~RefObj ( )  [virtual]
```

Reference object destructor.

If this object has been locked then this function won't return until the reference has been unlocked.

5.114.3 Member Function Documentation

5.114.3.1 GrabRef() [1/2]

```
uint32 GrabRef (
            void  )
```

Grab a reference to this object.

This function increases the reference count associated with the object. The reference returned can be safely used to lock and unlock the associated object.

For each call to GrabRef, there should be a corresponding call to RefObj::ReleaseRef to release the reference when it's no longer needed.

Returns

The object referece, or 0 if it wasn't possible to grab a reference to the object.

5.114.3.2 GrabRef() [2/2]

```
bool GrabRef (
            uint32 id )  [static]
```

Grab a reference to the object identified by the reference ID.

This function increases the reference count associated with the object and can be used if a reference ID needs to be copied and stored for use by another class.

For each call to GrabRef, there should be a corresponding call to RefObj::ReleaseRef to release the reference when it's no longer needed.

Parameters

id	The reference ID of the object.

Returns

true if the reference count was successfully increased. False if an invalid reference ID was passed.

5.114.3.3 KillRef()

```
void KillRef (
            void  )  [protected]
```

Destroy this reference.

This function should be called at the beginning of the destructor of any object that inherits from RefObj. It removes this reference from the system and delays until no other threads are actively using a pointer to the referenced object.

This should be the first thing done in a destructor of any class that inherits from a RefObj, even if it inherits indirectly from the reference. When KillRef is called, the reference class makes sure that no other thread is holding a lock on the class. This prevents accidental object deletion while an object is still in use by another thread.

5.114.3.4 LockRef()

```
RefObj * LockRef (
            uint32 val )  [static]
```

Find the object associated with the passed reference number and lock it to prevent the object from being destroyed while I'm accessing it.

The lock should only be held for a short time because it can prevent other threads from deleting the object. Call RefObj::Unlock when finished accessing the object.

Parameters

val	The reference ID associated with the object

Returns

A pointer to the referenced object if it still exists, or NULL if it's been destroyed.

5.114.3.5 LogRefs()

```
void LogRefs (
            void  )  [static]
```

This function is provided for debugging.

It prints out information on all references that are currently held to the cml.log file

5.114.3.6 ReleaseRef()

```
void ReleaseRef (
            uint32 val )  [static]
```

Release the local reference to this object.

Parameters

val	The reference previously returned by a call to RefObj::GrabRef

5.114.3.7 setAutoDelete()

```
void setAutoDelete (
            bool autoDeleteEna )
```

This function is used to enable or disable the autodelete function for a reference object.

If automatic deletion is enabled, the object will be deleted automatically when it's reference count indicates that there are no other objects in the system which are still holding a reference to it.

Obviously, this should only be enabled for objects that have been allocated from the heap using the new operator.

Parameters

autoDeleteEna	Boolean that if is true enabled the auto delete

5.114.3.8 SetRefName()

```
void SetRefName (
            const char * name )
```

Assign a name to this reference.

The name is used for debugging purposes.

Parameters

name	Pointer to the name. Note that a local copy of this pointer will be stored in the reference object.

The documentation for this class was generated from the following files:

- CML_Reference.h
- Reference.cpp

5.115 RefObjLocker< RefClass > Class Template Reference

This is a utility class that locks a reference in it's constructor, and unlocks it in it's destructor.

Public Member Functions

- RefObjLocker (uint32 r)
  Lock the passed reference.
- ~RefObjLocker ()
  Unlock the reference.
- RefClass * operator-> (void)
  Return a pointer to the referenced object.
- RefClass & operator* (void)
  Return a C++ reference to the referenced object.

5.115.1 Detailed Description

template<class RefClass>
class RefObjLocker< RefClass >

This is a utility class that locks a reference in it's constructor, and unlocks it in it's destructor.

It can be used to ensure that a reference is properly unlocked when a function returns.

5.115.2 Constructor & Destructor Documentation

5.115.2.1 RefObjLocker()

```
RefObjLocker (
            uint32 r )  [inline]
```

Lock the passed reference.

Parameters

r	The reference to lock

5.115.3 Member Function Documentation

5.115.3.1 operator∗()

```
RefClass& operator* (
            void  )  [inline]
```

Return a C++ reference to the referenced object.

Returns

a C++ reference to the locked reference object

5.115.3.2 operator->()

```
RefClass* operator-> (
            void  )  [inline]
```

Return a pointer to the referenced object.

Returns

a pointer to the locked reference object

The documentation for this class was generated from the following file:

- CML_Reference.h

5.116 RegenConfig Struct Reference

Configuration structure used to set up the amplifier regeneration resister.

Public Member Functions

- RegenConfig (void)

  Default constructor.

Public Attributes

- char model [COPLEY_MAX_STRING]

  Model number / name string for regen resister connected to the amplifier.
- uint16 resistance

  Regen resister resistance (100 milliohm units)
- uint16 contPower

  Continuous power limit for regen resister (Watts).
- uint16 peakPower

  Peak power limit for resister (Watts).
- uint16 peakTime

  Peak time limit (milliseconds).
- uint16 vOn

  Regen resister turn on voltage (100 millivolt units).
- uint16 vOff

  Regen resister turn off voltage (100 millivolt units).

5.116.1 Detailed Description

Configuration structure used to set up the amplifier regeneration resister.

The regen resister is not available on all amplifier models (currently only on the Xenus offline amplifier).

These settings may be up/download from the amplifier using the functions Amp::SetRegenConfig and Amp::GetRegen↩ Config.

5.116.2 Constructor & Destructor Documentation

5.116.2.1 RegenConfig()

```
RegenConfig (
            void  )  [inline]
```

Default constructor.

Initializes all structure elements to zero.

5.116.3 Member Data Documentation

5.116.3.1 contPower

```
uint16 contPower
```

Continuous power limit for regen resister (Watts).

This is the amount of power that the resister is able to disapate continuously

5.116.3.2 model

```
char model[COPLEY_MAX_STRING]
```

Model number / name string for regen resister connected to the amplifier.

5.116.3.3 peakPower

```
uint16 peakPower
```

Peak power limit for resister (Watts).

This is the maximum amount of power that the resister is able to dissapate for a limited amount of time.

5.116.3.4 peakTime

```
uint16 peakTime
```

Peak time limit (milliseconds).

This is the amount of time that the regen resister is able to dissapate peak power before it needs to be folded back to the continuous power limit.

5.116.3.5 vOff

```
uint16 vOff
```

Regen resister turn off voltage (100 millivolt units).

When the bus voltage drops below this value, the regen resiter will be disabled.

5.116.3.6 vOn

`uint16 vOn`

Regen resister turn on voltage (100 millivolt units).

When the bus voltage rises above this value the regen resister will be enabled.

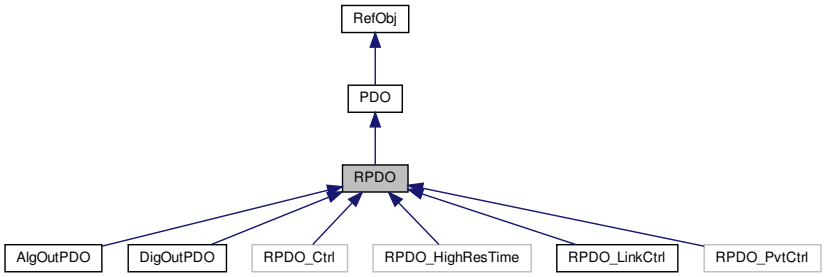
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

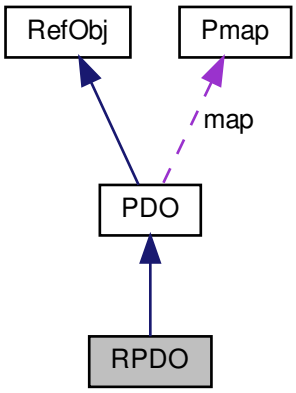
5.117 RPDO Class Reference

Receive PDO (received by node, transmitted by this software).

Inheritance diagram for RPDO:



Collaboration diagram for RPDO:



Public Member Functions

- RPDO ()
  Default constructor.
- bool IsTxPDO (void)
  Always return false for transmit PDO objects.
- RPDO (uint32 cobID)
  Construct the PDO object and initialize it.
- virtual ~RPDO ()
  Virtual destructor.
- virtual const Error * Init (uint32 cobID)
  Initialize the PDO object.
- virtual const Error * Transmit (Network &n)
  Transmit this PDO over the passed network.
- int LoadData (uint8 *buff, int max)
  Load the data from this PDO into the passed buffer.

Additional Inherited Members

5.117.1 Detailed Description

Receive PDO (received by node, transmitted by this software).

5.117.2 Constructor & Destructor Documentation

5.117.2.1 RPDO()

```
RPDO (
            uint32 cobID )  [inline]
```

Construct the PDO object and initialize it.

Parameters

cobID	The CAN message ID associated with this PDO

5.117.3 Member Function Documentation

5.117.3.1 Init()

```
virtual const Error* Init (
            uint32 cobID )  [inline], [virtual]
```

Initialize the PDO object.

Parameters

cobID	The CAN message ID associated with this PDO

Returns

A pointer to an error object, or NULL on success

5.117.3.2 LoadData()

```
int LoadData (
            uint8 * buff,
            int max )
```

Load the data from this PDO into the passed buffer.

Parameters

buff	Buffer where data will be loaded
max	The maximum number of bytes to load.

Returns

The actual number of bytes loaded.

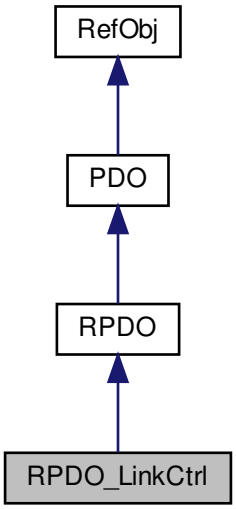
The documentation for this class was generated from the following files:

- CML_PDO.h
- PDO.cpp

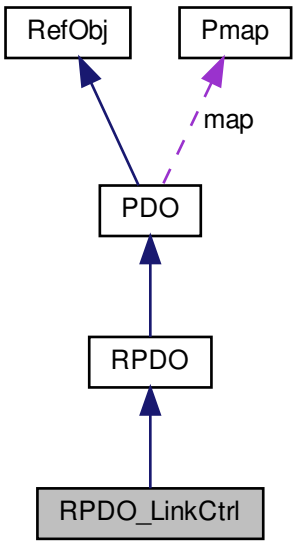
5.118 RPDO_LinkCtrl Class Reference

Receive PDO used to update the control word of all amplifiers in the linkage.

Inheritance diagram for RPDO_LinkCtrl:



Collaboration diagram for RPDO_LinkCtrl:



Public Member Functions

- RPDO_LinkCtrl (class Linkage &l)

  Default constructor for this PDO.
- const Error * Init (void)

  Initialize the receive PDO used to control words to each amplifier held by a linkage.
- const Error * Transmit (uint16 c)

  Transmit a control word using this PDO.

Additional Inherited Members

5.118.1 Detailed Description

Receive PDO used to update the control word of all amplifiers in the linkage.

This object is intended for internal use only.

5.118.2 Member Function Documentation

5.118.2.1 Init()

```
const Error * Init (
            void  )
```

Initialize the receive PDO used to control words to each amplifier held by a linkage.

The COB ID used for this PDO is the standard ID used for RPDO 1 of the first axis.

Returns

An error object pointer on failure, NULL on success

5.118.2.2 Transmit()

```
const Error * Transmit (
            uint16 c )
```

Transmit a control word using this PDO.

Parameters

c	Value of the control word to send.

Returns

A pointer to an error object, or NULL on success.

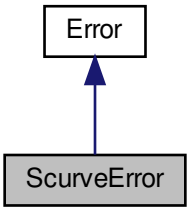
The documentation for this class was generated from the following files:

- CML_Linkage.h
- Linkage.cpp

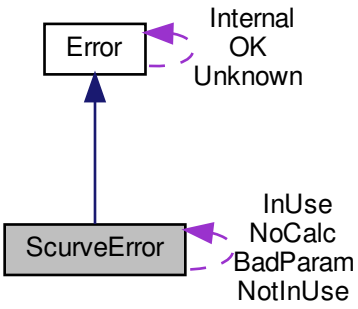
5.119 ScurveError Class Reference

This class represents error conditions that can occur in the TrjScurve class.

Inheritance diagram for ScurveError:



Collaboration diagram for ScurveError:



Static Public Attributes

- static const ScurveError BadParam

  Illegal input parameter.
- static const ScurveError NoCalc

  Trjaectory has not been calculated.
- static const ScurveError InUse

  Trajectory is currently in use.
- static const ScurveError NotInUse

  Trajectory has not been started.

Protected Member Functions

- ScurveError (uint16 id, const char ∗desc)

  Standard protected constructor.

Additional Inherited Members

5.119.1 Detailed Description

This class represents error conditions that can occur in the TrjScurve class.

The documentation for this class was generated from the following file:

- CML_TrjScurve.h

5.120 SDO Class Reference

CANopen Service Data Object (SDO).

Public Member Functions

- SDO ()

  Default SDO object constructor.
- const Error ∗ Init (Node ∗node, Timeout to=2000)

  Initialize a CANopen Service Data Object (SDO).
- const Error ∗ Download (int16 index, int16 sub, int32 size, byte ∗data)

  Download data using this SDO.
- const Error ∗ Upload (int16 index, int16 sub, int32 &size, byte ∗data)

  Upload data using this SDO.
- const Error ∗ BlockDnld (int16 index, int16 sub, int32 size, byte ∗data)

  Download data using this SDO.
- const Error ∗ BlockUpld (int16 index, int16 sub, int32 &size, byte ∗data)

Upload data using this SDO.

- const Error * Dnld32 (int16 index, int16 sub, uint32 data)

  Download a 32-bit value using this SDO.
- const Error * Upld32 (int16 index, int16 sub, uint32 &data)

  Upload a 32-bit value using this SDO.
- const Error * Dnld16 (int16 index, int16 sub, uint16 data)

  Download a 16-bit value using this SDO.
- const Error * Upld16 (int16 index, int16 sub, uint16 &data)

  Upload a 16-bit value using this SDO.
- const Error * Dnld8 (int16 index, int16 sub, uint8 data)

  Download a 8-bit value using this SDO.
- const Error * Upld8 (int16 index, int16 sub, uint8 &data)

  Upload a 8-bit value using this SDO.
- const Error * DnldString (int16 index, int16 sub, char *data)

  Download a visible string type using the SDO.
- const Error * UpldString (int16 index, int16 sub, int32 &len, char *data)

  Upload a visible string type from the SDO.
- const Error * Download (int16 index, int16 sub, int32 size, char *data)

  Download data using this SDO.
- const Error * Upload (int16 index, int16 sub, int32 &size, char *data)

  Upload data using this SDO.
- const Error * Dnld32 (int16 index, int16 sub, int32 data)

  Download a 32-bit signed integer using this SDO.
- const Error * Upld32 (int16 index, int16 sub, int32 &data)

  Upload a 32-bit signed integer using this SDO.
- const Error * DnldFlt (int16 index, int16 sub, float data)

  Download a floating point value using this SDO.
- const Error * UpldFlt (int16 index, int16 sub, float &data)

  Upload a floating point value using this SDO.
- const Error * Dnld16 (int16 index, int16 sub, int16 data)

  Download a 16-bit signed integer using this SDO.
- const Error * Upld16 (int16 index, int16 sub, int16 &data)

  Upload a 16-bit signed integer using this SDO.
- const Error * Dnld8 (int16 index, int16 sub, int8 data)

  Download an 8-bit signed integer using this SDO.
- const Error * Upld8 (int16 index, int16 sub, int8 &data)

  Upload an 8-bit signed integer using this SDO.
- void SetTimeout (Timeout to)

  Set the timeout used with this SDO.
- Timeout GetTimeout (void)

  Get the timeout used with this SDO.
- const Error * EnableBlkUpld (void)

  Enable the use of block uploads with this SDO object.
- const Error * DisableBlkUpld (void)

  Disable the use of block uploads with this SDO object.
- const Error * EnableBlkDnld (void)

  Enable the use of block downloads with this SDO object.

- const Error * DisableBlkDnld (void)

  Disable the use of block downloads with this SDO object.
- uint8 GetMaxRetry (void)

  Return the maximum number times that the SDO transfer will be attempted before returning an error.
- void SetMaxRetry (uint8 max)

  Set the maximum number of times the SDO transfer will be attempted before returning an error.

5.120.1 Detailed Description

CANopen Service Data Object (SDO).

This class represents the state of a CANopen SDO object. SDO objects are used to access the values in the object dictionary on a node. This class handles the low level protocol details of an SDO connection.

5.120.2 Constructor & Destructor Documentation

5.120.2.1 SDO()

```
SDO (
            void  )
```

Default SDO object constructor.

The SDO must be initialized by calling SDO::Init before it's actually used.

5.120.3 Member Function Documentation

5.120.3.1 BlockDnld()

```
const Error * BlockDnld (
            int16 index,
            int16 sub,
            int32 size,
            byte * data )
```

Download data using this SDO.

This function uses a block download protocol which makes sending large blocks of data more efficient. The data passed to this function is downloaded to the object dictionary of the CANopen node

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
size	The number of bytes of data to be downloaded
data	A character array holding the data to be downloaded.

Returns

A valid CANopen error object.

5.120.3.2 BlockUpld()

```
const Error * BlockUpld (
              int16 index,
              int16 sub,
              int32 & size,
              byte * data )
```

Upload data using this SDO.

This function uses a block upload protocol which makes sending large blocks of data more efficient. The specified object is upload from the CANopen node's object dictionary and stored in the array passed to this function.

Parameters

index	The index of the object to be uploaded.
sub	The sub-index of the object to be uploaded.
size	On entry, this should be the maximum number of bytes to upload, on successful return, this is the number of bytes actually received.
data	A character array which will store the uploaded data.

Returns

A valid CANopen error object.

5.120.3.3 DisableBlkDnld()

```
const Error * DisableBlkDnld (
              void  )
```

Disable the use of block downloads with this SDO object.

Returns

A CANopen error object, or null on success.

5.120.3.4 DisableBlkUpld()

```
const Error * DisableBlkUpld (
            void  )
```

Disable the use of block uploads with this SDO object.

Returns

A CANopen error object, or null on success.

5.120.3.5 Dnld16() [1/2]

```
const Error * Dnld16 (
            int16 index,
            int16 sub,
            uint16 data )
```

Download a 16-bit value using this SDO.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.6 Dnld16() [2/2]

```
const Error* Dnld16 (
            int16 index,
            int16 sub,
            int16 data )  [inline]
```

Download a 16-bit signed integer using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.7 Dnld32() [1/2]

```
const Error * Dnld32 (
            int16 index,
            int16 sub,
            uint32 data )
```

Download a 32-bit value using this SDO.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.8 Dnld32() [2/2]

```
const Error* Dnld32 (
            int16 index,
            int16 sub,
            int32 data )  [inline]
```

Download a 32-bit signed integer using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.9 Dnld8() [1/2]

```
const Error * Dnld8 (
            int16 index,
            int16 sub,
            uint8 data )
```

Download a 8-bit value using this SDO.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.10 Dnld8() [2/2]

```
const Error* Dnld8 (
            int16 index,
            int16 sub,
            int8 data )  [inline]
```

Download an 8-bit signed integer using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.11 DnldFlt()

```
const Error* DnldFlt (
             int16 index,
             int16 sub,
             float data )  [inline]
```

Download a floating point value using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be downloaded

Returns

A valid CANopen error code.

5.120.3.12 DnldString()

```
const Error * DnldString (
             int16 index,
             int16 sub,
             char * data )
```

Download a visible string type using the SDO.

The string is assumed to be null terminated.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	A null terminated string to be downloaded.

Returns

A valid CANopen error code.

5.120.3.13 Download() [1/2]

```
const Error * Download (
            int16 index,
            int16 sub,
            int32 size,
            byte * data )
```

Download data using this SDO.

The passed array of data is downloaded to the object dictionary of a node on the network using this SDO.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
size	The number of bytes of data to be downloaded
data	A character array holding the data to be downloaded.

Returns

A valid CANopen error object.

5.120.3.14 Download() [2/2]

```
const Error* Download (
            int16 index,
            int16 sub,
            int32 size,
            char * data )  [inline]
```

Download data using this SDO.

The passed array of data is downloaded to the object dictionary of a node on the CANopen network using this SDO.

Parameters

index	The index of the object to be downloaded.
sub	The sub-index of the object to be downloaded.
size	The number of bytes of data to be downloaded
data	A character array holding the data to be downloaded.

Returns

A valid CANopen error object.

5.120.3.15 EnableBlkDnld()

```
const Error * EnableBlkDnld (
            void  )
```

Enable the use of block downloads with this SDO object.

Returns

A CANopen error object, or null on success.

5.120.3.16 EnableBlkUpld()

```
const Error * EnableBlkUpld (
            void  )
```

Enable the use of block uploads with this SDO object.

Returns

A CANopen error object, or null on success.

5.120.3.17 GetMaxRetry()

```
uint8 GetMaxRetry (
            void  )  [inline]
```

Return the maximum number times that the SDO transfer will be attempted before returning an error.

Returns

The max number of retries

5.120.3.18 GetTimeout()

```
Timeout GetTimeout (
            void  )  [inline]
```

Get the timeout used with this SDO.

Returns

The timeout in milliseconds

5.120.3.19 Init()

```
const Error * Init (
            Node * n,
            Timeout to = 2000 )
```

Initialize a CANopen Service Data Object (SDO).

Parameters

n	Pointer to the node that this SDO is associated with.
to	The timeout (milliseconds) for use with this SDO.

Returns

A valid CANopen error object

5.120.3.20 SetTimeout()

```
void SetTimeout (
            Timeout to )  [inline]
```

Set the timeout used with this SDO.

Parameters

to	The timeout in milliseconds

5.120.3.21 Upld16() [1/2]

```
const Error * Upld16 (
            int16 index,
            int16 sub,
            uint16 & data )
```

Upload a 16-bit value using this SDO.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	The uploaded data will be returned here.

Returns

A valid CANopen error code.

5.120.3.22 Upld16() [2/2]

```
const Error* Upld16 (
            int16 index,
            int16 sub,
            int16 & data )  [inline]
```

Upload a 16-bit signed integer using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be uploaded

Returns

A valid CANopen error code.

5.120.3.23 Upld32() [1/2]

```
const Error * Upld32 (
            int16 index,
```

```
            int16 sub,
            uint32 & data )
```

Upload a 32-bit value using this SDO.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	The uploaded data will be returned here.

Returns

A valid CANopen error code.

5.120.3.24 Upld32() [2/2]

```
const Error* Upld32 (
            int16 index,
            int16 sub,
            int32 & data )  [inline]
```

Upload a 32-bit signed integer using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be uploaded

Returns

A valid CANopen error code.

5.120.3.25 Upld8() [1/2]

```
const Error * Upld8 (
            int16 index,
            int16 sub,
            uint8 & data )
```

Upload a 8-bit value using this SDO.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
data	The uploaded data will be returned here.

Returns

A valid CANopen error code.

5.120.3.26 Upld8() [2/2]

```
const Error* Upld8 (
            int16 index,
            int16 sub,
            int8 & data )  [inline]
```

Upload an 8-bit signed integer using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be uploaded

Returns

A CANopen error object, or null on success.

5.120.3.27 UpldFlt()

```
const Error* UpldFlt (
            int16 index,
            int16 sub,
            float & data )  [inline]
```

Upload a floating point value using this SDO.

Parameters

index	The index of the object to access
sub	The sub-index of the object
data	The data to be uploaded

Returns

A valid CANopen error code.

5.120.3.28 UpldString()

```
const Error * UpldString (
            int16 index,
            int16 sub,
            int32 & len,
            char * data )
```

Upload a visible string type from the SDO.

The only difference between this function and the lower level Upload function is that this function guarantees that there will be a zero character at the end of the string.

Parameters

index	The index of the object in the object dictionary
sub	The sub-index of the object in the object dictionary
len	Holds the size of the buffer on entry, and the length of the downloaded data on return.
data	The uploaded string will be returned here.

Returns

A valid CANopen error code.

5.120.3.29 Upload() [1/2]

```
const Error * Upload (
            int16 index,
            int16 sub,
            int32 & size,
            byte * data )
```

Upload data using this SDO.

The value of the object is uploaded from the object dictionary of a node on the CANopen network using this SDO. The results of the upload are stored in the passed buffer.

Parameters

index	The index of the object to be uploaded.
sub	The sub-index of the object to be uploaded.
size	On entry, this gives the maximum number of bytes of data to be uploaded. On successful return, it gives the actual number of bytes received.
data	A character array which will store the uploaded data.

Returns

A valid CANopen error object.

5.120.3.30 Upload() [2/2]

```
const Error* Upload (
             int16 index,
             int16 sub,
             int32 & size,
             char * data )  [inline]
```

Upload data using this SDO.

The value of the object is uploaded from the object dictionary of a node on the CANopen network using this SDO. The results of the upload are stored in the passed buffer.

Parameters

index	The index of the object to be uploaded.
sub	The sub-index of the object to be uploaded.
size	The number of bytes of data to be uploaded
data	A character array which will store the uploaded data.

Returns

A valid CANopen error object.

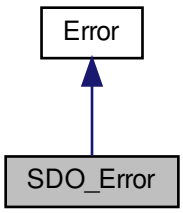
The documentation for this class was generated from the following files:

- CML_SDO.h
- SDO.cpp

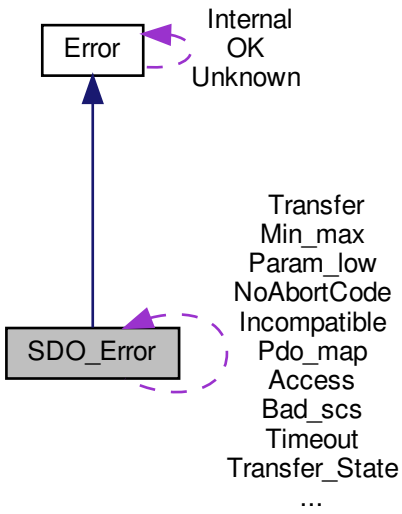
5.121 SDO_Error Class Reference

This class represents SDO errors.

Inheritance diagram for SDO_Error:



Collaboration diagram for SDO_Error:



Static Public Attributes

- static const SDO_Error NoAbortCode

  No abort code was specified with the SDO Abort message.
- static const SDO_Error Togglebit

  SDO Abort - toggle bit error.
- static const SDO_Error Timeout

SDO Abort - Timeout.

- static const SDO_Error Bad_scs
  SDO Abort - Bad SCS code.
- static const SDO_Error Block_size
  SDO Abort - Bad Block size.
- static const SDO_Error Block_seq
  SDO Abort - Block sequence error.
- static const SDO_Error Block_crc
  SDO Abort - Block CRC error.
- static const SDO_Error Memory
  SDO Abort - Memory allocation failure.
- static const SDO_Error Access
  SDO Abort - Access mode error.
- static const SDO_Error Writeonly
  SDO Abort - Object is write only.
- static const SDO_Error Readonly
  SDO Abort - Object is read only.
- static const SDO_Error Bad_object
  SDO Abort - Bad object specified.
- static const SDO_Error Pdo_map
  SDO Abort - PDO Mapping error.
- static const SDO_Error Pdo_length
  SDO Abort - PDO Length error.
- static const SDO_Error Bad_param
  SDO Abort - Bad parameter.
- static const SDO_Error Incompatible
  SDO Abort - Incompatible error.
- static const SDO_Error Hardware
  SDO Abort - Hardware error.
- static const SDO_Error Bad_length
  SDO Abort - Bad length specified.
- static const SDO_Error Too_long
  SDO Abort - Data too long for object.
- static const SDO_Error Too_short
  SDO Abort - Data too short for object.
- static const SDO_Error Subindex
  SDO Abort - Subindex is invalid.
- static const SDO_Error Param_range
  SDO Abort - Parameter range error.
- static const SDO_Error Param_high
  SDO Abort - Parameter too high.
- static const SDO_Error Param_low
  SDO Abort - Parameter too low.
- static const SDO_Error Min_max
  SDO Abort - Max less then min.
- static const SDO_Error General
  SDO Abort - General error.

- static const SDO_Error Transfer
  SDO Abort - Transfer error.
- static const SDO_Error Transfer_Local
  SDO Abort - Local transfer error.
- static const SDO_Error Transfer_State
  SDO Abort - Transfer state error.
- static const SDO_Error OD_Gen_Fail
  SDO Abort - Object dictionary generation failure.
- static const SDO_Error Unknown
  SDO Abort - Unknown abort code.
- static const SDO_Error NoBlkXfers
  Network does not support block transfers.
- static const SDO_Error ObjMapActive
  SDO Abort - sync manager mapping can't be changed while active.

Protected Member Functions

- SDO_Error (uint16 id, const char *desc)
  Standard protected constructor.

Additional Inherited Members

5.121.1 Detailed Description

This class represents SDO errors.

There is one static member for each SDO abort code.

The documentation for this class was generated from the following file:

- CML_SDO.h

5.122 Semaphore Class Reference

Generic semaphore class.

Public Member Functions

- Semaphore (int32 count=0)
  Create a new semaphore object.
- virtual ~Semaphore ()
  Free any system resources associated with this semaphore.
- const Error * Get (Timeout timeout=-1)
  Get the semaphore with an optional timeout.
- const Error * Put (void)
  Increase the count of the semaphore object.

5.122.1 Detailed Description

Generic semaphore class.

Semaphores can be used to allow multiple threads to share a pool of shared resources. Semaphores can be used like mutexes, however they also implement timeouts and multiple resource counts.

5.122.2 Constructor & Destructor Documentation

5.122.2.1 Semaphore()

```
Semaphore (
            int32 count = 0 )
```

Create a new semaphore object.

If a count is passed, then the initial semaphore count will be initialized to that value. If no count is passed, then a count of zero is used.

Parameters

count	The initial count of the semaphore. The semaphore's Get method may be called that many times before any thread will block on it.

5.122.2.2 ~Semaphore()

```
virtual ~Semaphore ( )  [virtual]
```

Free any system resources associated with this semaphore.

Any threads blocking on the semaphore should return from the Get() call with an error indication.

5.122.3 Member Function Documentation

5.122.3.1 Get()

```
const Error* Get (
            Timeout timeout = -1 )
```

Get the semaphore with an optional timeout.

An error is returned if the timeout expires before the semaphore is acquired.

Parameters

timeout	The timeout in milliseconds. Any negative value will cause the thread to wait indefinitely. If a timeout of zero is specified, the calling thread will return a timeout error without blocking if the semaphore is not available.

Returns

An error code indicating success or failure.

5.122.3.2 Put()

```
const Error* Put (
             void  )
```

Increase the count of the semaphore object.

If any threads are pending on the object, then the highest priority one will be made eligible to run.

Returns

An error object

The documentation for this class was generated from the following file:

- CML_Threads.h

5.123 ServoLoopConfig Struct Reference

This structure holds configuration info about specific parts of the velocity and position loops.

Public Attributes

- int32 servoLoopConfig

  Servo Loop Configuration This paramater it set up as follows: Bit 0: If set, disables the velocity loop gains.

5.123.1 Detailed Description

This structure holds configuration info about specific parts of the velocity and position loops.

5.123.2 Member Data Documentation

5.123.2.1 servoLoopConfig

```
int32 servoLoopConfig
```

Servo Loop Configuration This paramater it set up as follows: Bit 0: If set, disables the velocity loop gains.

The velocity loop command feed forward gain is still active as are the velocity loop output filters. Bit 1: If set, this enables the position loop I and D gains. If clear, these params are treated as zeros. Bit 2-31: Reserved for future use

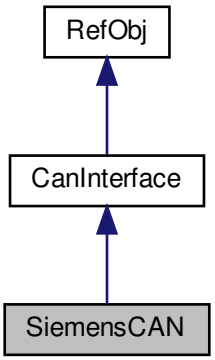
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

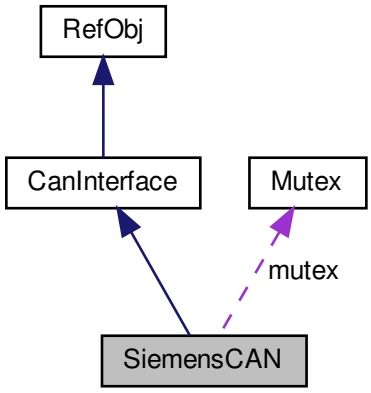
5.124 SiemensCAN Class Reference

SiemensPCIeToCAN specific CAN interface.

Inheritance diagram for SiemensCAN:



Collaboration diagram for SiemensCAN:



Public Member Functions

- SiemensCAN (void)

  Construct a default CAN object.
- SiemensCAN (const char *port)

  Construct a CAN object with a specified port name.
- virtual ~SiemensCAN (void)

  Destructor.
- const Error * Open (void)

  Open the CAN port.
- const Error * Close (void)

  Close the CAN interface.
- const Error * SetBaud (int32 baud)

  Set the CAN interface baud rate.

Protected Member Functions

- const Error * RecvFrame (CanFrame &frame, Timeout timeout)

  Receive the next CAN frame.
- const Error * XmitFrame (CanFrame &frame, Timeout timeout)

  Write a CAN frame to the CAN network.
- const Error * ConvertError (int err)

  Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Protected Attributes

- int open

  tracks the state of the interface as open or closed.
- int32 baud

  Holds a copy of the last baud rate set.
- int kvBaud

  Holds a value the SiemensPCIeToCan driver uses to identify bit rate.
- int Handle_Rd

  File handle used to configure and read from the CAN channel.
- int Handle_Wr

  File handle used to write to the CAN channel.

Additional Inherited Members

5.124.1 Detailed Description

SiemensPCIeToCAN specific CAN interface.

This class extends the generic CanInterface class into a working interface for the SiemensPCIeToCAN can device driver.

5.124.2 Constructor & Destructor Documentation

5.124.2.1 SiemensCAN() [1/2]

```
SiemensCAN (
            void  )
```

Construct a default CAN object.

The CAN interface is closed initially, and no port name is selected.

5.124.2.2 SiemensCAN() [2/2]

```
SiemensCAN (
            const char * port )
```

Construct a CAN object with a specified port name.

The port name should be of the form CANx or KVASERx where x is the port number. The port numbers start at 0, so the first port would be identified by the port name CAN0.

Parameters

port	The port name string identifying the CAN device.

5.124.2.3 ~SiemensCAN()

```
~SiemensCAN (
            void  )  [virtual]
```

Destructor.

This closes the CAN port and frees the .dll

5.124.3 Member Function Documentation

5.124.3.1 Close()

```
const Error * Close (
            void  )  [virtual]
```

Close the CAN interface.

Returns

A pointer to an error object on failure, nullptr on success.

Reimplemented from CanInterface.

5.124.3.2 ConvertError()

```
const Error * ConvertError (
            int err )  [protected]
```

Convert error codes defined by the Vector CAN library into the standard error codes used by the motion library.

Parameters

err	The Vector style status code

Returns

A pointer to an error object on failure, nullptr on success.

5.124.3.3 Open()

```
const Error * Open (
            void  )  [virtual]
```

Open the CAN port.

Before Open is called, the desired baud rate must have been specified by calling SetBaud, and the port name must have been set. If the baud was not specified, it will default to 1,000,000 BPS. If the port name is not set, it will default to CAN0.

Returns

A pointer to an error object on failure, nullptr on success.

Reimplemented from CanInterface.

5.124.3.4 RecvFrame()

```
const Error * RecvFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Receive the next CAN frame.

Parameters

frame	A reference to the frame object that will be filled by the read.
timeout	The timeout (ms) to wait for the frame. A timeout of 0 will return immediately if no data is available. A timeout of < 0 will wait forever.

Returns

A pointer to an error object on failure, nullptr on success.

Reimplemented from CanInterface.

5.124.3.5 SetBaud()

```
const Error * SetBaud (
            int32 b )  [virtual]
```

Set the CAN interface baud rate.

Parameters

b	The baud rate to set.

Returns

A pointer to an error object on failure, nullptr on success.

Reimplemented from CanInterface.

5.124.3.6 XmitFrame()

```
const Error * XmitFrame (
            CanFrame & frame,
            Timeout timeout )  [protected], [virtual]
```

Write a CAN frame to the CAN network.

Parameters

frame	A reference to the frame to write.
timeout	The time to wait for the frame to be successfully sent. If the timeout is 0, the frame is written to the output queue and the function returns without waiting for it to be sent. If the timeout is <0 then the function will delay forever.

Returns

A pointer to an error object on failure, nullptr on success.

Reimplemented from CanInterface.

The documentation for this class was generated from the following files:

- can_siemens.h
- can_siemens.cpp

5.125 SoftPosLimit Struct Reference

Software limit switch configuration.

Public Member Functions

- SoftPosLimit (void)

  Default constructor. Simply sets both limits to zero.

Public Attributes

- uunit neg

  Negative limit position.
- uunit pos

  Positive limit position.
- uunit accel

  Software limit acceleration.
- uunit motorPosWrap

  Motor position wrap value.
- uunit loadPosWrap

  Load postion wrap. Same as above except for load encoder.
- uunit macroEncoderCapture

  MACRO encoder caputure configuration.

5.125.1 Detailed Description

Software limit switch configuration.

This structure may be used to pass software limit switch settings to an Amp object using the functions Amp::SetSoftLimits and Amp::GetSoftLimits

5.125.2 Member Data Documentation

5.125.2.1 accel

```
uunit accel
```

Software limit acceleration.

This parameter defines the acceleration value that will be used to stop the motor at the software limit position. Note that this parameter was added in amplifier firmware version 4.60. Before that version the older current based software limit processing was used.

If this parameter is set to zero (the default) then the software limits will act like virtual limit switches. If the motor position exceeds the limit position then the amplifier will refuse to output current in the limit direction.

5.125.2.2 macroEncoderCapture

```
uunit macroEncoderCapture
```

MACRO encoder caputure configuration.

Configures the MACRO amplifier's encoder caputure circuit. Only implemented on MACRO amplifiers.

5.125.2.3 motorPosWrap

```
uunit motorPosWrap
```

Motor position wrap value.

Actual motor position will wrap back to zero when this value is reached. Setting this value to zero disables thsi feature. (Units counts). This feature is only implemented on the 8367 hardware with fimware 0.42 or later

5.125.2.4 neg

```
uunit neg
```

Negative limit position.

Any time the motors actual position is less then this value, a negative software limit condition will be in effect on the amplifier.

5.125.2.5 pos

```
uunit pos
```

Positive limit position.

Any time the motors actual position is greater then this value, a positive software limit condition will be in effect on the amplifier.

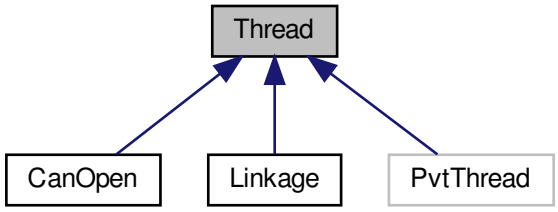
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.126 Thread Class Reference

Virtual class which provides multi-tasking.

Inheritance diagram for Thread:



Public Member Functions

- Thread (void)

  Create a new thread.
- virtual ~Thread ()

  Clean up any allocated resources.
- const Error * setPriority (int pri)

  Set the thread priority.
- const Error * start (void)

  Make this thread eligible to run.
- const Error * stop (Timeout to=1000)

  Stop this thread.
- virtual void run (void)=0

  When a new thread is started, this function will be called.
- void __run (void)

  This is an internal function which should not be called directly.

Static Public Member Functions

- static const Error * sleep (Timeout to)

  Cause the calling thread to sleep for a specified number of milliseconds.
- static uint32 getTimeMS (void)

  Return the current time in millisecond units.
- static uint32 getTimeUS (void)

  Return the current time in microsecond units.

5.126.1 Detailed Description

Virtual class which provides multi-tasking.

To add a new thread to a program, create a new class that is derived from Thread. The new thread of execution will start when the start() member function is called. This new thread of execution will start with the run() member function and will run concurrently with the rest of the system. When (if) the run method returns, the new thread will be terminated.

5.126.2 Constructor & Destructor Documentation

5.126.2.1 Thread()

```
Thread (
            void  )
```

Create a new thread.

The new thread will not start executing until the start member function of this class is called. The default thread priority will be set to 5.

5.126.3 Member Function Documentation

5.126.3.1 getTimeMS()

```
static uint32 getTimeMS (
            void  )  [static]
```

Return the current time in millisecond units.

The value returned may be offset by a consistent, but arbitrary value. This makes it useful for checking relative times, but not useful for absolute time calculations.

Returns

The time in millisecond units

5.126.3.2 getTimeUS()

```
static uint32 getTimeUS (
             void  )  [static]
```

Return the current time in microsecond units.

The value returned may be offset by a consistent, but arbitrary value. This makes it useful for checking relative times, but not useful for absolute time calculations.

Returns

The time in microsecond units

5.126.3.3 run()

```
virtual void run (
             void  )  [pure virtual]
```

When a new thread is started, this function will be called.

All of the thread specific code should be contained in this function. If the run() method ever returns, the thread will be destroyed.

5.126.3.4 setPriority()

```
const Error* setPriority (
             int pri )
```

Set the thread priority.

This function should be called before the thread is started if the default priority (5) is not acceptable.

Parameters

pri	The priority for this thread to run at. The range is 0 to 9 where 0 is a very low priority task, and 9 is a critically high priority.

Returns

An error object is returned indicating the success of the call.

5.126.3.5 sleep()

```
static const Error* sleep (
            Timeout to )  [static]
```

Cause the calling thread to sleep for a specified number of milliseconds.

Parameters

to	The time to sleep, in milliseconds.

Returns

An error object is returned indicating the success of the call.

5.126.3.6 start()

```
const Error* start (
            void  )
```

Make this thread eligible to run.

The new thread will be created if possible and identified to the operating system as eligible to run. When the thread actually starts, the run() method will be called. Note that depending on the priority of the thread and of the calling task, the run() function may or may not be called before start() returns.

Returns

An error object is returned indicating the success of the call.

5.126.3.7 stop()

```
const Error* stop (
            Timeout to = 1000 )
```

Stop this thread.

The thread will have exited by the time this function returns. If the calling thread is the thread being stopped, then this function will not return.

Parameters

to	The amount of time to wait for the thread to stop before returning an error (default 1 second).

Returns

An error object is returned indicating the success of the call.

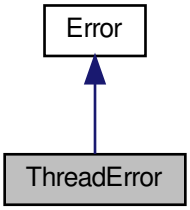
The documentation for this class was generated from the following file:

- CML_Threads.h

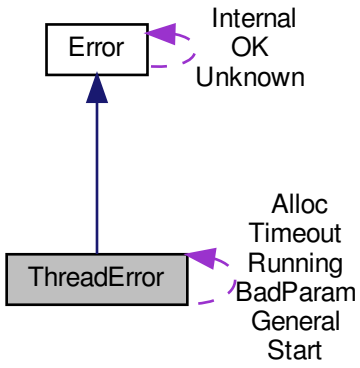
5.127 ThreadError Class Reference

Errors related to the multi-threaded libraries.

Inheritance diagram for ThreadError:



Collaboration diagram for ThreadError:



Static Public Attributes

- static const ThreadError Start

  Error starting the thread.
- static const ThreadError Running

  Thread has already been started.
- static const ThreadError Timeout

  Timeout waiting on semaphore.
- static const ThreadError General

  General failure.
- static const ThreadError BadParam

  Bad parameter passed to a thread function.
- static const ThreadError Alloc

  Error allocating memory for thread data.

Additional Inherited Members

5.127.1 Detailed Description

Errors related to the multi-threaded libraries.

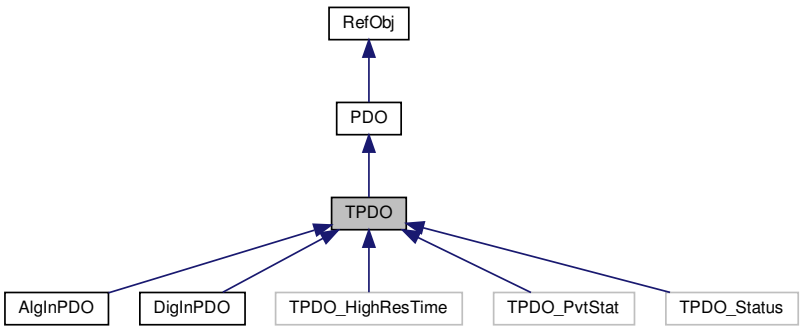
The documentation for this class was generated from the following file:

- CML_Threads.h

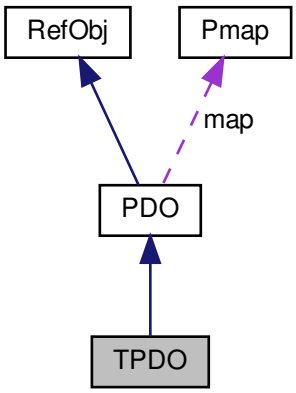
5.128 TPDO Class Reference

Transmit PDO (transmitted by node, received by this software).

Inheritance diagram for TPDO:



Collaboration diagram for TPDO:



Public Member Functions

- TPDO ()

  Default constructor.
- bool IsTxPDO (void)

  Always return true for transmit PDO objects.
- TPDO (uint32 cobID)

  Calls Init() at construction time.
- virtual ~TPDO ()

  Virtual destructor.
- virtual const Error * Init (uint32 cobID)

  Initialize the PDO.
- virtual void SetRtrOk (int ok)

  Enable or disable RTR requests for this PDO.
- virtual bool HandleRxMsg (void)

  Should the network thread call Received() on this object when a message matching the PDO ID is received? Normally this is true, but could be false if for example we are configuring a PDO to transmit from one node to another and therefore don't care about handling it ourselves.
- virtual void Received ()

  This function is called by the Network read thread when this PDO has been received.
- void ProcessData (uint8 *data, int ct, uint32 time)

  Process data received from the network.
- virtual const Error * Request (Network &net)

  Send a remote request for this PDO if the network supports it.

Protected Attributes

- uint32 timestamp

  Timestamp of received frame if available.

Additional Inherited Members

5.128.1 Detailed Description

Transmit PDO (transmitted by node, received by this software).

5.128.2 Member Function Documentation

5.128.2.1 HandleRxMsg()

```
virtual bool HandleRxMsg (
            void  )  [inline], [virtual]
```

Should the network thread call Received() on this object when a message matching the PDO ID is received? Normally this is true, but could be false if for example we are configuring a PDO to transmit from one node to another and therefore don't care about handling it ourselves.

Returns

true if the Receive() method should be called

5.128.2.2 ProcessData()

```
void ProcessData (
            uint8 * data,
            int ct,
            uint32 time )
```

Process data received from the network.

This function is called by the network object when new data is received for this PDO. It updates the values of the variables mapped to the PDO and calls the virtual Received() function.

Parameters

data	Pointer to the newly received PDO data.
ct	Size of the PDO data in bytes.
time	System time stamp indicating the time of reception

5.128.2.3 Received()

```
virtual void Received (
            void  )  [inline], [virtual]
```

This function is called by the Network read thread when this PDO has been received.

Reimplemented in IOModule::AlgInPDO, and IOModule::DigInPDO.

5.128.2.4 SetRtrOk()

```
virtual void SetRtrOk (
            int ok )  [inline], [virtual]
```

Enable or disable RTR requests for this PDO.

Parameters

ok	zero for no RTR, non-zero for RTR allowed

The documentation for this class was generated from the following files:

- CML_PDO.h
- PDO.cpp

5.129 TrackingWindows Struct Reference

Position and velocity error windows.

Public Member Functions

- TrackingWindows (void)

  Default constructor for tracking window structure.

Public Attributes

- uunit trackErr

  Tracking error window.
- uunit trackWarn

  Position warning window.
- uunit settlingWin

  Position tracking & settling window.
- uint16 settlingTime

  Position tracking & settling time (ms).
- uunit velWarnWin

  Velocity warning window See Amp::SetVelocityWarnWindow for more information.
- uint16 velWarnTime

  Velocity warning window time See Amp::SetVelocityWarnTime for more information.

5.129.1 Detailed Description

Position and velocity error windows.

5.129.2 Constructor & Destructor Documentation

5.129.2.1 TrackingWindows()

```
TrackingWindows (
            void  )
```

Default constructor for tracking window structure.

This simply sets all tracking window parameter default values of zero.

5.129.3 Member Data Documentation

5.129.3.1 settlingTime

```
uint16 settlingTime
```

Position tracking & settling time (ms).

See Amp::SetSettlingTime for more info

5.129.3.2 settlingWin

```
uunit settlingWin
```

Position tracking & settling window.

See Amp::SetSettlingWindow for more information

5.129.3.3 trackErr

```
uunit trackErr
```

Tracking error window.

See Amp::SetPositionErrorWindow for more information

5.129.3.4 trackWarn

```
uunit trackWarn
```

Position warning window.

See Amp::SetPositionWarnWindow for more information

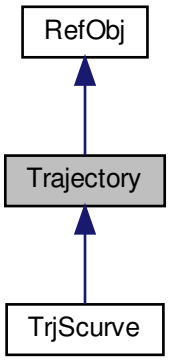
The documentation for this struct was generated from the following files:

- CML_AmpStruct.h
- AmpStruct.cpp

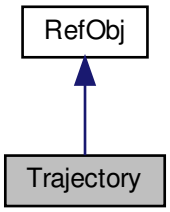
5.130 Trajectory Class Reference

Trajectory information class.

Inheritance diagram for Trajectory:



Collaboration diagram for Trajectory:



Public Member Functions

- virtual ~Trajectory ()

  Virtual destructor.
- virtual const Error * StartNew (void)

  Start a new trajectory.
- virtual void Finish (void)

  Trajectory finished.
- virtual bool UseVelocityInfo (void)

  This function indicates whether the velocity information returned by NextSegment should be used.
- virtual int MaximumBufferPointsToUse (void)

  This function allows a trajectory object to effectively reduce the size of the amplifier's internal trajectory buffer.
- virtual const Error * NextSegment (uunit &pos, uunit &vel, uint8 &time)=0

  Get the next segment of position, velocity & time info.

Additional Inherited Members

5.130.1 Detailed Description

Trajectory information class.

One of the control modes supported by the Copley Controls amplifiers is called interpolated position mode. This allows complex trajectories to be generated by the CANopen master controller, and streamed to one or more amplifiers in real time. Each point on the trajectory contains three pieces of information; Position, Velocity, and Time until the next point. For this reason, these trajectories are also often called PVT profiles.

The Amp object contains support for streaming a PVT trajectory down to the amplifier automatically (see Amp::SendTrajectory). This virtual class provides the interface through which the trajectory data is retrieved by the Amp object.

Note that this pure virtual class provides no actual implementation code. The calculation of the trajectory points is the responsibility of the inheriting class.

5.130.2 Member Function Documentation

5.130.2.1 Finish()

```
virtual void Finish (
            void  )  [inline], [virtual]
```

Trajectory finished.

This function is called by the Amp object when it is finished with the trajectory object and no longer holding a reference to it. Typically, this will happen after the Trajectory::NextSegment function has returned a zero time value, although it can also occur when some external event causes the Amp object to abort a running trajectory.

Once the Amp object calls Trajectory::Finish it will clear it's reference to the trajectory object. No further access to the trajectory object will be made after Finish is called.

Reimplemented in TrjScurve.

5.130.2.2 MaximumBufferPointsToUse()

```
virtual int MaximumBufferPointsToUse (
            void  )  [inline], [virtual]
```

This function allows a trajectory object to effectively reduce the size of the amplifier's internal trajectory buffer.

Normally it's desirable to download as many points as possible to the amplifier at once. For some applicaitons however, the trajectory information is calculated in real time and the amplifier's buffer causes unacceptable latency. For such applications this function may be used to reduce the delay between calculating trajectory points and the amplifier's acting on them.

Note that the amplifier requires some buffering of points in order to interpolate between them. This function should never return a value less then 2 or a trajectory underflow will certainly occur.

Returns

The maximum number of trajectory points that should be stored in the amplifier at any time. By default this returns a very large number which ensures that the amplifier's full buffer will be used.

5.130.2.3 NextSegment()

```
virtual const Error* NextSegment (
            uunit & pos,
            uunit & vel,
            uint8 & time )  [pure virtual]
```

Get the next segment of position, velocity & time info.

Note that this function will be called from the high priority CANopen receiver task. Therefore, no lengthy processing should be done here.

Parameters

pos	The new position value is returned here. This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.
vel	The new velocity value is returned here. This parameter is specified in "user units". See Amp::SetCountsPerUnit for details. Note that the velocity data will be ignored if the function UseVelocityInfo() returns false. In this case the amplifier will use linear interpolation between points.
time	The segment time is returned here. This is in milliseconds and ranges from 1 to 255. If zero is returned, this is the last frame in the profile.

Returns

An error object. If this is not Error::OK, then the segment data is assumed to be invalid.

Implemented in TrjScurve.

5.130.2.4 StartNew()

```
virtual const Error* StartNew (
            void  )  [inline], [virtual]
```

Start a new trajectory.

This function is called by Amp::SendTrajectory before the first call to Trajectory::NextSegment. It gives the trajectory object a chance to return an error indicating that it isn't ready to be sent.

Returns

An error pointer if the trajectory object is not available, or NULL if it is ready to be sent.

Reimplemented in TrjScurve.

5.130.2.5 UseVelocityInfo()

```
virtual bool UseVelocityInfo (
            void  )  [inline], [virtual]
```

This function indicates whether the velocity information returned by NextSegment should be used.

If this returns true, then the amplifier will operate in PVT mode and use cubic polynomial interpolation between trajectory segments. If this returns false, then the velocity returned by NextSegment will be ignored, and the amplifier will run in PT mode using linear interpolation between sets of points.

Returns

true if velocity information should be used (default), or false if velocities should be ignored.

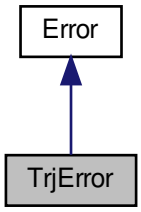
The documentation for this class was generated from the following file:

- CML_Trajectory.h

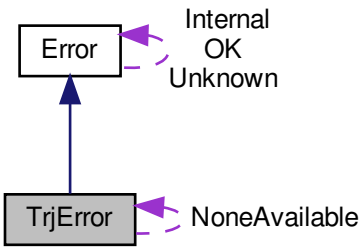
5.131 TrjError Class Reference

This class represents error conditions reported by the trajectory classes.

Inheritance diagram for TrjError:



Collaboration diagram for TrjError:



Static Public Attributes

- static const TrjError NoneAvailable

  No trajectory information available at the moment.

Protected Member Functions

- TrjError (uint16 id, const char *desc)

  Standard protected constructor.

Additional Inherited Members

5.131.1 Detailed Description

This class represents error conditions reported by the trajectory classes.

5.131.2 Member Data Documentation

5.131.2.1 NoneAvailable

```
const TrjError NoneAvailable  [static]
```

No trajectory information available at the moment.

This error code may be returned from Trajectory::NextSegment to limit the amount of trajectory information uploaded to the amplifier at one time. The Amp object will not treat this as an error condition, but will stop requesting trajectory data until the next PVT status PDO is received.

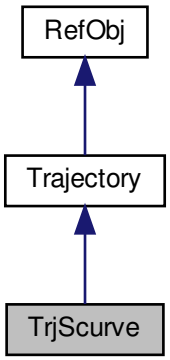
The documentation for this class was generated from the following file:

- CML_Trajectory.h

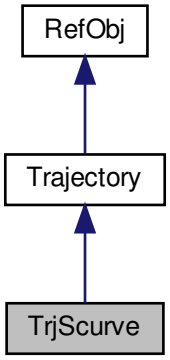
5.132 TrjScurve Class Reference

Asymmetric S-curve profile generator.

Inheritance diagram for TrjScurve:



Collaboration diagram for TrjScurve:



Public Member Functions

- TrjScurve ()

  S-curve trajectory default constructor.
- void SetStartPos (uunit s)

  Set the trajectory starting position.
- uunit GetStartPos (void)

  Return the current starting position of the trajectory.
- const Error * Calculate (uunit start, uunit end, uunit vel, uunit acc, uunit dec, uunit jrk)

  Calculate a new S-curve profile and also set it's starting position.
- const Error * Calculate (uunit dist, uunit vel, uunit acc, uunit dec, uunit jrk)

  Calculate a new S-curve profile.
- const Error * StartNew (void)

  Reset this object so it may be passed to an amplifier.
- void Finish (void)

  Notify the trajectory object that it is no longer in use.
- const Error * NextSegment (uunit &pos, uunit &vel, uint8 &time)

  Get the next PVT segment for this s-curve profile.

Additional Inherited Members

5.132.1 Detailed Description

Asymmetric S-curve profile generator.

A symmetric S-curve profile uses the same constraint for acceleration & deceleration. Asymmetric profiles use different acceleration & deceleration values. Copley amplifiers are able to calculate symmetric profiles internally, however if

asymmetric s-curve profiles are required then they must be calculate external to the amplifier, and passed to it using PVT profile mode.

This class extends the generic Trajectory class, and provides the code necessary to calculate an asymmetric s-curve profile. Since it extends the Trajectory object, it may be passed to the Amp::SendTrajectory function.

Internally, the s-curve profile is stored as an absolute move from some starting position. This allows the same trajectory object to be reused for multiple moves of the same distance from different starting positions. The starting position may be either passed to the TrjScurve::Calculate function, or set using TrjScurve::SetStartPos.

5.132.2 Constructor & Destructor Documentation

5.132.2.1 TrjScurve()

```
TrjScurve (
            void  )
```

S-curve trajectory default constructor.

This simply sets the profile to zero length with a starting position of zero.

5.132.3 Member Function Documentation

5.132.3.1 Calculate() [1/2]

```
const Error * Calculate (
            uunit start,
            uunit end,
            uunit vel,
            uunit acc,
            uunit dec,
            uunit jrk )
```

Calculate a new S-curve profile and also set it's starting position.

Parameters

start	The profile starting position.
end	The profile ending position.
vel	The maximum allowable velocity for the move.
acc	The maximum allowable acceleration for the move.
dec	The maximum allowable deceleration for the move.
jrk	The maximum jerk (rate of change of acceleration) for the move.

Returns

A pointer to an error object, or NULL on success.

5.132.3.2 Calculate() [2/2]

```
const Error * Calculate (
            uunit dist,
            uunit maxVel,
            uunit maxAcc,
            uunit maxDec,
            uunit maxJrk )
```

Calculate a new S-curve profile.

The resulting profile may then be sent to an Amp object using the Amp::SendTrajectory method.

Note, all profile parameters are passed in 'user units'. See the documentation for Amp::SetCountsPerUnit for details.

Note also that this function calculates the profile as an absolute move from the starting position that is set using TrjScurve::SetStartPos. The same profile may be used multiple times with different starting positions without calling the TrjScurve::Calculate function again.

Parameters

dist	The distance to move.
maxVel	The maximum allowable velocity for the move.
maxAcc	The maximum allowable acceleration for the move.
maxDec	The maximum allowable deceleration for the move.
maxJrk	The maximum jerk (rate of change of acceleration) for the move.

Returns

A pointer to an error object, or NULL on success.

The distance will always be met exactly. This may be either positive or negative.

The velocity, acceleration, deceleration and jerk values are constraints and won't be exceeded. These must all be positive numbers greater than zero.

5.132.3.3 GetStartPos()

```
uunit GetStartPos (
            void  )
```

Return the current starting position of the trajectory.

The starting position will either be the value set using TrjScurve::SetStartPos, or the value set using TrjScurve::Calculate. If neither has been called since construction, then the starting position will be zero.

Returns

The trajectory starting position.

5.132.3.4 SetStartPos()

```
void SetStartPos (
            uunit s )
```

Set the trajectory starting position.

S-curve profiles are internally stored as absolute moves of some length. This allows them to be used multiple times with different starting positions.

This function may be used to update the starting position of the trajectory.

Parameters

s	The new starting position

5.132.3.5 StartNew()

```
const Error * StartNew (
            void  )  [virtual]
```

Reset this object so it may be passed to an amplifier.

This will return an error if the trajectory has not yet been calculated, or if it is currently being sent to another amp.

Returns

A pointer to an error object, or NULL on success

Reimplemented from Trajectory.

The documentation for this class was generated from the following files:

- CML_TrjScurve.h
- TrjScurve.cpp

5.133 UstepConfig Struct Reference

Configuration structure used to set up the microstepper.

Public Member Functions

- UstepConfig (void)

  Default constructor.

Public Attributes

- uint32 maxVelAdj

  Maximum Velocity adjustment.
- uint16 ustepPGainOutLoop

  Propertional Gain for stepper outer loop.
- int16 detentCorrectionGain

  Detent correction Gain factor.
- uint16 ustepConfigAndStatus

  Stepper config and status Bit mapped as follows (Bit 0 Use the encoder input for phase compensation if enabled.

5.133.1 Detailed Description

Configuration structure used to set up the microstepper.

These settings may be up/download from the amplifier using the functions Amp::SetUstepConfig and Amp::GetUstep↩ Config.

5.133.2 Constructor & Destructor Documentation

5.133.2.1 UstepConfig()

```
UstepConfig (
            void  )  [inline]
```

Default constructor.

Initializes all structure elements to zero.

5.133.3 Member Data Documentation

5.133.3.1 maxVelAdj

```
uint32 maxVelAdj
```

Maximum Velocity adjustment.

This is the maximum velocity adjustment made by the stepper outer position loop when enbled. This parameter is only used when the stepper outer loop ios engaged. (when bit 1 of ustepConfig ustepConfigAndStatus is set)

5.133.3.2 ustepConfigAndStatus

```
uint16 ustepConfigAndStatus
```

Stepper config and status Bit mapped as follows (Bit 0 Use the encoder input for phase compensation if enabled.

Pure stepper if disabled (Bit 1 Use encoder outer loop to adjust the stepper position based on position error. When this bit is set, the gain value maximum velocity adjustment is multiplied by the position error, and the result is a velocity that is added to the microstepping position bits 2 - 15 reserved

5.133.3.3 ustepPGainOutLoop

```
uint16 ustepPGainOutLoop
```

Propertional Gain for stepper outer loop.

This parameter gives the gain used for calculating a velocity adjustment based on position error. This parameter is only used when stepper outer loop is engaged (when bit 1 of ustepConfig ustepConfigAndStatus is set)

The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

5.134 VelLoopConfig Struct Reference

This structure holds the velocity loop configuration parameters specific to the Copley amplifier.

Public Member Functions

- VelLoopConfig (void)

  Default constructor.

Public Attributes

- int16 kp

  Proportional gain.
- int16 ki

  Integral gain.
- int16 kaff

  Acceleration feed forward.
- int16 velCmdff

  Velocity loop command feed forward The input command (after limiting) to the velocity loop is scaled by this value and added to the output of the velocity loop.
- int16 shift

  Output shift value.
- int16 viDrain

  Velocity loop drain value for integral sum. Set to 0 to disable.
- uunit maxVel

  Maximum allowed velocity.
- uunit maxAcc

  Maximum allowed acceleration.
- uunit maxDec

  Maximum allowed deceleration This value limits the rate of change of the velocity command input to the velocity loop.
- uunit estopDec

  Deceleration used for emergency stop When the position loop is driving the velocity loop this value is only used for tracking error conditions.

5.134.1 Detailed Description

This structure holds the velocity loop configuration parameters specific to the Copley amplifier.

The velocity loop is one of three servo control loops used by the amplifier to control a motor. The configuration parameters used by this control loop allow the servo performance to be 'tuned' for various motors and loads.

The amplifier member functions Amp::GetVelLoopConfig and Amp::SetVelLoopConfig are used to read and write this data to the amplifier.

5.134.2 Constructor & Destructor Documentation

5.134.2.1 VelLoopConfig()

```
VelLoopConfig (
            void  )  [inline]
```

Default constructor.

Simply initializes all members to zero.

5.134.3 Member Data Documentation

5.134.3.1 estopDec

```
uunit estopDec
```

Deceleration used for emergency stop When the position loop is driving the velocity loop this value is only used for tracking error conditions.

If a tracking error occurs, the velocity loop takes over control and drives to zero velocity using this deceleration value.

Setting this value to zero indicates that the deceleration is unlimited.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.134.3.2 maxAcc

```
uunit maxAcc
```

Maximum allowed acceleration.

This value limits the rate of change of the velocity command input to the velocity loop. It is used when the magnitude of the command is increasing.

Note that the acceleration & deceleration limits are NOT used when the position loop is driving the velocity loop.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.134.3.3 maxDec

```
uunit maxDec
```

Maximum allowed deceleration This value limits the rate of change of the velocity command input to the velocity loop.

It is used when the magnitude of the command is decreasing.

Note that the acceleration & deceleration limits are not used when the position loop is driving the velocity loop.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.134.3.4 maxVel

```
uunit maxVel
```

Maximum allowed velocity.

This value is used to limit the velocity command before the velocity loop uses it to calculate output current. When running in a position mode (normal for CAN operation) The velocity command is the output from the position loop. This command is clipped by this value before it is passed to the velocity loop.

This parameter is specified in "user units". See Amp::SetCountsPerUnit for details.

5.134.3.5 shift

```
int16 shift
```

Output shift value.

The output of the loop is downshifted this many bits to get the current loop command

5.134.3.6 velCmdff

```
int16 velCmdff
```

Velocity loop command feed forward The input command (after limiting) to the velocity loop is scaled by this value and added to the output of the velocity loop.

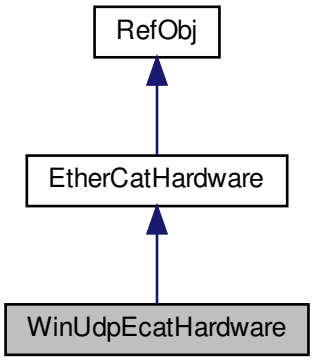
The documentation for this struct was generated from the following file:

- CML_AmpStruct.h

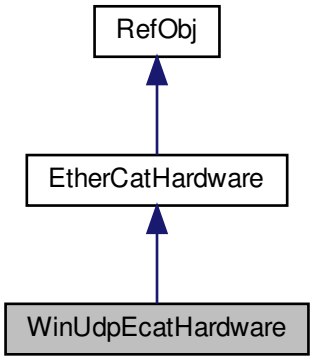
5.135 WinUdpEcatHardware Class Reference

This class provides an interface to the Ethernet ports on a windows system.

Inheritance diagram for WinUdpEcatHardware:



Collaboration diagram for WinUdpEcatHardware:



Public Member Functions

- WinUdpEcatHardware (const char *name=0)

  Create an EtherCAT hardware interface which uses UDP formatted messages.

Additional Inherited Members

5.135.1 Detailed Description

This class provides an interface to the Ethernet ports on a windows system.

It can be used to send and received formatted EtherCAT packets.

Windows doesn't allow raw Ethernet packets to be sent, so this class packages the EtherCAT packets into a UDP wrapper.

5.135.2 Constructor & Destructor Documentation

5.135.2.1 WinUdpEcatHardware()

```
WinUdpEcatHardware (
            const char * name = 0 )
```

Create an EtherCAT hardware interface which uses UDP formatted messages.

This is the only type of EtherCAT interface that can be used under Windows without installing special drivers.

The low level EtherCAT protocol normally does not use an IP address, however since this driver transmits EtherCAT packets over UDP/IP, the Ethernet interface used with this driver must have a valid IP address assigned. In addition, the network mask associated with the Ethernet interface should be defined in such a way that no other network interface on the same PC is a member of the same network. That is, if multiple interfaces are installed then they should be allocated to seperate networks.

i.e. (IP1 & mask1) != (IP2 & mask2) where IP1 and mask1 are the IP address and net mask of the first interface, and IP2 and mask2 are for the second interface.

For example, the following two interfaces are on different networks: IP: 192.168.1.1 mask: 255.255.255.0 IP: 192.↩ 168.2.1 mask: 255.255.255.0

but the following two interfaces are on the same network: IP: 192.168.1.1 mask: 255.255.255.0 IP: 192.168.1.2 mask: 255.255.255.0

This is important because this drive has no direct control of which interface the packets are being sent out. This is entirely controlled by the upper layer routing algorithms in the windows network stack.

The name parameter passed to this function can be used to identify which interface this object should bind to. It can take any of the following forms:

- If not specified, then the first valid interface found will be used. This is useful if there's only one interface on the PC.
- If of the form; eth0, eth1, eth2, etc, then the nth valid interface will be used.
- For more control, the IP address of the desired interface can be passed. This should be sent as a string in dotted decimal notation. For example: "192.168.1.1"

Parameters

name	Used to identify the Ethernet interface as described above.

The documentation for this class was generated from the following files:

- ecat_winudp.h
- ecat_winudp.cpp

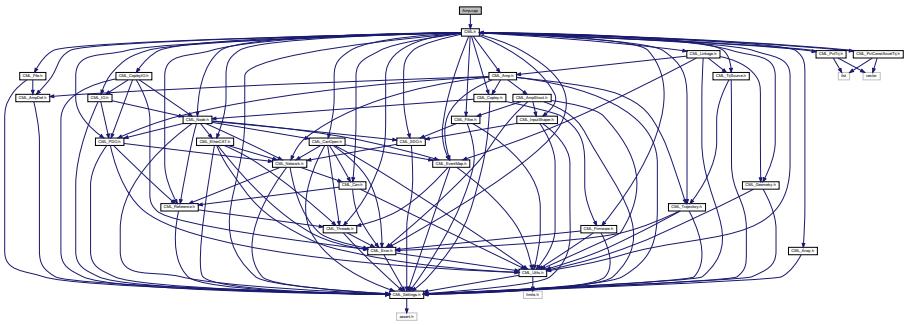
Chapter 6

File Documentation

6.1 Amp.cpp File Reference

This file provides most of the implementation for the Copley Amplifier object.

Include dependency graph for Amp.cpp:



6.1.1 Detailed Description

This file provides most of the implementation for the Copley Amplifier object.

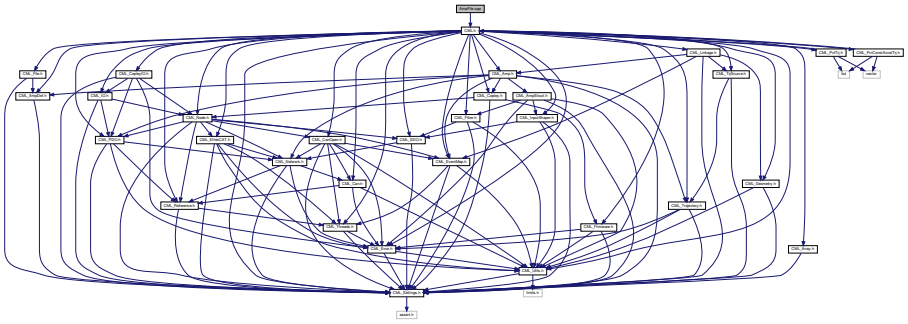
Since the Amp object is large and complex, it's member functions have been split into several files:

- This file: Contains the core code.
- AmpParam.cpp: Holds functions used to upload and download various blocks of amplifier parameters.
- AmpPDO.cpp: Contains functions used to implement the various PDO objects used in conjunction with the Amp object.

6.2 AmpFile.cpp File Reference

This file contains code used to read CME-2 .ccx and .ccd amplifier files.

Include dependency graph for AmpFile.cpp:



Functions

- const Error ∗ WriteCCDToEcatDrive (const char ∗name, EtherCAT &net, Amp &amp)

  Reads the ccd file from the drive using File Access over EtherCAT and creates a file.
- const Error ∗ WriteCCDToCANDrive (const char ∗name, CanOpen &net, Amp &amp)

  Writes the ccd file to the drive using serial binary over CANopen.
- long GetCCDFileSize (FILE ∗fp)

  Gets the file size.

6.2.1 Detailed Description

This file contains code used to read CME-2 .ccx and .ccd amplifier files.

6.2.2 Function Documentation

6.2.2.1 GetCCDFileSize()

```
long GetCCDFileSize (
            FILE * fp )
```

Gets the file size.

The passed file needs to be opened in binary mode.

Parameters

fp	opened file to be passed.

Returns

The size of the file in bytes.

6.2.2.2 WriteCCDToCANDrive()

```
const Error * WriteCCDToCANDrive (
            const char * name,
            CanOpen & net,
            Amp & amp )
```

Writes the ccd file to the drive using serial binary over CANopen.

Parameters

name	Name of the file to be passed.
net	Reference to the CANopen network object.
amp	Reference to the amplifier.

Returns

The most recent error code returned by this node.

6.2.2.3 WriteCCDToEcatDrive()

```
const Error * WriteCCDToEcatDrive (
            const char * name,
            EtherCAT & net,
            Amp & amp )
```

Reads the ccd file from the drive using File Access over EtherCAT and creates a file.

Parameters

name	Name of the file to be created.
net	Reference to the EtherCAT network object.
amp	Reference to the amplifier.

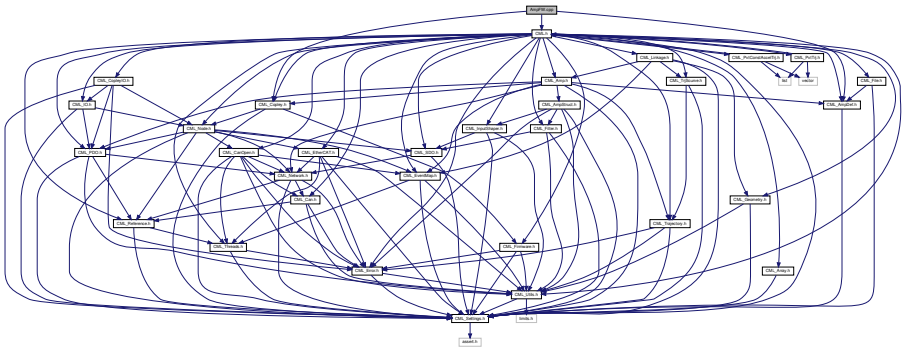
Returns

The most recent error code returned by this node.

6.3 AmpFW.cpp File Reference

This file contains code used to update an amplifier's firmware over the CANopen network.

Include dependency graph for AmpFW.cpp:



6.3.1 Detailed Description

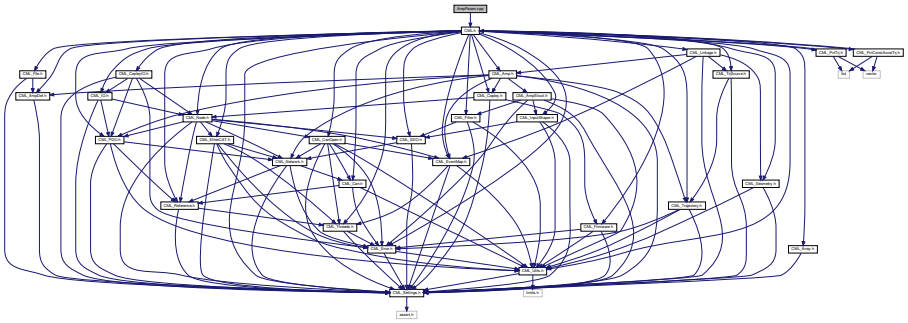
This file contains code used to update an amplifier's firmware over the CANopen network.

The firmware update protocol is a non-standard protocol which uses CAN messages with the same IDs as the node's CANopen SDO protocol. These message IDs were used because they are fixed for a given node ID.

6.4 AmpParam.cpp File Reference

This file contains the AMP object methods used to upload / download various amplifier parameters.

Include dependency graph for AmpParam.cpp:



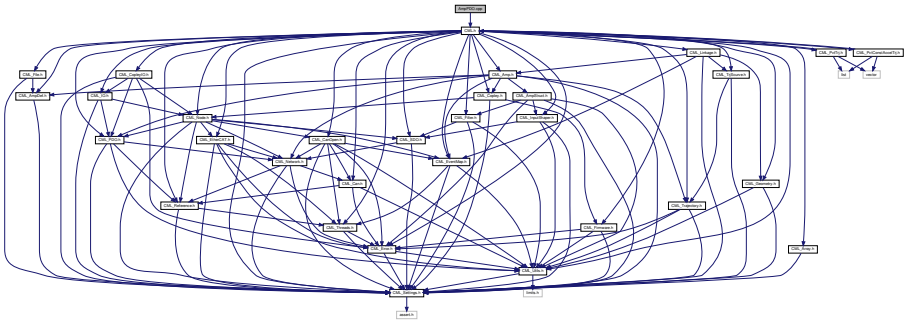
6.4.1 Detailed Description

This file contains the AMP object methods used to upload / download various amplifier parameters.

6.5 AmpPDO.cpp File Reference

This file contains code that implements PDO objects used by the Copley Controls amplifier object.

Include dependency graph for AmpPDO.cpp:



Macros

- #define MAX_PENDING_PVT_STATUS 50

  This thread is used to handle the PVT status updates for all amps.

6.5.1 Detailed Description

This file contains code that implements PDO objects used by the Copley Controls amplifier object.

6.5.2 Macro Definition Documentation

6.5.2.1 MAX_PENDING_PVT_STATUS

```
#define MAX_PENDING_PVT_STATUS 50
```

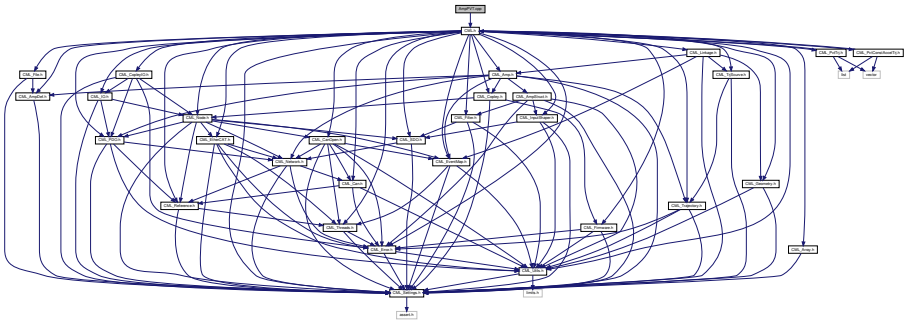
This thread is used to handle the PVT status updates for all amps.

A separate thread is necessary, because EtherCAT drives need to send SDOs when handling PVT status updates, and SDOs can't be used from within the context of the network read thread.

6.6 AmpPVT.cpp File Reference

This file contains the code used by the Amp object to stream PVT trajectory profiles over the CANopen network.

Include dependency graph for AmpPVT.cpp:



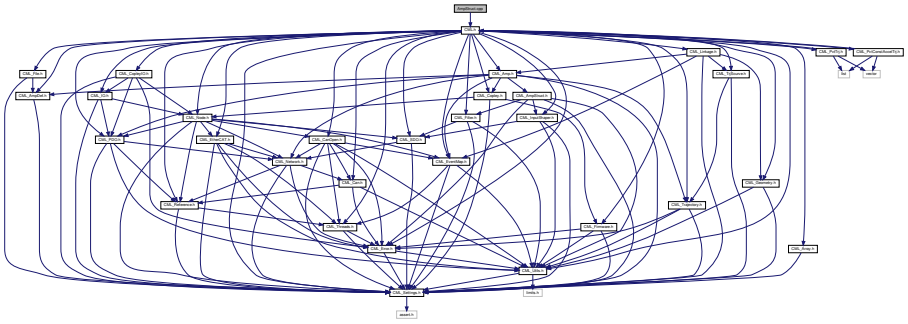
6.6.1 Detailed Description

This file contains the code used by the Amp object to stream PVT trajectory profiles over the CANopen network.

6.7 AmpStruct.cpp File Reference

This file contains the AMP object methods used to upload / download structures containing groups of amplifier parameters.

Include dependency graph for AmpStruct.cpp:



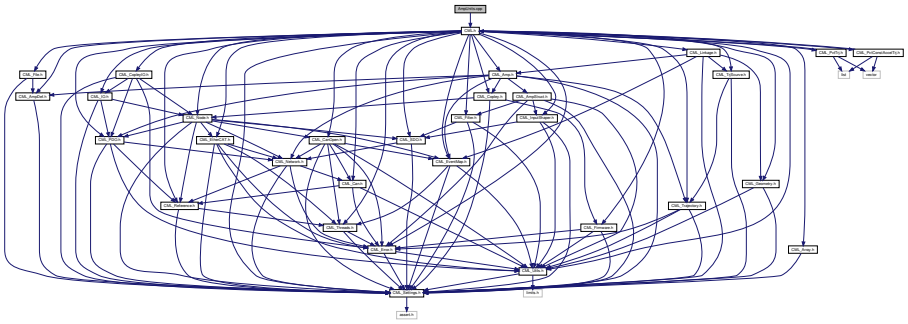
6.7.1 Detailed Description

This file contains the AMP object methods used to upload / download structures containing groups of amplifier parameters.

6.8 AmpUnits.cpp File Reference

This file contains the AMP object methods used to handle unit conversions.

Include dependency graph for AmpUnits.cpp:



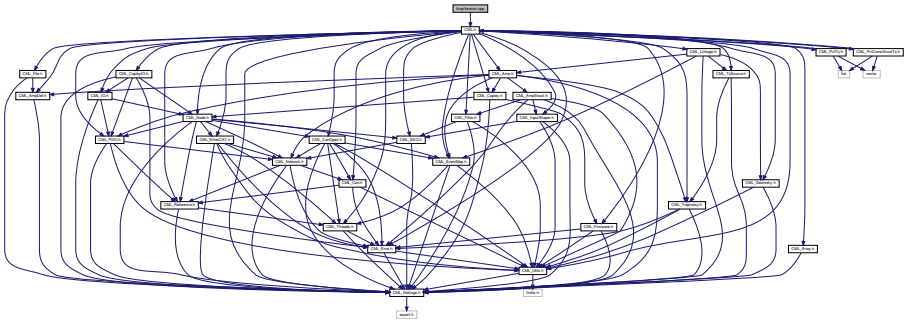
6.8.1 Detailed Description

This file contains the AMP object methods used to handle unit conversions.

6.9 AmpVersion.cpp File Reference

This file contains some rules used by the Amp object to determine if certain features are supported by the amplifier based on it's model number and firmware version number.

Include dependency graph for AmpVersion.cpp:



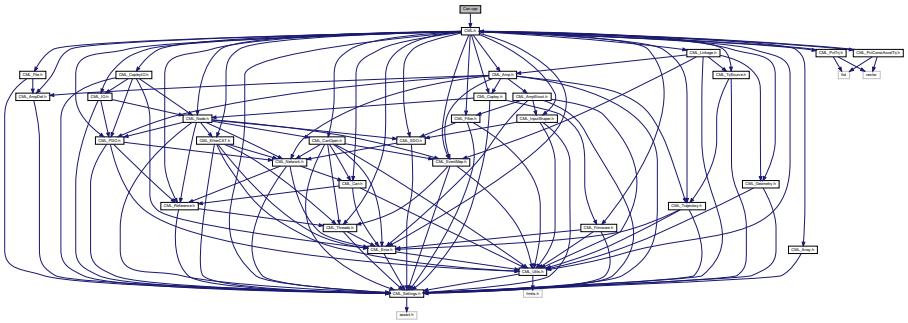
6.9.1 Detailed Description

This file contains some rules used by the Amp object to determine if certain features are supported by the amplifier based on it's model number and firmware version number.

6.10 Can.cpp File Reference

This file handles the initialization of the static variables (error codes) used by the CanError and CanInterface classes.

Include dependency graph for Can.cpp:



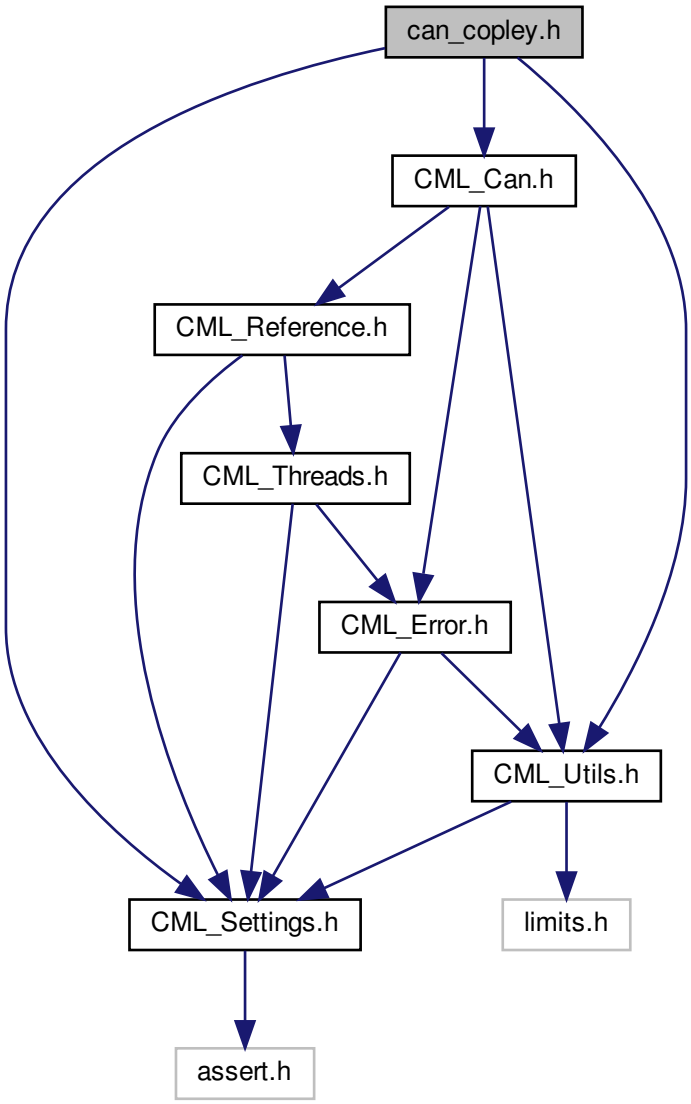
6.10.1 Detailed Description

This file handles the initialization of the static variables (error codes) used by the CanError and CanInterface classes.

6.11 can_copley.h File Reference

CAN hardware interface for the Copley Controls CAN card.

Include dependency graph for can_copley.h:



Classes

- class CopleyCAN

  This class extends the generic CanInterface class into a working interface for the Copley can device driver.

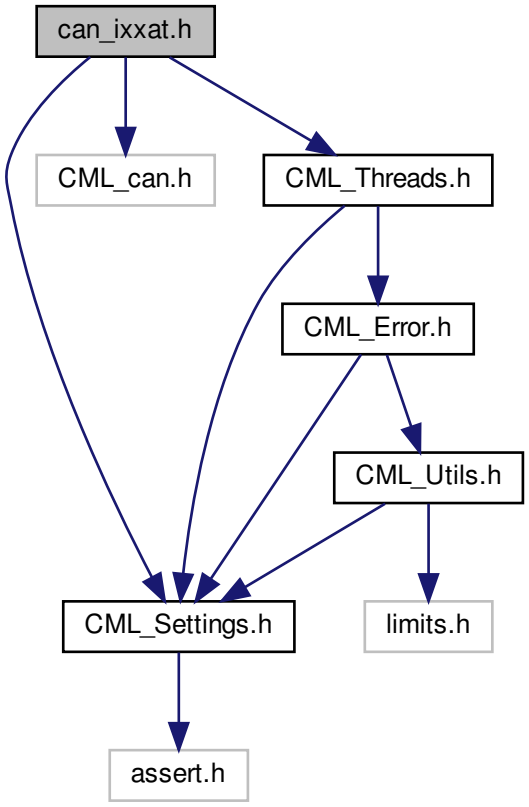
6.11.1 Detailed Description

CAN hardware interface for the Copley Controls CAN card.

6.12 can_ixxat.h File Reference

CAN hardware interface for the Ixxat CAN driver.

Include dependency graph for can_ixxat.h:



Classes

- class IxxatCAN

  Ixxat specific CAN interface.

Macros

- #define IXXAT_RX_QUEUE_SZ 50

  This gives the size of the CAN message receive queue used for Ixxat cards.

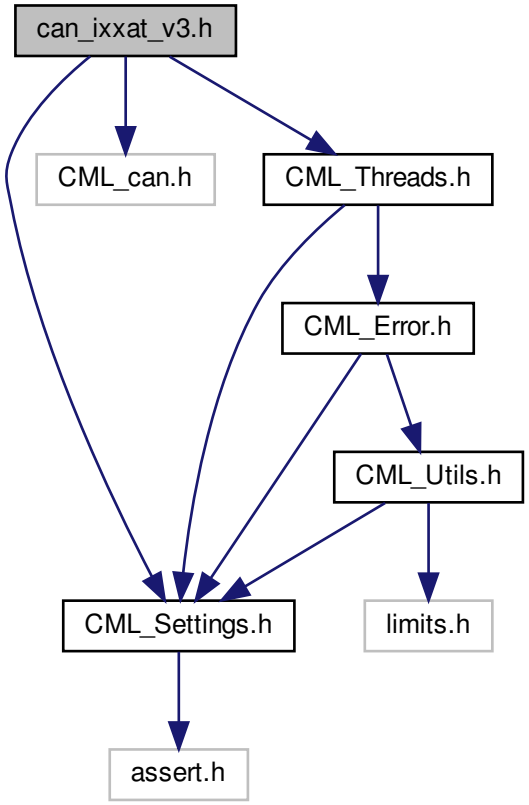
6.12.1 Detailed Description

CAN hardware interface for the Ixxat CAN driver.

6.13 can_ixxat_v3.h File Reference

CAN hardware interface for the Ixxat CAN driver.

Include dependency graph for can_ixxat_v3.h:



Classes

- class IxxatCANV3

  Ixxat specific CAN interface.

Macros

- #define IXXAT_RX_QUEUE_SZ 50

  This gives the size of the CAN message receive queue used for Ixxat cards.

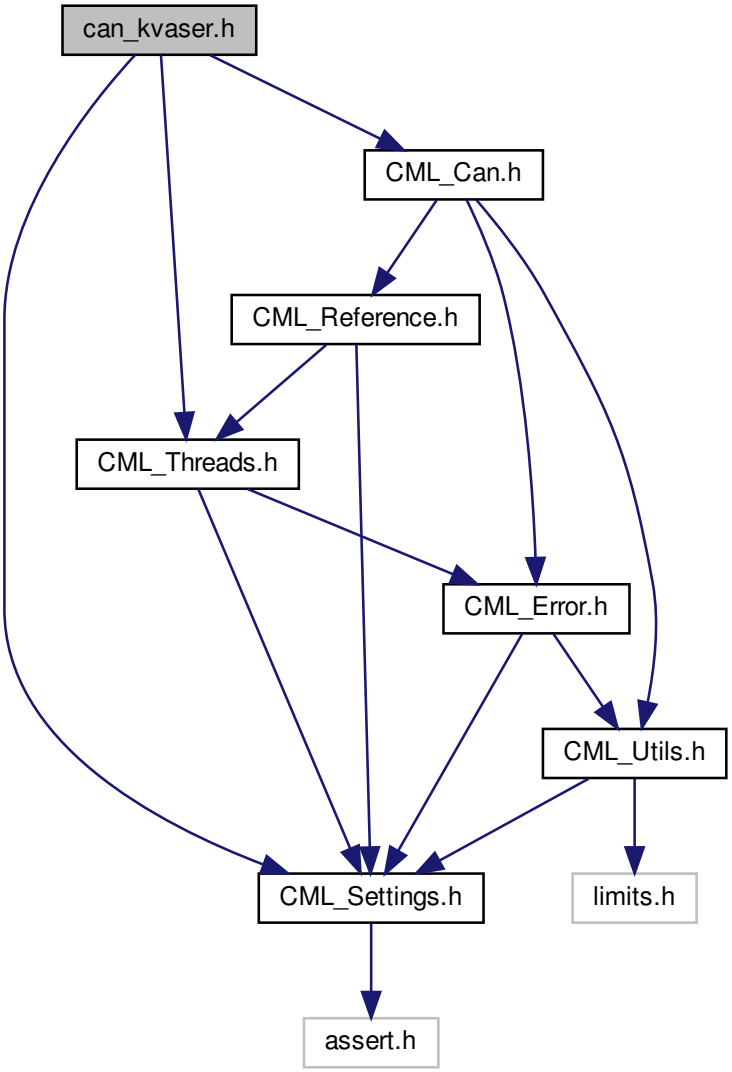
6.13.1 Detailed Description

CAN hardware interface for the Ixxat CAN driver.

6.14 can_kvaser.h File Reference

CAN hardware interface for the Kvaser CAN driver.

Include dependency graph for can_kvaser.h:



Classes

- class KvaserCAN

  Kvaser specific CAN interface.

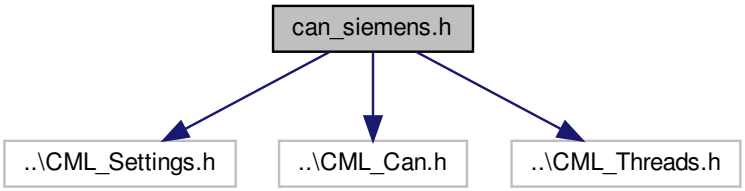
6.14.1 Detailed Description

CAN hardware interface for the Kvaser CAN driver.

6.15 can_siemens.h File Reference

CAN hardware interface for the SiemensPCIeToCAN CAN driver.

Include dependency graph for can_siemens.h:



Classes

- class SiemensCAN

  SiemensPCIeToCAN specific CAN interface.

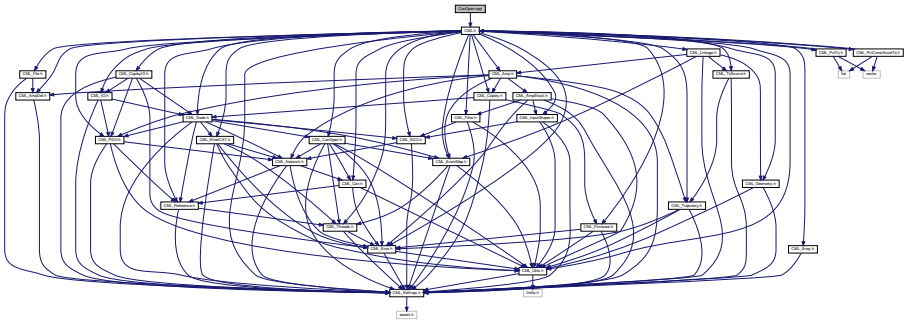
6.15.1 Detailed Description

CAN hardware interface for the SiemensPCIeToCAN CAN driver.

6.16 CanOpen.cpp File Reference

This file holds code for the top level CANopen class.

Include dependency graph for CanOpen.cpp:



6.16.1 Detailed Description

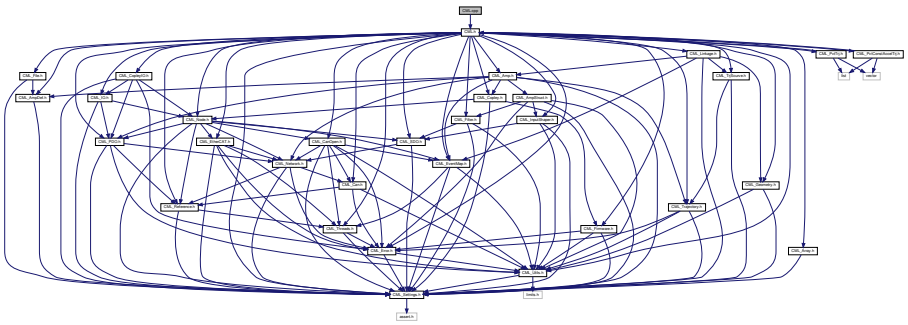
This file holds code for the top level CANopen class.

This class is used for over all control of the CANopen network.

6.17 CML.cpp File Reference

CML object definition.

Include dependency graph for CML.cpp:



Variables

- CopleyMotionLibrary cml

  Global CML object.

6.17.1 Detailed Description

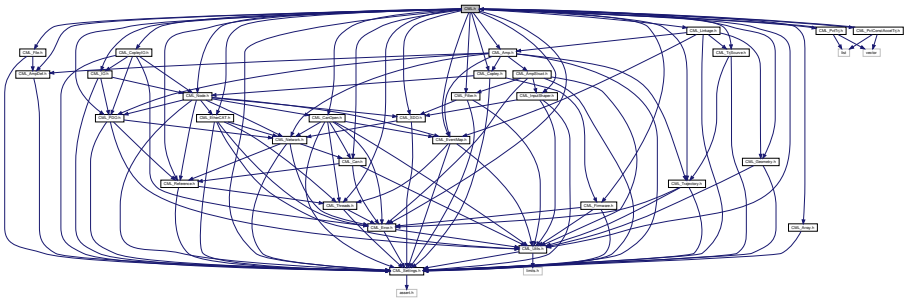
CML object definition.

This file contains the code used to implement the global CML object.

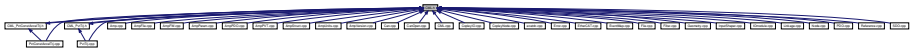
6.18 CML.h File Reference

Top level include file for the CML libraries.

Include dependency graph for CML.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CopleyMotionLibrary

  Copley Motion Libraries utility object.

Enumerations

- enum CML_LOG_LEVEL {
  LOG_NONE = 0,
  LOG_ERRORS = 1,
  LOG_WARNINGS = 2,
  LOG_DEBUG = 3,
  LOG_FILT_CAN = 5,
  LOG_CAN = 6,
  LOG_EVERYTHING = 99 }

  Copley Motion Libraries debug logging level.

Variables

- CopleyMotionLibrary cml

  Global CML object.

6.18.1 Detailed Description

Top level include file for the CML libraries.

This file serves two purposes; it includes all the other CML header files and it defines the CML object. The CML object contains a number of utility methods dealing with the library as a whole.

6.18.2 Enumeration Type Documentation

6.18.2.1 CML_LOG_LEVEL

```
enum CML_LOG_LEVEL
```

Copley Motion Libraries debug logging level.

The CML libraries may be configured to generate a log file for use in debugging system problems. This feature is turned off by default, but may be enabled by calling the method CopleyMotionLibrary::SetDebugLevel of the global cml object.

```
cml.SetDebugLevel( LOG_EVERYTHING );
```

This enumeration gives the logging levels that may be passed to the SetDebugLevel function. Debug logging levels are cumulative, so enabling a high level of logging will cause all messages that would have been logged at a lower level to be written to the log as well. For example, setting the log level to LOG_DEBUG will cause all debug messages to be written to the log, as well as all warnings and errors.

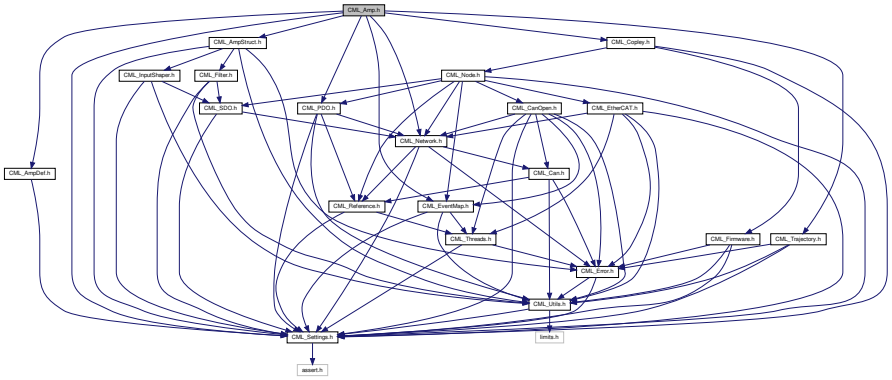
Enumerator

LOG_NONE	Debug logging is disabled.
LOG_ERRORS	Log serious errors only.
LOG_WARNINGS	Log warning messages and errors.
LOG_DEBUG	Log some debugging info.
LOG_FILT_CAN	Log most CAN messages. A few common messages are filtered out.
LOG_CAN	Log all CAN messages.
LOG_EVERYTHING	Log everything.

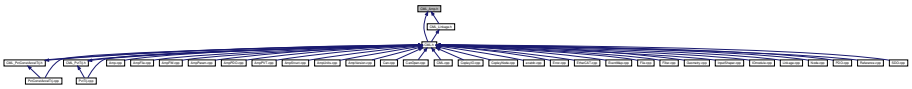
6.19 CML_Amp.h File Reference

This file defines the Copley Amplifier object.

Include dependency graph for CML_Amp.h:



This graph shows which files directly or indirectly include this file:



Classes

- class AmpError

  This class represents error conditions that can occur in the Copley Amplifier object.
- class AmpFault

  This class represents latching amplifier fault conditions.
- class AmpSettings

  Copley amplifier settings object.
- class PvtSegCache

  PVT trajectory segment cache object.
- struct ProfileConfigTrap

  Trapezoidal profile parameters.
- struct ProfileConfigScurve

  S-curve profile parameters.
- struct ProfileConfigVel

  Velocity profile parameters.
- class Amp

  Copley Controls amplifier object.

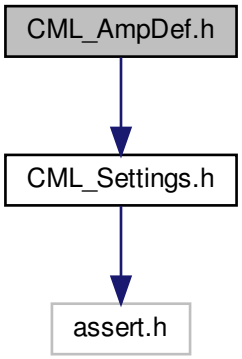
6.19.1 Detailed Description

This file defines the Copley Amplifier object.

This object may be used to interface to an amplifier over the CANopen network.

6.20 CML_AmpDef.h File Reference

Include dependency graph for CML_AmpDef.h:



This graph shows which files directly or indirectly include this file:

Enumerations

- enum AMP_OBJID

  This enumeration holds the object identifiers of all of the objects in the amplifier's object dictionary.

- enum INPUT_PIN_CONFIG {
  INCFG_NONE = 0x0000,
  INCFG_RESET_R = 0x0002,
  INCFG_RESET_F = 0x0003,
  INCFG_POSLIM_H = 0x0004,
  INCFG_POSLIM_L = 0x0005,
  INCFG_NEGLIM_H = 0x0006,
  INCFG_NEGLIM_L = 0x0007,
  INCFG_MOTOR_TEMP_H = 0x0008,
  INCFG_MOTOR_TEMP_L = 0x0009,
  INCFG_CLR_FAULTS_H = 0x000A,
  INCFG_CLR_FAULTS_L = 0x000B,
  INCFG_RESET_DISABLE_R = 0x000C,
  INCFG_RESET_DISABLE_F = 0x000D,
  INCFG_HOME_H = 0x000E,

INCFG_HOME_L = 0x000F,
INCFG_DISABLE_H = 0x0010,
INCFG_DISABLE_L = 0x0011,
INCFG_PWM_SYNC_H = 0x0013,
INCFG_MOTION_ABORT_H = 0x0014,
INCFG_MOTION_ABORT_L = 0x0015,
INCFG_SCALE_ADC_H = 0x0016,
INCFG_SCALE_ADC_L = 0x0017,
INCFG_HIGHSPEED_CAPTURE_R = 0x0018,
INCFG_HIGHSPEED_CAPTURE_F = 0x0019,
INCFG_COUNT_EDGES_R = 0x001A,
INCFG_COUNT_EDGES_F = 0x001B,
INCFG_ENCODER_FAULT_H = 0x001C,
INCFG_ENCODER_FAULT_L = 0x001D,
INCFG_ABORT_WINDOW_R = 0x0024,
INCFG_ABORT_WINDOW_F = 0x0025,
INCFG_HV_LOSS_DISABLE_H = 0x0026,
INCFG_HV_LOSS_DISABLE_L = 0x0027,
INCFG_TRJ_UPDATE_R = 0x0028,
INCFG_TRJ_UPDATE_F = 0x0029,
INCFG_CLR_FAULTS_EVENTS_R = 0x002A,
INCFG_CLR_FAULTS_EVENTS_F = 0x002B,
INCFG_DIS_SIM_ENC_L_BURST_R = 0x002C,
INCFG_DIS_SIM_ENC_H_BURST_F = 0x002D }

Input pin configuration settings.

- enum OUTPUT_PIN_CONFIG {
OUTCFG_EVENT_STATUS_L = 0x0000,
OUTCFG_EVENT_STATUS_H = 0x0100,
OUTCFG_EVENT_LATCH_L = 0x0001,
OUTCFG_EVENT_LATCH_H = 0x0101,
OUTCFG_MANUAL_L = 0x0002,
OUTCFG_MANUAL_H = 0x0102,
OUTCFG_TRJ_STATUS = 0x0003,
OUTCFG_POSITION_WINDOW = 0x0004,
OUTCFG_POSITION_TRIG_LOW2HIGH = 0x0005,
OUTCFG_POSITION_TRIG_HIGH2LOW = 0x0006,
OUTCFG_POSITION_TRIG = 0x0007,
OUTCFG_POSITION_TRIG_LIST = 0x0009,
OUTCFG_SYNC_OUTPUT = 0x0200,
OUTCFG_ACTIVE_HIGH = 0x0100 }

  Output pin configuration settings.

- enum AMP_MODE {
AMPMODE_CAN_PROFILE = 0x0001,
AMPMODE_CAN_VELOCITY = 0x0003,
AMPMODE_CAN_TORQUE = 0x0004,
AMPMODE_CAN_HOMING = 0x0006,
AMPMODE_CAN_PVT = 0x0007,
AMPMODE_CAN_SERVO = 0x1E00,
AMPMODE_CAN_USTEP = 0x2800,
AMPMODE_DISABLED = 0x0000,
AMPMODE_PROG_CRNT = 0x0100,
AMPMODE_AIN_CRNT = 0x0200,
AMPMODE_DIN_CRNT = 0x0300,
AMPMODE_FGEN_CRNT = 0x0400,

AMPMODE_PROG_VEL = 0x0B00,
AMPMODE_AIN_VEL = 0x0C00,
AMPMODE_DIN_VEL = 0x0D00,
AMPMODE_FGEN_VEL = 0x0E00,
AMPMODE_DIN_POS = 0x1700,
AMPMODE_FGEN_POS = 0x1800,
AMPMODE_CAM_POS = 0x1900,
AMPMODE_DIN_USTEP = 0x2100,
AMPMODE_FGEN_USTEP = 0x2200,
AMPMODE_DIAG_USTEP = 0x2A00 }

This enumeration is used to specify the mode of operation of the amplifier.

- enum EVENT_STATUS {
ESTAT_SHORT_CRCT = 0x00000001,
ESTAT_AMP_TEMP = 0x00000002,
ESTAT_OVER_VOLT = 0x00000004,
ESTAT_UNDER_VOLT = 0x00000008,
ESTAT_MTR_TEMP = 0x00000010,
ESTAT_ENCODER_PWR = 0x00000020,
ESTAT_PHASE_ERR = 0x00000040,
ESTAT_CRNT_LIM = 0x00000080,
ESTAT_VOLT_LIM = 0x00000100,
ESTAT_POSLIM = 0x00000200,
ESTAT_NEGLIM = 0x00000400,
ESTAT_DISABLE_INPUT = 0x00000800,
ESTAT_SOFT_DISABLE = 0x00001000,
ESTAT_STOP = 0x00002000,
ESTAT_BRAKE = 0x00004000,
ESTAT_PWM_DISABLE = 0x00008000,
ESTAT_SOFTLIM_POS = 0x00010000,
ESTAT_SOFTLIM_NEG = 0x00020000,
ESTAT_TRK_ERR = 0x00040000,
ESTAT_TRK_WARN = 0x00080000,
ESTAT_RESET = 0x00100000,
ESTAT_POSWRAP = 0x00200000,
ESTAT_FAULT = 0x00400000,
ESTAT_VEL_LIMIT = 0x00800000,
ESTAT_ACC_LIMIT = 0x01000000,
ESTAT_TRK_WIN = 0x02000000,
ESTAT_HOME = 0x04000000,
ESTAT_MOVING = 0x08000000,
ESTAT_VEL_WIN = 0x10000000,
ESTAT_PHASE_INIT = 0x20000000,
ESTAT_CMD_INPUT = 0x40000000 }

Amplifier event status word bit definitions.

- enum AMP_EVENT {
AMPEVENT_MOVEDONE = 0x00000001,
AMPEVENT_TRJDONE = 0x00000002,
AMPEVENT_NODEGUARD = 0x00000004,
AMPEVENT_SPACK = 0x00000008,
AMPEVENT_FAULT = 0x00000010,
AMPEVENT_ERROR = 0x00000020,
AMPEVENT_POSWARN = 0x00000040,
AMPEVENT_POSWIN = 0x00000080,
AMPEVENT_VELWIN = 0x00000100,

AMPEVENT_DISABLED = 0x00000200,
AMPEVENT_POSLIM = 0x00000400,
AMPEVENT_NEGLIM = 0x00000800,
AMPEVENT_SOFTLIM_POS = 0x00001000,
AMPEVENT_SOFTLIM_NEG = 0x00002000,
AMPEVENT_QUICKSTOP = 0x00004000,
AMPEVENT_ABORT = 0x00008000,
AMPEVENT_SOFTDISABLE = 0x00010000,
AMPEVENT_HOME_CAPTURE = 0x00020000,
AMPEVENT_PVT_EMPTY = 0x00040000,
AMPEVENT_PHASE_INIT = 0x00080000,
AMPEVENT_TRSTP = 0x00100000,
AMPEVENT_NOT_INIT = 0x80000000 }

Amplifier events.

- enum AMP_FAULT {
FAULT_DATAFLASH = 0x0001,
FAULT_ADCOFFSET = 0x0002,
FAULT_SHORT_CRCT = 0x0004,
FAULT_AMP_TEMP = 0x0008,
FAULT_MTR_TEMP = 0x0010,
FAULT_OVER_VOLT = 0x0020,
FAULT_UNDER_VOLT = 0x0040,
FAULT_ENCODER_PWR = 0x0080,
FAULT_PHASE_ERR = 0x0100,
FAULT_TRK_ERR = 0x0200,
FAULT_I2T_ERR = 0x0400 }

  Latching Amplifier faults conditions.

- enum HALT_MODE {
HALT_DISABLE = 0,
HALT_DECEL = 1,
HALT_QUICKSTOP = 2,
HALT_ABRUPT = 3 }

  The amplifier's halt mode defines it's action when a halt command is issued (Amp::HaltMove).

- enum QUICK_STOP_MODE {
QSTOP_DISABLE = 0,
QSTOP_DECEL = 1,
QSTOP_QUICKSTOP = 2,
QSTOP_ABRUPT = 3,
QSTOP_DECEL_HOLD = 5,
QSTOP_QUICKSTOP_HOLD = 6,
QSTOP_ABRUPT_HOLD = 7 }

  The amplifier's quick stop mode defines it's action when a quick stop command is issued (Amp::QuickStop).

- enum COPLEY_HOME_METHOD {
CHM_NLIM_ONDX = 1,
CHM_PLIM_ONDX = 2,
CHM_PHOME_ONDX = 3,
CHM_PHOME_INDX = 4,
CHM_NHOME_ONDX = 5,
CHM_NHOME_INDX = 6,
CHM_LHOME_ONDX_POS = 7,
CHM_LHOME_INDX_POS = 8,
CHM_UHOME_INDX_POS = 9,
CHM_UHOME_ONDX_POS = 10,

CHM_UHOME_ONDX_NEG = 11,
CHM_UHOME_INDX_NEG = 12,
CHM_LHOME_INDX_NEG = 13,
CHM_LHOME_ONDX_NEG = 14,
CHM_NLIM = 17,
CHM_PLIM = 18,
CHM_PHOME = 19,
CHM_NHOME = 21,
CHM_LHOME_POS = 23,
CHM_UHOME_POS = 25,
CHM_UHOME_NEG = 27,
CHM_LHOME_NEG = 29,
CHM_NDX_NEG = 33,
CHM_NDX_POS = 34,
CHM_NONE = 35,
CHM_HARDSTOP_POS = 255,
CHM_HARDSTOP_NEG = 254,
CHM_HARDSTOP_ONDX_POS = 253,
CHM_HARDSTOP_ONDX_NEG = 252,
CHM_EXTENDED = 128 }

Home methods supported by the Copley amplifier.

- enum PROFILE_TYPE {
PROFILE_VEL = -1,
PROFILE_TRAP = 0,
PROFILE_SCURVE = 3 }

  Point to point profile types.

- enum AMP_PHASE_MODE {
PHASE_MODE_ENCODER = 0,
PHASE_MODE_TRAP = 1,
PHASE_MODE_NOADJUST = 2,
PHASE_MODE_AHALL90 = 3,
PHASE_MODE_BRUSHED = 4,
PHASE_MODE_NOHALL = 5,
PHASE_MODE_ENCPHASE = 6,
PHASE_MODE_TRAPINTERP = 7 }

  Amplifier phasing mode.

- enum AMP_PWM_MODE {
PWM_MODE_STANDARD = 0x0000,
PWM_MODE_FORCECLAMP = 0x0001,
PWM_MODE_AUTOCLAMP = 0x0002,
PWM_MODE_HEXLIMIT = 0x0010 }

  Amplifier PWM output mode.

- enum AMP_TRACE_VAR {
TRACEVAR_HIGH_VOLT = 6,
TRACEVAR_TEMP = 37,
TRACEVAR_ANALOG_REF = 5,
TRACEVAR_ENC_SIN = 46,
TRACEVAR_ENC_COS = 47,
TRACEVAR_PHASE = 36,
TRACEVAR_HALLS = 40,
TRACEVAR_INPUTS = 48,
TRACEVAR_RAW_INPUTS = 33,
TRACEVAR_EVENTS = 38,

TRACEVAR_EVENTLATCH = 39,
TRACEVAR_CRNT_A = 3,
TRACEVAR_CRNT_B = 4,
TRACEVAR_CRNT_CMD = 7,
TRACEVAR_CRNT_LIM = 8,
TRACEVAR_CRNT_CMD_D = 9,
TRACEVAR_CRNT_CMD_Q = 10,
TRACEVAR_CRNT_ACT_D = 13,
TRACEVAR_CRNT_ACT_Q = 14,
TRACEVAR_CRNT_ERR_D = 15,
TRACEVAR_CRNT_ERR_Q = 16,
TRACEVAR_VOLT_D = 19,
TRACEVAR_VOLT_Q = 20,
TRACEVAR_VEL_MTR = 23,
TRACEVAR_VEL_RAW = 50,
TRACEVAR_VEL_LOAD = 43,
TRACEVAR_VLOOP_CMD = 24,
TRACEVAR_VLOOP_LIM = 25,
TRACEVAR_VLOOP_ERR = 26,
TRACEVAR_LOAD_POS = 28,
TRACEVAR_MTR_POS = 31,
TRACEVAR_POS_ERR = 30,
TRACEVAR_CMD_POS = 29,
TRACEVAR_CMD_VEL = 44,
TRACEVAR_CMD_ACC = 45,
TRACEVAR_DEST_POS = 49 }

Amplifier trace variables.

- enum AMP_TRACE_STATUS

  Amplifier trace status bits.

- enum AMP_TRACE_TRIGGER {
  TRACETRIG_CHANNEL = 0x000F,
  TRACETRIG_TYPE = 0x0F00,
  TRACETRIG_NONE = 0x0000,
  TRACETRIG_ABOVE = 0x0100,
  TRACETRIG_BELOW = 0x0200,
  TRACETRIG_RISE = 0x0300,
  TRACETRIG_FALL = 0x0400,
  TRACETRIG_BITSET = 0x0500,
  TRACETRIG_BITCLR = 0x0600,
  TRACETRIG_CHANGE = 0x0700,
  TRACETRIG_EVENTSET = 0x0800,
  TRACETRIG_EVENTCLR = 0x0900,
  TRACETRIG_FGEN_CYCLE = 0x0A00,
  TRACETRIG_CAP_SET = 0x0B00,
  TRACETRIG_BITCHANGE = 0x0C00,
  TRACETRIG_NODELAY = 0x4000,
  TRACETRIG_SAMPLE = 0x8000 }

  Amplifier trace trigger settings.

- enum POS_CAPTURE_CFG {
  CAPTURE_INDEX_RISING = 0x0001,
  CAPTURE_INDEX_FALLING = 0x0002,
  CAPTURE_INDEX_LATCH = 0x0004,
  CAPTURE_HOME_LATCH = 0x0040,

CAPTURE_HIGH_SPEED_INPUT = 0x0100,
CAPTURE_HIGH_SPEED_INPUT_LATCH = 0x0400 }

Position capture configuration.

- enum POS_CAPTURE_STAT {
CAPTURE_INDEX_FULL = 0x0001,
CAPTURE_INDEX_OVER = 0x0008,
CAPTURE_HOME_FULL = 0x0010,
CAPTURE_HOME_OVER = 0x0080 }

  Position capture status register value.

- enum AMP_FEATURE { ,
FEATURE_EXTENDED_OUTPUT_PIN_CONFIG,
FEATURE_GEAR_RATIO,
FEATURE_RESOLVER_CYCLES,
FEATURE_HALL_VEL_SHIFT,
FEATURE_PLOOP_SCALE,
FEATURE_STEPPER_CRNT,
FEATURE_CURRENT_SLOPE,
FEATURE_SOFTLIM_ACCEL,
FEATURE_PWMIN_FREQ,
FEATURE_VLOOP_CMD_FILT,
FEATURE_USTEP_OUTER_LOOP,
FEATURE_STEP_DETENT_GAIN,
FEATURE_USTEP_CONFIG_STATUS,
FEATURE_ALGO_PHASE_INIT_CUR,
FEATURE_ALGO_PHASE_INIT_CONFIG,
FEATURE_CAMMING,
FEATURE_POS_WRAP,
FEATURE_ENC_OPTIONS,
FEATURE_GAIN_SCHED,
FEATURE_PIN_MAP,
FEATURE_CAN_OPTIONS,
FEATURE_CAN_SETTINGS,
FEATURE_AIN_FILT,
FEATURE_VLOOP_OUT_FILT,
FEATURE_ILOOP_CMD_FILT,
FEATURE_PWMIN_MIN_PULSE,
FEATURE_PWMIN_MAX_PULSE,
FEATURE_DA_CONV_CONFIG,
FEATURE_SERVO_CONFIG,
FEATURE_MTR_OVERTEMP,
FEATURE_NET_OPTIONS,
FEATURE_PLOOP_KI,
FEATURE_PLOOP_KD,
FEATURE_VLOOP_CMDFF,
FEATURE_BRAKE_ENABLE_DELAY,
FEATURE_INPUT_SHAPING,
FEATURE_FLOAT_FILT_COEF,
FEATURE_IO_OPTIONS,
FEATURE_PWMIN_UVCFG,
FEATURE_AXIS_CT }

  This enumeration is used internally by the amplifier object to check for certain features that are not present in every amp model or firmware version number.

6.20.1 Enumeration Type Documentation

6.20.1.1 AMP_EVENT

```
enum AMP_EVENT
```

Amplifier events.

This enumeration provides a list of events that can be used to wait on amplifier conditions.

Enumerator

AMPEVENT_MOVEDONE	Set when a move is finished and the amplifier has settled in to position at the end of the move. Cleared when a new move is started.
AMPEVENT_TRJDONE	Set when the trajectory generator finishes a move. The motor may not have settled into position at this point. Cleared when a new move is started.
AMPEVENT_NODEGUARD	A node guarding (or heartbeat) error has occurred. This indicates that the amplifier failed to respond within the expected amount of time for either a heartbeat or node guarding message. This could be caused by a network wiring problem, amplifier power down, amp reset, etc. This bit is set when the error occurs, and is cleared by a call to the function Amp::ClearNodeGuardEvent.
AMPEVENT_SPACK	This event bit is used internally by the amplifier object. It is set when the amp acknowledges a new move start.
AMPEVENT_FAULT	A latching amplifier fault has occurred. The specifics of what caused the fault can be obtained by calling Amp::GetFaults, and the fault conditions can be cleared by calling Amp::ClearFaults.
AMPEVENT_ERROR	A non-latching amplifier error has occurred.
AMPEVENT_POSWARN	The amplifier's absolute position error is greater then the window set with Amp::SetPositionWarnWindow.
AMPEVENT_POSWIN	The amplifier's absolute position error is greater then the window set with Amp::SetSettlingWindow.
AMPEVENT_VELWIN	The amplifier's absolute velocity error is greater then the window set with Amp::SetVeliocityWarnWindow.
AMPEVENT_DISABLED	The amplifier's outputs are disabled. The reason for the disable can be determined by Amp::GetEventStatus
AMPEVENT_POSLIM	The positive limit switch is currently active.
AMPEVENT_NEGLIM	The negative limit switch is currently active.
AMPEVENT_SOFTLIM_POS	The positive software limit is currently active.
AMPEVENT_SOFTLIM_NEG	The negative software limit is currently active.
AMPEVENT_QUICKSTOP	The amplifier is presently performing a quick stop sequence.
AMPEVENT_ABORT	The last profile was aborted without finishing.
AMPEVENT_SOFTDISABLE	The amplifier is software disabled.
AMPEVENT_HOME_CAPTURE	A new home position has been captured. Note that this features requires firmware version >= 4.77
AMPEVENT_PVT_EMPTY	PVT buffer is empty.

Enumerator

AMPEVENT_PHASE_INIT	Amplifier is currently performing a phase initialization.
AMPEVENT_TRSTP	Amplifier is currently trying to stop the motor.
AMPEVENT_NOT_INIT	This amplifier's event mask has not yet been initialized. This event is for internal use only.

6.20.1.2 AMP_FAULT

`enum AMP_FAULT`

Latching Amplifier faults conditions.

Once a fault is detected in the amplifier, the amp will be disabled until the fault condition has been cleared.

Use Amp::GetFaults to get a list of any active fault conditions, and Amp::ClearFaults to clear one or more faults.

Enumerator

FAULT_DATAFLASH	Fatal hardware error: the flash data is corrupt.
FAULT_ADCOFFSET	Fatal hardware error: An A/D offset error has occurred.
FAULT_SHORT_CRCT	The amplifier detected a short circuit condition.
FAULT_AMP_TEMP	The amplifier is over temperature.
FAULT_MTR_TEMP	A motor temperature error was detected.
FAULT_OVER_VOLT	The amplifier bus voltage is over the acceptable limit.
FAULT_UNDER_VOLT	The amplifier bus voltage is below the acceptable limit.
FAULT_ENCODER_PWR	Over current on the encoder power supply.
FAULT_PHASE_ERR	Amplifier phasing error.
FAULT_TRK_ERR	Tracking error, the position error is too large.
FAULT_I2T_ERR	Current limited by i^2t algorithm.

6.20.1.3 AMP_FEATURE

`enum AMP_FEATURE`

This enumeration is used internally by the amplifier object to check for certain features that are not present in every amp model or firmware version number.

Enumerator

FEATURE_EXTENDED_OUTPUT_PIN_CONFIG	Can the amplifier firmware accept more then 6 words of data for it's output pin configuration parameter?
FEATURE_GEAR_RATIO	Does the amplifier support the gear ratio parameter?
FEATURE_RESOLVER_CYCLES	Does the amplifier support the resolver cycles parameter?
FEATURE_HALL_VEL_SHIFT	Does the amplifier support the hall velocity shift parameter?
FEATURE_PLOOP_SCALE	Does the amplifier support the position loop scaling factor?
FEATURE_STEPPER_CRNT	Does the amp support the stepper current parameters (hold current, run to hold time, etc)?
FEATURE_CURRENT_SLOPE	Does the amp support the current slope limits?
FEATURE_SOFTLIM_ACCEL	Does the amp support the software limit acceleration parameter?
FEATURE_PWMIN_FREQ	Does the amp support the pwm input frequency parameter?
FEATURE_VLOOP_CMD_FILT	Does the amp support the velocity loop command filter?
FEATURE_USTEP_OUTER_LOOP	Does the amp support ustep outer loop.
FEATURE_STEP_DETENT_GAIN	Does the amp support ustep Detent gain correction factor.
FEATURE_USTEP_CONFIG_STATUS	Does the amp support ustep Stepper config and status.
FEATURE_ALGO_PHASE_INIT_CUR	Does the amp support algorithmic phase init max current an time settings.
FEATURE_ALGO_PHASE_INIT_CONFIG	Does the amp support algorithmic phase init config.
FEATURE_CAMMING	Does the amp support camming.
FEATURE_POS_WRAP	Does the amp support positon wrap.
FEATURE_ENC_OPTIONS	Does the amp support encoder options.
FEATURE_GAIN_SCHED	Does the amp support gain scheduling.
FEATURE_PIN_MAP	Does the amp support input pin mapping.
FEATURE_CAN_OPTIONS	Does the amp support can option.
FEATURE_CAN_SETTINGS	Does the amp support can option.
FEATURE_AIN_FILT	Does the amp support analog reference input filter.
FEATURE_VLOOP_OUT_FILT	Does the amp support the velocity loop cmd filter.
FEATURE_ILOOP_CMD_FILT	Does the amp support the current loop cmd filter.
FEATURE_PWMIN_MIN_PULSE	Does the amp support configuring the min pwm pulse width.
FEATURE_PWMIN_MAX_PULSE	Does the amp support configuring the max pwm pulse width.
FEATURE_DA_CONV_CONFIG	Does the amp support DA configuration.
FEATURE_SERVO_CONFIG	Does the amp support advanced servo loop configurations.
FEATURE_MTR_OVERTEMP	Does the amp support analog mtr over temp.
FEATURE_NET_OPTIONS	Does the amp support configuring the network.
FEATURE_PLOOP_KI	Does the amp support the position loop Ki param.
FEATURE_PLOOP_KD	Does the amp support the position loop Kd param.
FEATURE_VLOOP_CMDFF	Does the amp support the velocity loop command feed forward.
FEATURE_BRAKE_ENABLE_DELAY	Does the amp support motor brake enable delay time.
FEATURE_INPUT_SHAPING	Does the amp support input shaping.
FEATURE_FLOAT_FILT_COEF	Are filter coefficients stored at floating point values?
FEATURE_IO_OPTIONS	Does the amp support IO configuration.
FEATURE_PWMIN_UVCFG	Does the amp support the UV config in UV mode?

Enumerator

FEATURE_AXIS_CT	Does the amp support the paramter containing the number of axis?

6.20.1.4 AMP_MODE

`enum AMP_MODE`

This enumeration is used to specify the mode of operation of the amplifier.

The amplifier mode of operation specifies the control method to be used by the amplifier, as well as the source of input to that control structure.

The amplifier can be controlled in servo mode or in microstepping mode. When running in servo mode the amplifier uses up to three nested control loops. These loops control current, velocity and position. In microstepping mode the low level current loop is retained, but the upper level loops are replaced with a simple position command.

The command source of the amplifier will normally be the CANopen network itself. However, the amplifier also supports several low level control methods in which commands are received through analog or digital input pins, or even from an internal function generator.

Normally, only the CANopen modes of operation will be used when running over the CANopen network. These modes are AMPMODE_CAN_PROFILE, AMPMODE_CAN_VELOCITY, AMPMODE_CAN_HOMING, and AMPMODE_CAN↩_PVT. Each of these modes can be used on either servo or microstepping drives. It's typically not necessary to specify the type of control method (servo or microstepping) to be used with these modes as it can be determined by the type of amplifier being used. Servo amplifier's (such as Accelnet) default to servo mode, and microstepping amplifiers (such as Stepnet) will default to microstepping mode. If this default is not appropriate for the application, then the control method may be forced by ORing in one of the following two values; AMPMODE_CAN_SERVO and AMPMODE_CAN_USTEP.

Enumerator

AMPMODE_CAN_PROFILE	In this mode the CANopen network sends move commands to the amplifier, and the amplifier uses it's internal trajectory generator to perform the moves. This mode conforms to the CANopen device profile for motion control (DSP-402) profile position mode.
AMPMODE_CAN_VELOCITY	In this mode the CANopen network commands target velocity values to the amplifier. The amplifier uses it's programmed acceleration and deceleration values to ramp the velocity up/down to the target. Note that support for profile velocity mode was added in amplifier firmware version 3.06. Earlier versions of firmware will report an error if this mode is selected.
AMPMODE_CAN_TORQUE	In this mode the CANopen network commands torque values to the amplifier. Note that support for profile torque mode was added in amplifier firmware version 3.34. Earlier versions of firmware will report an error if this mode is selected.
AMPMODE_CAN_HOMING	This mode is used to home the motor (i.e. find the motor zero position) under the control of the CANopen network. This mode conforms to the CANopen device profile for motion control (DSP-402) homing mode.

Enumerator

AMPMODE_CAN_PVT	In this mode the CANopen master calculates the motor trajectory and streams it over the CANopen network as a set of points that the amplifier interpolated between. This mode conforms to the CANopen device profile for motion control (DSP-402) interpolated position mode.
AMPMODE_CAN_SERVO	This value may be combined with one of the standard CAN control modes to specify that servo control should be used. This is most often specified when a microstepping amplifier (such as the Stepnet) is to be used in servo mode.
AMPMODE_CAN_USTEP	This value may be combined with one of the standard CAN control modes to specify that microstepping control should be used. This is most often specified when a servo amplifier (such as the Accelnet) is to be used in microstepping mode.
AMPMODE_DISABLED	Disable the amplifier. In this mode, none of the controls loops are running, and no voltage will be applied across the motor windings.
AMPMODE_PROG_CRNT	Current mode in which the command to the current loop is simply a static value that may be programmed over the serial port or CANopen network. The programmed current value can be set with the function Amp::SetCurrentProgrammed
AMPMODE_AIN_CRNT	Current mode in which the command to the current loop is derived from the analog input. Note that some amplifier models do not support an analog input. Please refer to the amplifier datasheet to determine if this mode is applicable.
AMPMODE_DIN_CRNT	Current mode in which the command to the current loop is derived from the digital input pins. One or two of the digital inputs are used as a PWM input command which is interpreted as a current command. Please refer to the amplifier data sheet to determine which input(s) should be used in this mode.
AMPMODE_FGEN_CRNT	Current mode in which the command to the current loop is derived from the internal function generator.
AMPMODE_PROG_VEL	Velocity mode in which the command to the velocity loop is simply a static value that may be programmed over the serial or CANopen network. The programmed velocity value can be set with the function Amp::SetVelocityProgrammed
AMPMODE_AIN_VEL	Velocity mode in which the command to the velocity loop is derived from the analog input. Note that some amplifier models do not support an analog input. Please refer to the amplifier datasheet to determine if this mode is applicable.
AMPMODE_DIN_VEL	Velocity mode in which the command to the velocity loop is derived from the digital input pins. One or two of the digital inputs are used as a PWM input command which is interpreted as a velocity command. Please refer to the amplifier data sheet to determine which input(s) should be used in this mode.
AMPMODE_FGEN_VEL	Velocity mode in which the command to the velocity loop is derived from the internal function generator.
AMPMODE_DIN_POS	Position mode in which the command to the position loop is derived from the digital input pins. Two of the digital inputs can be configured as either a master encoder input (quadrature input), a step & direction input, or a step up / step down input. Please refer to the amplifier data sheet to determine which inputs should be used in this mode.
AMPMODE_FGEN_POS	Position mode in which the command to the position loop is derived from the internal function generator.
AMPMODE_CAM_POS	Position mode in which the command to the position loop is derived from CAM tables located in the amplifiers memory.

Enumerator

AMPMODE_DIN_USTEP	Microstepping mode in which the commanded position is derived from the digital input pins. Two of the digital inputs can be configured as either a master encoder input (quadrature input), a step & direction input, or a step up / step down input. Please refer to the amplifier data sheet to determine which inputs should be used in this mode.
AMPMODE_FGEN_USTEP	Microstepping mode in which the commanded position is derived from the internal function generator.
AMPMODE_DIAG_USTEP	Diagnostic microstepping mode. This is a very simple microstepping mode that can be used for motor setup and testing. A constant motor current is set by the programmed current value, and the motor phase is microstepped at a fixed rate. The position and velocity loops are not used in this mode.

6.20.1.5 AMP_PHASE_MODE

`enum AMP_PHASE_MODE`

Amplifier phasing mode.

This enumeration gives the legal values for the amplifier phasing mode setting. The phasing mode controls what type of input the amplifier uses to determing the phase angle when commutating a brushless DC motor.

Enumerator

PHASE_MODE_ENCODER	Use a combination of hall sensors and encoder input. The hall sensors are used at startup, and will be used to constantly adjust the phase angle on every hall transition. This is the default phase mode, and should be used when both hall sensors and encoder input are present.
PHASE_MODE_TRAP	Phase using only the hall sensor inputs. This mode gives rougher operation then the encoder based mode, however it can be used when no encoder input is available.
PHASE_MODE_NOADJUST	Use both encoder & hall inputs, but only use the hall inputs on startup and ignore them after that. This mode should normally not be used unless there is a good reason to ignore the hall inputs after startup.
PHASE_MODE_AHALL90	Use analog hall inputs offset at 90 deg connected to the encoder sine / cosine inputs.
PHASE_MODE_BRUSHED	This phase mode is used to force brushed DC motor output. It should only be used when the amplifier is connected to a brushed DC motor. NOTE - this mode is obsolete. The motor type parameter (object 0x2383) should now be used to indicate a brush motor which will force the correct commutation mode.
PHASE_MODE_NOHALL	Phase using the encoder only. In this mode, the amplifier will use an algorithmic phase initialization on startup. This mode can be used when an encoder is present, but no halls are available.
PHASE_MODE_ENCPHASE	Use phase information obtained from the encoder. This mode is used with resolvers and most absolute encoders which are able to communicate phasing information along with the position.
PHASE_MODE_TRAPINTERP	Use interpolated trapezoidal commutation. This mode can be used when digital hall sensors are available on the motor. At high speeds the amp interpolates between the hall sensor angles and creates a smoother commutation angle.

6.20.1.6 AMP_PWM_MODE

enum AMP_PWM_MODE

Amplifier PWM output mode.

This enumeration gives the legal values for setting up the amplifier's PWM output mode. The PWM output mode controls some details of how the amplifier drives it's PWM outputs.

Enumerator

PWM_MODE_STANDARD	Standard PWM mode. This mode should be selected for most applications.
PWM_MODE_FORCECLAMP	This bit forces the amplifier into PWM bus clamping mode. Bus clamping mode is a different method of driving the PWM outputs. It can produce less switching loss at the expense of greater cross over distortion.
PWM_MODE_AUTOCLAMP	Automatically switch between bus clamping and normal output mode based on the PWM duty cycle. Bus clamping mode is used at high duty cycles, normal mode is used at low duty cycles.
PWM_MODE_HEXLIMIT	If this bit is set, the amplifier's output voltage is limited using a method known as hexagonal limiting. If this bit is clear, circular limiting is used. Hexagonal limiting gives the maximum voltage output at the expense of some added torque ripple. Higher top speeds may be attained using hexagonal limiting.

6.20.1.7 AMP_TRACE_STATUS

enum AMP_TRACE_STATUS

Amplifier trace status bits.

The amplifier's trace mechanism reports it's status as a collection of these bits.

6.20.1.8 AMP_TRACE_TRIGGER

enum AMP_TRACE_TRIGGER

Amplifier trace trigger settings.

Enumerator

TRACETRIG_CHANNEL	These bits define which of the trace channels to use for triggering. Not all trigger types require a trace channel, for those this value is ignored.
TRACETRIG_TYPE	These bits define the trace trigger type to use.

Enumerator

TRACETRIG_NONE	Trace trigger type none. The trace is triggered immediately on start.
TRACETRIG_ABOVE	Trigger as soon as the value on the selected variable is above the trigger level.
TRACETRIG_BELOW	Trigger as soon as the value on the selected variable is below the trigger level.
TRACETRIG_RISE	Trigger when the value on the selected variable changes from below the trigger level to above it.
TRACETRIG_FALL	Trigger when the value on the selected variable changes from above the trigger level to below it.
TRACETRIG_BITSET	Treat the trigger level as a bit mask which selects one or more bits on the selected trace variable. The trigger occurs as soon as any of the selected bits are set.
TRACETRIG_BITCLR	Treat the trigger level as a bit mask which selects one or more bits on the selected trace variable. The trigger occurs as soon as any of the selected bits are clear.
TRACETRIG_CHANGE	Trigger any time the selected trace variable value changes.
TRACETRIG_EVENTSET	Treat the trigger level as a bit mask which selects one or more bits on the amplifier's event status register. The trigger occurs as any of the selected bits are set. Note that this trigger type does not use a trace variable.
TRACETRIG_EVENTCLR	Treat the trigger level as a bit mask which selects one or more bits on the amplifier's event status register. The trigger occurs as any of the selected bits are clear. Note that this trigger type does not use a trace variable.
TRACETRIG_FGEN_CYCLE	Trigger at the start of the next function generator cycle. This trigger type is only useful when running in function generator mode. It does not use a trace variable or the trigger level.
TRACETRIG_CAP_SET	Treat the trigger level as a bit mask which selects one or more bits on the amplifier's capture status register. The trigger occurs as any of the selected bits are set. Note that this trigger type does not use a trace variable.
TRACETRIG_BITCHANGE	Treat the trigger level as a bit mask which selects one or more bits on the selected trace variable. The trigger occurs as soon as any of the selected bits change state.
TRACETRIG_NODELAY	If this bit is set, then the trigger is allowed to occur even if the trace setup delay has not yet occurred. Normally, if a negative trace delay is set then that much time must expire after the trace has been started before a trigger will be recognized. If this bit is set, the trigger will be recognized even if the setup delay hasn't been met.
TRACETRIG_SAMPLE	Only take a single sample for each trigger. Normally, the occurance of the trigger causes the trace to begin sampling data and stop when the trace buffer is full. If this bit is set, each trigger occurance will cause a single sample of trace data.

6.20.1.9 AMP_TRACE_VAR

```
enum AMP_TRACE_VAR
```

Amplifier trace variables.

This enumeration lists the amplifier variables that are available for use with the amplifier's internal trace routine.

Enumerator

TRACEVAR_HIGH_VOLT	High voltage bus.

Enumerator

TRACEVAR_TEMP	Amplifier temperature.
TRACEVAR_ANALOG_REF	Analog reference input.
TRACEVAR_ENC_SIN	Analog encoder sine.
TRACEVAR_ENC_COS	Analog encoder cosine.
TRACEVAR_PHASE	Motor phase angle.
TRACEVAR_HALLS	Hall sensor state.
TRACEVAR_INPUTS	digital input pins (after deadtime)
TRACEVAR_RAW_INPUTS	digital input pins (before deadtime)
TRACEVAR_EVENTS	Event status register.
TRACEVAR_EVENTLATCH	Latched version of event status register.
TRACEVAR_CRNT_A	Actual current, current sensor A.
TRACEVAR_CRNT_B	Actual current, current sensor B.
TRACEVAR_CRNT_CMD	Commanded current (before limiting)
TRACEVAR_CRNT_LIM	Commanded current (after limiting)
TRACEVAR_CRNT_CMD_D	Commanded current, D axis.
TRACEVAR_CRNT_CMD_Q	Commanded current, Q axis.
TRACEVAR_CRNT_ACT_D	Actual current, calculated for D axis.
TRACEVAR_CRNT_ACT_Q	Actual current, calculated for Q axis.
TRACEVAR_CRNT_ERR_D	Current loop error, D axis.
TRACEVAR_CRNT_ERR_Q	Current loop error, Q axis.
TRACEVAR_VOLT_D	Current loop output voltage, D axis.
TRACEVAR_VOLT_Q	Current loop output voltage, Q axis.
TRACEVAR_VEL_MTR	Motor velocity with some filtering.
TRACEVAR_VEL_RAW	Motor velocity, unfiltered.
TRACEVAR_VEL_LOAD	Load encoder velocity.
TRACEVAR_VLOOP_CMD	Velocity loop commanded velocity (before limiting)
TRACEVAR_VLOOP_LIM	Velocity loop commanded velocity (after limiting)
TRACEVAR_VLOOP_ERR	Velocity loop error.
TRACEVAR_LOAD_POS	Load encoder position.
TRACEVAR_MTR_POS	Motor encoder position.
TRACEVAR_POS_ERR	Position error.
TRACEVAR_CMD_POS	Commanded position from trajectory generator.
TRACEVAR_CMD_VEL	Commanded velocity from trajectory generator.
TRACEVAR_CMD_ACC	Commanded acceleration from trajectory generator.
TRACEVAR_DEST_POS	Destination position.

6.20.1.10 COPLEY_HOME_METHOD

```
enum COPLEY_HOME_METHOD
```

Home methods supported by the Copley amplifier.

This enumeration gives more useful names to the various homing methods currently supported by the Copley amplifier.

The names of the members of this enumeration define the type of homing procedure. These names are made up of the following elements:

1. CHM: prefix that identifies the member as a Copley Home Method
2. Sensor: Defines the type of sensor that defines the location of the home position. It will be one of the following:
    - PLIM: A positive limit switch
    - NLIM: A negative limit switch
    - PHOME: A positive home switch. This is a home switch that goes active at some point, and remains active for all greater positions.
    - NHOME: A negative home switch. This is a home switch that goes active at some point, and remains active for all lower positions.
    - LHOME: The lower side of a momentary home switch. This type of home switch has an active region and is inactive on either side of that region. This choice selects the lower edge of that switch.
    - UHOME: The upper side of a momentary home switch. This type of home switch has an active region and is inactive on either side of that region. This choice selects the upper edge of that switch.
3. Index: This defines whether an encoder index pulse will be used to mark the exact home location in conjunction with the sensor. If so, it identifies which index position will be used. It will be one of the following:
    - none: If not specified, then no index is used. The edge of the sensor will define the home position.
    - ONDX: Outter index switch. This is an index on the inactive side of the sensor
    - INDX: Inner index switch. This is the first index on the active side of the sensor.
4. Initial move direction: For some home methods, this is provided and defines the initial move direction. The initial move direction is only specified if it isn't already obvious based on the home type.
    - none: The initial move direction is always obvious and does not need to be specified.
    - NEG: Move in the negative direction if the home position is not obvious. If the negative limit switch is encountered before the home region is found, then the move direction will be reversed.
    - POS: Move in the negative direction if the home position is not obvious. If the positive limit switch is encountered before the home region is found, then the move direction will be reversed.

Enumerator

CHM_NLIM_ONDX	Move into the negative limit switch, then back out to the first encoder index pulse beyond it. The index position is home.
CHM_PLIM_ONDX	Move into the positive limit switch, then back out to the first encoder index pulse beyond it. The index position is home.
CHM_PHOME_ONDX	Move to a positive home switch, then back out of it to the first encoder index outside the home region. The index position is home.
CHM_PHOME_INDX	Move to a positive home switch, and continue into it to the first encoder index inside the home region. The index position is home.
CHM_NHOME_ONDX	Move to a negative home switch, then back out of it to the first encoder index outside the home region. The index position is home.

Enumerator

CHM_NHOME_INDX	Move to a negative home switch, and continue into it to the first encoder index inside the home region. The index position is home.
CHM_LHOME_ONDX_POS	Move to the lower side of a momentary home switch. Then find the first encoder index pulse outside the home region. If the home switch is not active when the home sequence starts, then the initial move will be positive.
CHM_LHOME_INDX_POS	Move to the lower side of a momentary home switch. Then find the first encoder index pulse inside the home region. If the home switch is not active when the home sequence starts, then the initial move will be positive.
CHM_UHOME_INDX_POS	Move to the upper side of a momentary home switch. Then find the first encoder index pulse inside the home region. If the home switch is not active when the home sequence starts, then the initial move will be positive.
CHM_UHOME_ONDX_POS	Move to the upper side of a momentary home switch. Then find the first encoder index pulse outside the home region. If the home switch is not active when the home sequence starts, then the initial move will be positive.
CHM_UHOME_ONDX_NEG	Move to the upper side of a momentary home switch. Then find the first encoder index pulse outside the home region. If the home switch is not active when the home sequence starts, then the initial move will be negative.
CHM_UHOME_INDX_NEG	Move to the upper side of a momentary home switch. Then find the first encoder index pulse inside the home region. If the home switch is not active when the home sequence starts, then the initial move will be negative.
CHM_LHOME_INDX_NEG	Move to the lower side of a momentary home switch. Then find the first encoder index pulse inside the home region. If the home switch is not active when the home sequence starts, then the initial move will be negative.
CHM_LHOME_ONDX_NEG	Move to the lower side of a momentary home switch. Then find the first encoder index pulse outside the home region. If the home switch is not active when the home sequence starts, then the initial move will be negative.
CHM_NLIM	Move into the negative limit switch. The edge of the limit is home.
CHM_PLIM	Move into the positive limit switch. The edge of the limit is home.
CHM_PHOME	Move to a positive home switch. The edge of the home region is home.
CHM_NHOME	Move to a negative home switch. The edge of the home region is home.
CHM_LHOME_POS	Move to the lower side of a momentary home switch. The edge of the home region is home. If the home switch is not active when the home sequence starts, then the initial move will be positive.
CHM_UHOME_POS	Move to the upper side of a momentary home switch. The edge of the home region is home. If the home switch is not active when the home sequence starts, then the initial move will be positive.
CHM_UHOME_NEG	Move to the upper side of a momentary home switch. The edge of the home region is home. If the home switch is not active when the home sequence starts, then the initial move will be negative.
CHM_LHOME_NEG	Move to the lower side of a momentary home switch. The edge of the home region is home. If the home switch is not active when the home sequence starts, then the initial move will be negative.
CHM_NDX_NEG	Move in the negative direction until the first encoder index pulse is found. The index position is home.
CHM_NDX_POS	Move in the positive direction until the first encoder index pulse is found. The index position is home.
CHM_NONE	Set the current position to home.

Enumerator

CHM_HARDSTOP_POS	Home to a hard stop. The motor will start running in the positive direction until the homing current has been reached. It will hold this current until the homing delay has expired. The actual position after that delay is home.
CHM_HARDSTOP_NEG	Home to a hard stop. The motor will start running in the negative direction until the homing current has been reached. It will hold this current until the homing delay has expired. The actual position after that delay is home.
CHM_HARDSTOP_ONDX_POS	Home to a hard stop. The motor will start running in the positive direction until the homing current has been reached. It will hold this current until the homing delay has expired. It will them move away from the hard stop until an index mark is located. The index position is home.
CHM_HARDSTOP_ONDX_NEG	Home to a hard stop. The motor will start running in the negative direction until the homing current has been reached. It will hold this current until the homing delay has expired. It will them move away from the hard stop until an index mark is located. The index position is home.
CHM_EXTENDED	Home using an extended home method. This is not a real home method, but instead a flag that is used by the Amp::GoHome function to indicate that a non-standard homing method is to be used. When this value is passed it instructs the GoHome function to execute the home sequence through the use of a special low-level homing parameter implemented in the amplifier firmware. This allows a bit more flexibility on the home sequencer.

6.20.1.11 EVENT_STATUS

```
enum EVENT_STATUS
```

Amplifier event status word bit definitions.

Enumerator

ESTAT_SHORT_CRCT	Amplifier short circuit.
ESTAT_AMP_TEMP	Amplifier over temperature.
ESTAT_OVER_VOLT	Amplifier over voltage.
ESTAT_UNDER_VOLT	Amplifier under voltage.
ESTAT_MTR_TEMP	Motor over temperature.
ESTAT_ENCODER_PWR	Encoder power error.
ESTAT_PHASE_ERR	Phasing error.
ESTAT_CRNT_LIM	Current limited.
ESTAT_VOLT_LIM	Voltage limited.
ESTAT_POSLIM	Positive limit switch triggered.
ESTAT_NEGLIM	Negative limit switch triggered.
ESTAT_DISABLE_INPUT	Enable input pin not set.
ESTAT_SOFT_DISABLE	Disabled due to software request.
ESTAT_STOP	Try to stop motor (after disable, before brake)

Enumerator

ESTAT_BRAKE	Brake actuated.
ESTAT_PWM_DISABLE	PWM outputs disabled.
ESTAT_SOFTLIM_POS	Positive software limit reached.
ESTAT_SOFTLIM_NEG	Negative software limit reached.
ESTAT_TRK_ERR	Tracking error.
ESTAT_TRK_WARN	Tracking warning.
ESTAT_RESET	Amplifier has been reset.
ESTAT_POSWRAP	Encoder position wrapped (rotory) or hit limit (linear).
ESTAT_FAULT	Latching fault in effect.
ESTAT_VEL_LIMIT	Velocity is at limit.
ESTAT_ACC_LIMIT	Acceleration is at limit.
ESTAT_TRK_WIN	Not in tracking window if set.
ESTAT_HOME	Home switch is active.
ESTAT_MOVING	Trajectory generator active OR not yet settled.
ESTAT_VEL_WIN	Velocity error outside of velocity window when set.
ESTAT_PHASE_INIT	Set when using algorithmic phase init mode & phase not initialized.
ESTAT_CMD_INPUT	Command input fault is active.

6.20.1.12 HALT_MODE

```
enum HALT_MODE
```

The amplifier's halt mode defines it's action when a halt command is issued (Amp::HaltMove).

When the halt command is issued, the move in progress will be terminated using the method defined in this mode. Unless the HALT_DISABLE method is selected, the amplifier will remain enabled and holding position at the end of the halt sequence.

Enumerator

HALT_DISABLE	Disable the amplifier immediately.
HALT_DECEL	Slow down using the profile deceleration.
HALT_QUICKSTOP	Slow down using the quick stop deceleration.
HALT_ABRUPT	Slow down with unlimited deceleration.

6.20.1.13 INPUT_PIN_CONFIG

```
enum INPUT_PIN_CONFIG
```

Input pin configuration settings.

The digital input pins located on an amplifier can be programmed to perform some action. This enumeration provides a list of the possible settings for an input pin.

Note that it is perfectly legal to program more then one input pin to perform the same action. It's often useful to have two hardware disable inputs for example. If either of these inputs becomes active, the amplifier will be disabled.

Note: Bits 12-13 of the input config specify the axis the input configuration will be applied to.

Whether the inputs are configured to perform some action or not, it's still possible to read them directly using the Amp↩ ::GetInputs function.

Enumerator

INCFG_NONE	No function assigned to the input.
INCFG_RESET_R	Reset the amplifier on the Rising edge of the input.
INCFG_RESET_F	Reset the amplifier on the Falling edge of the input.
INCFG_POSLIM_H	Positive limit switch, active High.
INCFG_POSLIM_L	Positive limit switch, active Low.
INCFG_NEGLIM_H	Negative limit switch, active High.
INCFG_NEGLIM_L	Negative limit switch, active Low.
INCFG_MOTOR_TEMP_H	Motor temp sensor active high.
INCFG_MOTOR_TEMP_L	Motor temp sensor active low.
INCFG_CLR_FAULTS_H	Clear faults on edge, disable while high.
INCFG_CLR_FAULTS_L	Clear faults on edge, disable while low.
INCFG_RESET_DISABLE_R	Reset on rising edge, disable while high.
INCFG_RESET_DISABLE_F	Reset on falling edge, disable while low.
INCFG_HOME_H	Home switch, active high.
INCFG_HOME_L	Home switch, active low.
INCFG_DISABLE_H	Amplifier disable active high.
INCFG_DISABLE_L	Amplifier disable active low.
INCFG_PWM_SYNC_H	Sync input on falling edge, valid only on high speed inputs.
INCFG_MOTION_ABORT_H	Abort motion active high.
INCFG_MOTION_ABORT_L	Abort motion active low.
INCFG_SCALE_ADC_H	Scale analog reference input by a factor of 8 when high.
INCFG_SCALE_ADC_L	Scale analog reference input by a factor of 8 when low.
INCFG_HIGHSPEED_CAPTURE_R	High speed position capture on rising edge.
INCFG_HIGHSPEED_CAPTURE_F	High speed position capture on falling edge.
INCFG_COUNT_EDGES_R	Count rising edges of input, store the results to an indexer register.
INCFG_COUNT_EDGES_F	Count falling edges of input, store the results to an indexer register.
INCFG_ENCODER_FAULT_H	Encoder fault input, active high.
INCFG_ENCODER_FAULT_L	Encoder fault input, active low.
INCFG_ABORT_WINDOW_R	Abort move on rising edge if not within N counts of destination position.
INCFG_ABORT_WINDOW_F	Abort move on falling edge if not within N counts of destination position.
INCFG_HV_LOSS_DISABLE_H	Mark HV loss on rising edge, disable while high.
INCFG_HV_LOSS_DISABLE_L	Mark HV loss on falling edge, disable while low.
INCFG_TRJ_UPDATE_R	Trajectory update on rising edge.

Enumerator

INCFG_TRJ_UPDATE_F	Trajectory update on falling edge.
INCFG_CLR_FAULTS_EVENTS_R	Clear faults and event latch on rising edge.
INCFG_CLR_FAULTS_EVENTS_F	Clear faults and event latch on falling edge.
INCFG_DIS_SIM_ENC_L_BURST_R	Disable simulated encoder output when low. Burst current position on encoder output on rising edge.
INCFG_DIS_SIM_ENC_H_BURST↩ _F	Disable simulated encoder output when high. Burst current position on encoder output on falling edge.

6.20.1.14 OUTPUT_PIN_CONFIG

```
enum OUTPUT_PIN_CONFIG
```

Output pin configuration settings.

The digital output pins located on the amplifier can be programmed to follow one or more bits in one of the amplifier's status words.

This enumeration is used to specify which status word a particular output pin will follow, and whether the output will be active high or active low.

Each output pin has a configuration value associated with it (which should be programmed using one of the values of this enumeration), and a 32-bit mask value. If the output pin is configured to follow a status register, the mask identifies which bit(s) of the status register should be used to control the output pin. If any of the masked bits in the status register are set, then the output pin will go active.

Enumerator

OUTCFG_EVENT_STATUS_L	The output pin follows the amplifier's event status register and is active Low.
OUTCFG_EVENT_STATUS_H	The output pin follows the amplifier's event status register and is active High.
OUTCFG_EVENT_LATCH_L	The output pin follows the latched version of the amplifier's event status register and is active Low.
OUTCFG_EVENT_LATCH_H	The output pin follows the latched version of the amplifier's event status register and is active High.
OUTCFG_MANUAL_L	The output pin is manually controlled using the Amp::SetOutputs function, and the output is active Low.
OUTCFG_MANUAL_H	The output pin is manually controlled using the Amp::SetOutputs function, and the output is active High.
OUTCFG_TRJ_STATUS	The output pin follows bits in the trajectory status register.
OUTCFG_POSITION_WINDOW	The output pin will go active when the actual motor position is greater then the first output parameter, and less then the second output parameter.

Enumerator

OUTCFG_POSITION_TRIG_LOW2HIGH	The output pin will go active when the motor actual position crosses through a programmed value in the low to high direction. The pin will stay active for a programmed amount of time. The first output parameter specifies the position, and the second output parameter specifies the time to remain active in milliseconds.
OUTCFG_POSITION_TRIG_HIGH2LOW	The output pin will go active when the motor actual position crosses through a programmed value in the high to low direction. The pin will stay active for a programmed amount of time. The first output parameter specifies the position, and the second output parameter specifies the time to remain active in milliseconds.
OUTCFG_POSITION_TRIG	The output pin will go active when the motor actual position crosses through a programmed value in either direction. The pin will stay active for a programmed amount of time. The first output parameter specifies the position, and the second output parameter specifies the time to remain active in milliseconds.
OUTCFG_POSITION_TRIG_LIST	The output pin will go active when the motor actual position crosses through any one of a series of programmed values in either direction. The pin will stay active for a programmed amount of time. The list of output positions to trigger on must be uploaded into an area of trace memory. The first output parameter gives a 16-bit word offset into trace memory in it's upper half, and the number of positions in it's lower half. The second parameter specifies the time to remain active in milliseconds.
OUTCFG_SYNC_OUTPUT	If set the output pin is used as Sync output. This bit can only be used with output pin 0.
OUTCFG_ACTIVE_HIGH	This bit may be ORed with any of the other output pin configuration values to make them active high.

6.20.1.15 POS_CAPTURE_CFG

```
enum POS_CAPTURE_CFG
```

Position capture configuration.

The amplifier is able to capture the encoder position on one of two events; either the encoder index signal, or a general purpose input pin which has been configured as a home switch.

This enumeration gives the values that my be used to configure this capture mechanism using the Amp::SetPos↩ CaptureCfg method.

Enumerator

CAPTURE_INDEX_RISING	If this bit is set, the rising edge of the encoder index signal will be used to capture the index position.
CAPTURE_INDEX_FALLING	If this bit is set, the falling edge of the encoder index signal will be used to capture the index position.

Enumerator

CAPTURE_INDEX_LATCH	If this bit is set, then index capture values will not be overwritten if a new index edge is received before the previously captured value has been read.
CAPTURE_HOME_LATCH	If this bit is set, then captured home sensor positions will not be overwritten if a new home input edge is received before the previous captured value was read.
CAPTURE_HIGH_SPEED_INPUT	If this bit is set, then the high speed input based position capture is enabled. Note that this features requires firmware versions >= 5.12 to work.
CAPTURE_HIGH_SPEED_INPUT_LATCH	If this bit is set, then captured high speed input positions will not be overwritten if a new input is received before the previous position was read.

6.20.1.16 POS_CAPTURE_STAT

```
enum POS_CAPTURE_STAT
```

Position capture status register value.

The current status of the position capture mechanism may be read from the amplifier using the method Amp::GetPos↩CaptureCfg.

This status value is bitmapped as described by this enumeration. Any bits not described here should be ignored. Bits not described here are reserved and may be either 1 or 0.

Enumerator

CAPTURE_INDEX_FULL	If this bit is set it indicates that a new encoder index position has been captured. This position may be read using the method Amp::GetIndexCapture. Reading the captured position will cause this bit to be cleared.
CAPTURE_INDEX_OVER	If this bit is set it indicates that a new encoder index was received before the previous captured index position was read from the amplifier. The setting of the CAPTURE_INDEX_LATCH bit in the capture control register determines whether or not the new captured position was stored. If the CAPTURE_INDEX_LATCH configuration bit is set, then the new captured position will be lost. If this bit is clear then the newly captured position will overwrite the previous position. Reading the captured position will cause this bit to be cleared.
CAPTURE_HOME_FULL	If this bit is set it indicates that a new home sensor position has been captured. This position may be read using the method Amp::GetHomeCapture. Reading the captured position will cause this bit to be cleared.
CAPTURE_HOME_OVER	If this bit is set it indicates that a new home sensor transition was received before the previous captured home position was read from the amplifier. The setting of the CAPTURE_HOME_LATCH bit in the capture control register determines whether or not the new captured position was stored. If the CAPTURE_HOME_LATCH configuration bit is set, then the new captured position will be lost. If this bit is clear then the newly captured position will overwrite the previous position. Reading the captured position will cause this bit to be cleared.

6.20.1.17 PROFILE_TYPE

enum PROFILE_TYPE

Point to point profile types.

This enumeration gives the various profile types supported by the Copley amplifiers. These profile types are used when running in profile position mode (point to point moves).

Enumerator

PROFILE_VEL	Velocity profile. In this profile mode the velocity, acceleration and deceleration values are used. The position value is also used, but it only defines the direction of motion (positive is position is >= 0, negative if position is < 0).
PROFILE_TRAP	Trapezoidal profile. In this profile mode a position, velocity, acceleration and deceleration may be specified. This profile mode allows any of it's parameters (position, vel, accel, decel) to be changed during the course of a move.
PROFILE_SCURVE	Jerk limited (S-curve) profile. In this mode, position, velocity, acceleration, and jerk (rate of change of acceleration) may be specified.

6.20.1.18 QUICK_STOP_MODE

enum QUICK_STOP_MODE

The amplifier's quick stop mode defines it's action when a quick stop command is issued (Amp::QuickStop).

The quick stop command differs from the halt command in that the amplifier is always disabled at the end of the sequence. For some modes, the amplifier automatically disables after halting the move. For others, the amplifier halts the move and holds in the quick stop state. No new moves may be started until the amplifier has manually been disabled.

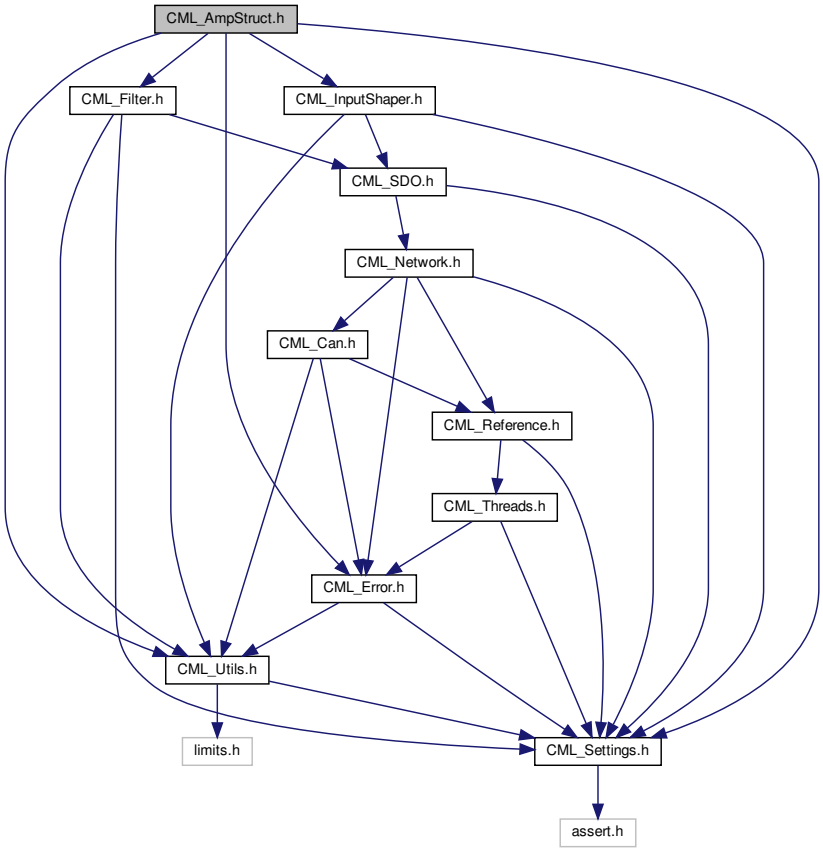
Enumerator

QSTOP_DISABLE	Disable the amplifier immediately.
QSTOP_DECEL	Slow down using the profile deceleration then disable.
QSTOP_QUICKSTOP	Slow down using the quick stop deceleration then disable.
QSTOP_ABRUPT	Slow down with unlimited deceleration then disable.
QSTOP_DECEL_HOLD	Slow down and hold.
QSTOP_QUICKSTOP_HOLD	Quick stop and hold.
QSTOP_ABRUPT_HOLD	Abrupt stop and hold.

6.21 CML_AmpStruct.h File Reference

This file contains a number of structures used to pass configuration parameters to an Amp object.

Include dependency graph for CML_AmpStruct.h:



This graph shows which files directly or indirectly include this file:

Classes

- class AmpFileError

  This class represents error conditions that can occur when loading amplifer data from a data file.
- struct AmpInfo

Amplifier characteristics data structure.

- struct PosLoopConfig

  This structure holds the position loop configuration parameters specific to the Copley amplifier.
- struct ServoLoopConfig

  This structure holds configuration info about specific parts of the velocity and position loops.
- struct EncoderErrorConfig

  This structure holds configuration info about the encoder error filter.
- struct VelLoopConfig

  This structure holds the velocity loop configuration parameters specific to the Copley amplifier.
- struct CrntLoopConfig

  This structure holds the current loop configuration parameters.
- struct HomeConfig

  Homing parameter structure.
- struct ProfileConfig

  Amplifier profile parameters.
- struct TrackingWindows

  Position and velocity error windows.
- struct MtrInfo

  Motor information structure.
- struct AmpIoCfg

  Programmable I/O pin configuration.
- struct SoftPosLimit

  Software limit switch configuration.
- struct RegenConfig

  Configuration structure used to set up the amplifier regeneration resister.
- struct FuncGenConfig

  Configuration parameters for amplifier's internal function generator.
- struct AnalogRefConfig

  Analog input configuration.
- struct PwmInConfig

  PWM or Pulse/Direction input configuration.
- struct CanNetworkConfig

  CANopen Node ID and bit rate configuration.
- struct NetworkOptions

  Configuration structure used to configure the amplifiers network support.
- struct DAConfig

  Configuration structure used to hold the settings for a drive's D/A converter.
- struct UstepConfig

  Configuration structure used to set up the microstepper.
- struct AlgoPhaseInit

  Configuration structure used to set up algorithmic phase init.
- struct CammingConfig

  Configuration structure used to set up the camming.
- struct GainScheduling

  Configuration structure used to set up the Gain Scheduling.
- struct AmpConfig

  Amplifier configuration structure.

Macros

- #define COPLEY_MAX_INPUTS 26

  Maximum available on any amplifier.

Enumerations

- enum CAN_BIT_RATE {
  CAN_RATE_1MEG = 0x0000,
  CAN_RATE_800K = 0x1000,
  CAN_RATE_500K = 0x2000,
  CAN_RATE_250K = 0x3000,
  CAN_RATE_125K = 0x4000,
  CAN_RATE_50K = 0x5000,
  CAN_RATE_20K = 0x6000,
  CAN_RATE_100K = 0x8000 }

  CANopen network bit rate enumeration.

6.21.1 Detailed Description

This file contains a number of structures used to pass configuration parameters to an Amp object.

6.21.2 Enumeration Type Documentation

6.21.2.1 CAN_BIT_RATE

`enum CAN_BIT_RATE`

CANopen network bit rate enumeration.

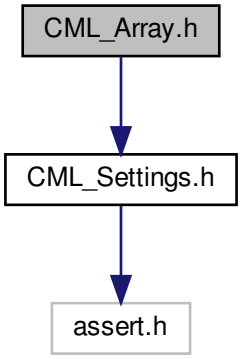
Enumerator

CAN_RATE_1MEG	1,000,000 bits / second
CAN_RATE_800K	800,000 bits / second
CAN_RATE_500K	500,000 bits / second
CAN_RATE_250K	250,000 bits / second
CAN_RATE_125K	125,000 bits / second
CAN_RATE_50K	50,000 bits / second
CAN_RATE_20K	20,000 bits / second
CAN_RATE_100K	100,000 bits / second

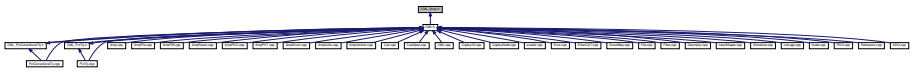
6.22 CML_Array.h File Reference

This file implements a simple dynamic array template used in CML.

Include dependency graph for CML_Array.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Array< C >

  This class template implements a simple dynamic array of a given type.

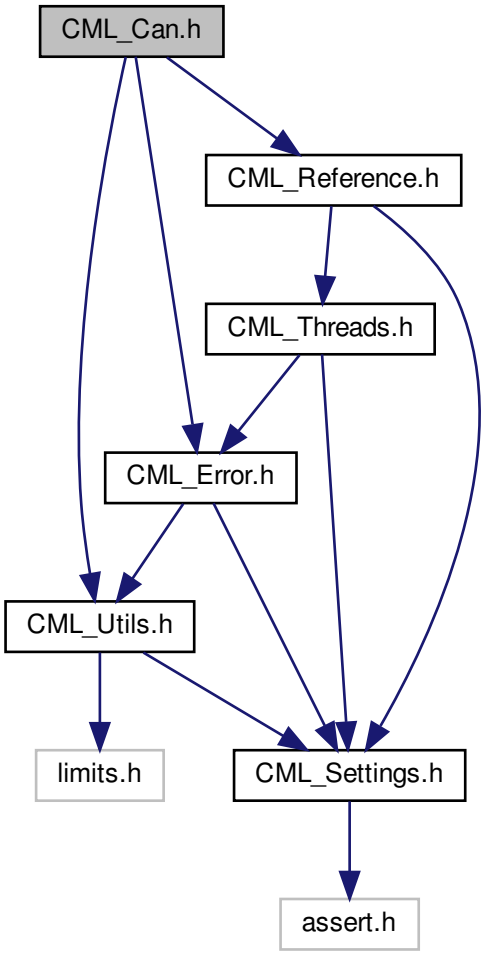
6.22.1 Detailed Description

This file implements a simple dynamic array template used in CML.

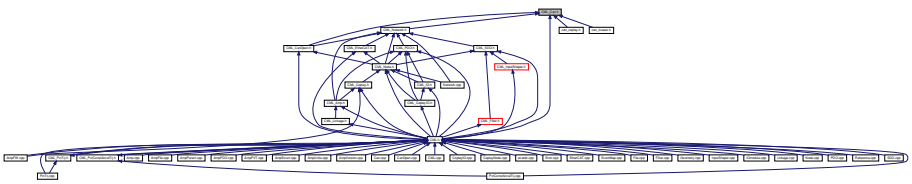
6.23 CML_Can.h File Reference

This file contains the base classes used to define the low level interface to the CAN network hardware.

Include dependency graph for CML_Can.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct CanFrame

  Low level CAN data frame.
- class CanError

  Class used to represent an error condition returned from a CAN interface function.
- class CanInterface

  Abstract class used for low level interaction with CAN hardware.

Enumerations

- enum CAN_FRAME_TYPE {
  CAN_FRAME_DATA,
  CAN_FRAME_REMOTE,
  CAN_FRAME_ERROR }

  This enumeration is used to identify the type of CAN frame.

6.23.1 Detailed Description

This file contains the base classes used to define the low level interface to the CAN network hardware.

6.23.2 Enumeration Type Documentation

6.23.2.1 CAN_FRAME_TYPE

`enum CAN_FRAME_TYPE`

This enumeration is used to identify the type of CAN frame.

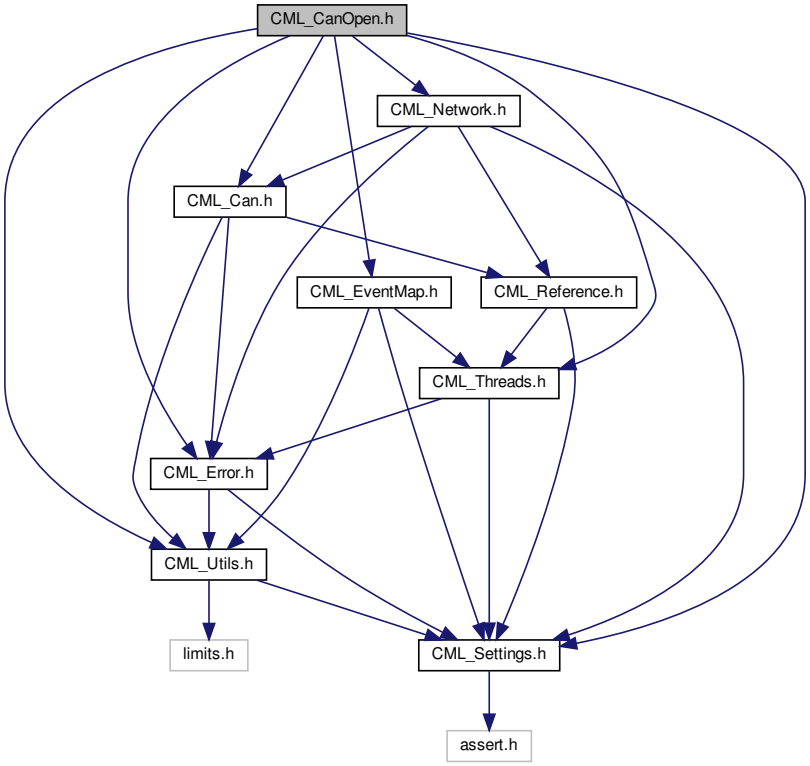
Enumerator

CAN_FRAME_DATA	Standard CAN data frame.
CAN_FRAME_REMOTE	Remote frame.
CAN_FRAME_ERROR	Error frame.

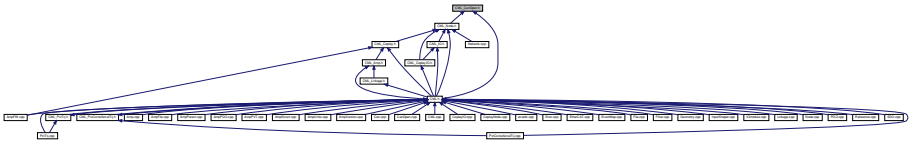
6.24 CML_CanOpen.h File Reference

This header file defines the classes used for the top level of the CANopen network.

Include dependency graph for CML_CanOpen.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CanOpenError

  This class holds the error codes that describe CANopen error conditions.
- class CanOpenSettings

  Configuration object used to customize global settings for the CANopen network.
- struct CanOpenNodeInfo

  The CanOpenNodeInfo structure holds some data required by the CANopen network interface which is present in every node it manages.
- class CanOpen

The CanOpen class is the top level interface into the CANopen network.

- class Receiver

  CANopen receiver object.

- class LSS

  CANopen Layer Setting Services object.

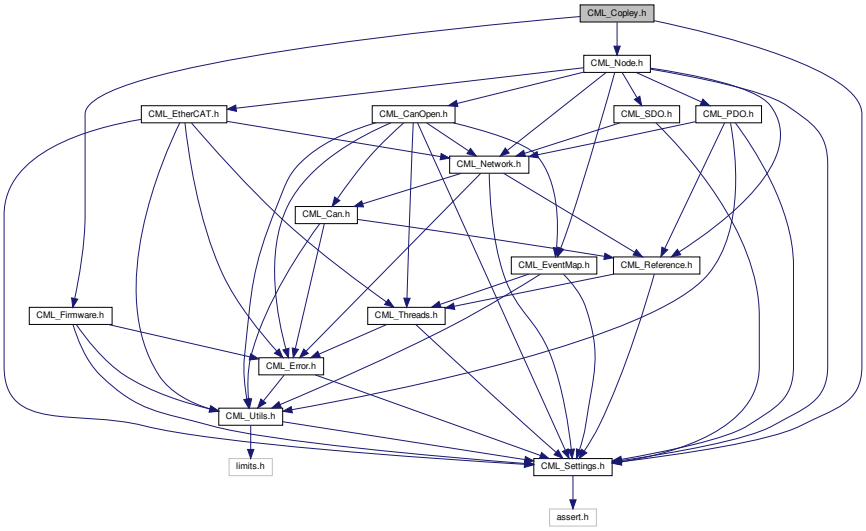
6.24.1 Detailed Description

This header file defines the classes used for the top level of the CANopen network.

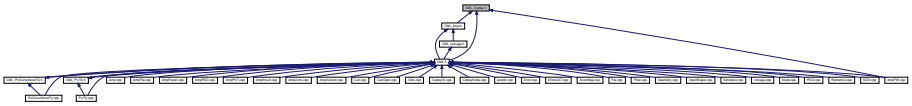
6.25 CML_Copley.h File Reference

This header file defines a generic Copley node type.

Include dependency graph for CML_Copley.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CopleyNodeError

  This class represents errors that can be returned by the CopleyNode class.
- class CopleyNode

  Copley CANopen Node class.

6.25.1 Detailed Description

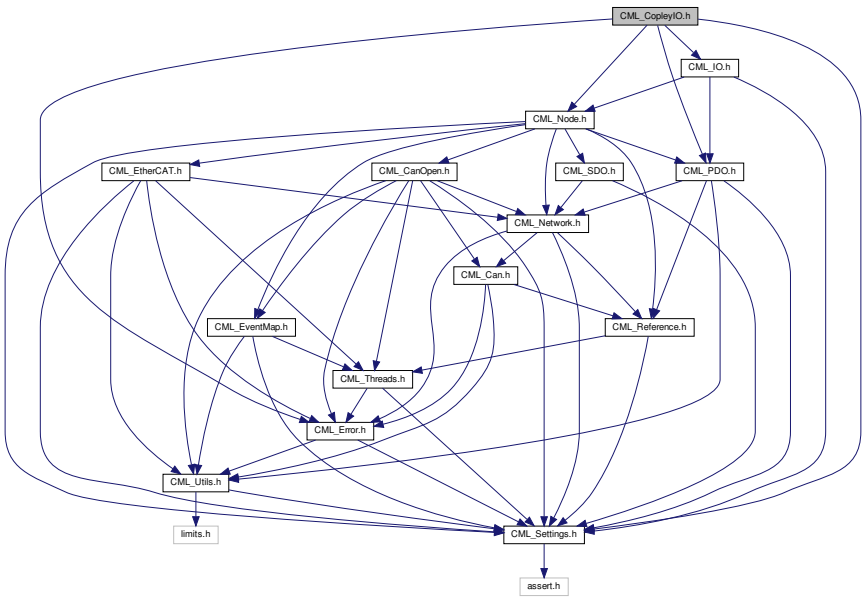
This header file defines a generic Copley node type.

This is the base class of all CANopen devices produced by Copley Controls Corp.

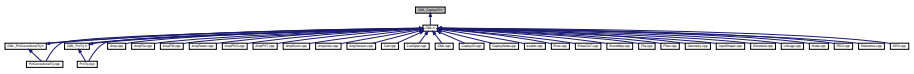
6.26 CML_CopleyIO.h File Reference

Standard CANopen I/O module support.

Include dependency graph for CML_CopleyIO.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct CopleyIOInfo

  IO Module characteristics data structure.
- struct CopleyIODigi

  This structure is used to return information about the digital I/O of a Copley I/O module.
- struct CopleyIOAnlg

  This structure is used to return information about the analog inputs of a Copley I/O module.
- struct CopleyIOPWM

  This structure is used to return information about the PWM outputs of a Copley I/O module.
- struct CopleyIOCfg

  IO Module configuration structure.
- class CopleyIO

  This class represents a Copley CANopen I/O module.
- class IOFileError

  This class represents error conditions that can occur when loading IO module data from a data file.

Enumerations

- enum CIO_OBJID {
  CIOOBJID_INFO_SERIAL = 0x3000,
  CIOOBJID_INFO_MODEL = 0x3001,
  CIOOBJID_INFO_MFGINFO = 0x3002,
  CIOOBJID_INFO_HWTYPE = 0x3003,
  CIOOBJID_INFO_LOOPRATE = 0x3004,
  CIOOBJID_INFO_FWVERSION = 0x3010,
  CIOOBJID_INFO_BAUD = 0x3011,
  CIOOBJID_INFO_MAXWORDS = 0x3012,
  CIOOBJID_INFO_NAME = 0x3013,
  CIOOBJID_INFO_HOSTCFG = 0x3014,
  CIOOBJID_INFO_NODECFG = 0x3015,
  CIOOBJID_INFO_RATECFG = 0x3016,
  CIOOBJID_INFO_NODEID = 0x3017,
  CIOOBJID_INFO_STATUS = 0x3018,
  CIOOBJID_INFO_RATE = 0x3019,
  CIOOBJID_INFO_ANLGINT = 0x301A,
  CIOOBJID_INFO_ANLGINTENA = 0x301B,
  CIOOBJID_INFO_DIGIINTENA = 0x301C,
  CIOOBJID_INFO_PWMPERIODA = 0x301E,
  CIOOBJID_INFO_PWMPERIODB = 0x301F,
  CIOOBJID_DIGI_BANKMODE = 0x3020,
  CIOOBJID_DIGI_PULLUPMSK = 0x3021,
  CIOOBJID_DIGI_TYPEMSK = 0x3022,
  CIOOBJID_DIGI_FAULTMSK = 0x3023,
  CIOOBJID_DIGI_INVMSK = 0x3024,
  CIOOBJID_DIGI_VALUEMSK = 0x3025,
  CIOOBJID_DIGI_MODEMSK = 0x3026,
  CIOOBJID_DIGI_RAWMSK = 0x3027,
  CIOOBJID_DIGI_HILOMSK = 0x3028,
  CIOOBJID_DIGI_LOHIMSK = 0x3029,
  CIOOBJID_DIGI_DEBOUNCE0 = 0x3030,

CIOOBJID_DIGI_DEBOUNCE1 = 0x3031,
CIOOBJID_DIGI_DEBOUNCE2 = 0x3032,
CIOOBJID_DIGI_DEBOUNCE3 = 0x3033,
CIOOBJID_DIGI_DEBOUNCE4 = 0x3034,
CIOOBJID_DIGI_DEBOUNCE5 = 0x3035,
CIOOBJID_DIGI_DEBOUNCE6 = 0x3036,
CIOOBJID_DIGI_DEBOUNCE7 = 0x3037,
CIOOBJID_ANLG_IRAW = 0x3040,
CIOOBJID_ANLG_ISCALED = 0x3041,
CIOOBJID_ANLG_IFACTOR = 0x3042,
CIOOBJID_ANLG_IOFFSET = 0x3043,
CIOOBJID_ANLG_IUPLIMIT = 0x3044,
CIOOBJID_ANLG_ILOLIMIT = 0x3045,
CIOOBJID_ANLG_IABSDELTA = 0x3046,
CIOOBJID_ANLG_IPOSDELTA = 0x3047,
CIOOBJID_ANLG_INEGDELTA = 0x3048,
CIOOBJID_ANLG_IFLAGS = 0x3049,
CIOOBJID_ANLG_IMASK = 0x304A,
CIOOBJID_PWM_ORAW = 0x3050,
CIOOBJID_PWM_OSCALED = 0x3051,
CIOOBJID_PWM_OFACTOR = 0x3052,
CIOOBJID_PWM_OOFFSET = 0x3053 }

Object dictionary ID values used on Copley I/O modules.

6.26.1 Detailed Description

Standard CANopen I/O module support.

6.26.2 Enumeration Type Documentation

6.26.2.1 CIO_OBJID

`enum CIO_OBJID`

Object dictionary ID values used on Copley I/O modules.

Enumerator

CIOOBJID_INFO_SERIAL	Serial number.
CIOOBJID_INFO_MODEL	Model number string.
CIOOBJID_INFO_MFGINFO	Amplifier's manufacturing information string.
CIOOBJID_INFO_HWTYPE	Hardware type code.
CIOOBJID_INFO_LOOPRATE	Main loop update rate (Hz)
CIOOBJID_INFO_FWVERSION	Firmware version number.
CIOOBJID_INFO_BAUD	Serial port baud rate (bps)

Enumerator

CIOOBJID_INFO_MAXWORDS	Maximum number of words sent with any command.
CIOOBJID_INFO_NAME	I/O module name.
CIOOBJID_INFO_HOSTCFG	Host configuration state (CME use only)
CIOOBJID_INFO_NODECFG	CAN node ID configuration.
CIOOBJID_INFO_RATECFG	CAN bit rate configuration.
CIOOBJID_INFO_NODEID	CAN node ID.
CIOOBJID_INFO_STATUS	CAN network status word.
CIOOBJID_INFO_RATE	CAN network bit rate.
CIOOBJID_INFO_ANLGINT	Active analog interrupts.
CIOOBJID_INFO_ANLGINTENA	Analog input global interrupt enable.
CIOOBJID_INFO_DIGIINTENA	Digital input global interrupt enable.
CIOOBJID_INFO_PWMPERIODA	PWM bank A period.
CIOOBJID_INFO_PWMPERIODB	PWM bank B period.
CIOOBJID_DIGI_BANKMODE	Digital I/O bank mode.
CIOOBJID_DIGI_PULLUPMSK	Digital I/O pull-up resistor mask.
CIOOBJID_DIGI_TYPEMSK	Digital I/O output type mask.
CIOOBJID_DIGI_FAULTMSK	Digital I/O output fault state mask.
CIOOBJID_DIGI_INVMSK	Digital I/O inversion mask.
CIOOBJID_DIGI_VALUEMSK	Digital I/O data value mask.
CIOOBJID_DIGI_MODEMSK	Digital I/O output fault mode mask.
CIOOBJID_DIGI_RAWMSK	Digital I/O raw data value mask.
CIOOBJID_DIGI_HILOMSK	Digital I/O input low->high interrupt mask.
CIOOBJID_DIGI_LOHIMSK	Digital I/O input high->low interrupt mask.
CIOOBJID_DIGI_DEBOUNCE0	Digital I/O debounce time, bit 0.
CIOOBJID_DIGI_DEBOUNCE1	Digital I/O debounce time, bit 1.
CIOOBJID_DIGI_DEBOUNCE2	Digital I/O debounce time, bit 2.
CIOOBJID_DIGI_DEBOUNCE3	Digital I/O debounce time, bit 3.
CIOOBJID_DIGI_DEBOUNCE4	Digital I/O debounce time, bit 4.
CIOOBJID_DIGI_DEBOUNCE5	Digital I/O debounce time, bit 5.
CIOOBJID_DIGI_DEBOUNCE6	Digital I/O debounce time, bit 6.
CIOOBJID_DIGI_DEBOUNCE7	Digital I/O debounce time, bit 7.
CIOOBJID_ANLG_IRAW	Analog input raw value.
CIOOBJID_ANLG_ISCALED	Analog input scaled value.
CIOOBJID_ANLG_IFACTOR	Analog input scaling factor.
CIOOBJID_ANLG_IOFFSET	Analog input offset.
CIOOBJID_ANLG_IUPLIMIT	Analog input upper limit for interrupt.
CIOOBJID_ANLG_ILOLIMIT	Analog input lower limit for interrupt.
CIOOBJID_ANLG_IABSDELTA	Analog input absolute delta value for interrrupt.
CIOOBJID_ANLG_IPOSDELTA	Analog input positive delta value for interrrupt.
CIOOBJID_ANLG_INEGDELTA	Analog input negative delta value for interrrupt.
CIOOBJID_ANLG_IFLAGS	Analog input interrrupt flags.
CIOOBJID_ANLG_IMASK	Analog input interrrupt mask.
CIOOBJID_PWM_ORAW	PWM output raw value.

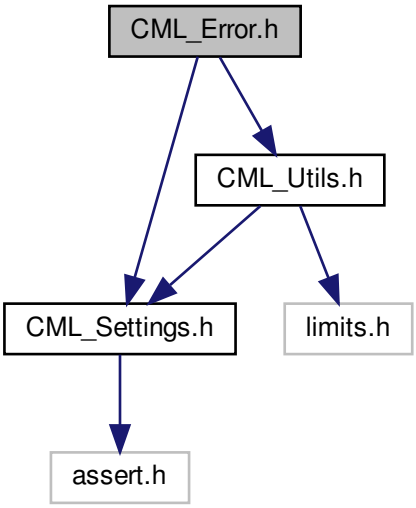
Enumerator

CIOOBJID_PWM_OSCALED	PWM output scaled value.
CIOOBJID_PWM_OFACTOR	PWM output scaling factor.
CIOOBJID_PWM_OOFFSET	PWM output offset.

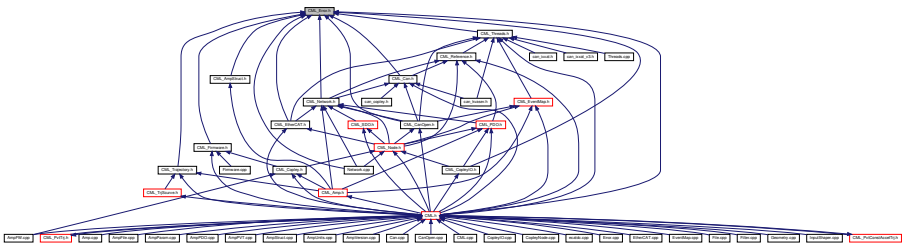
6.27 CML_Error.h File Reference

This file defines the top level error class used throughout the library.

Include dependency graph for CML_Error.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Error

  This class is the root class for all error codes returned by functions defined within the Motion Library.

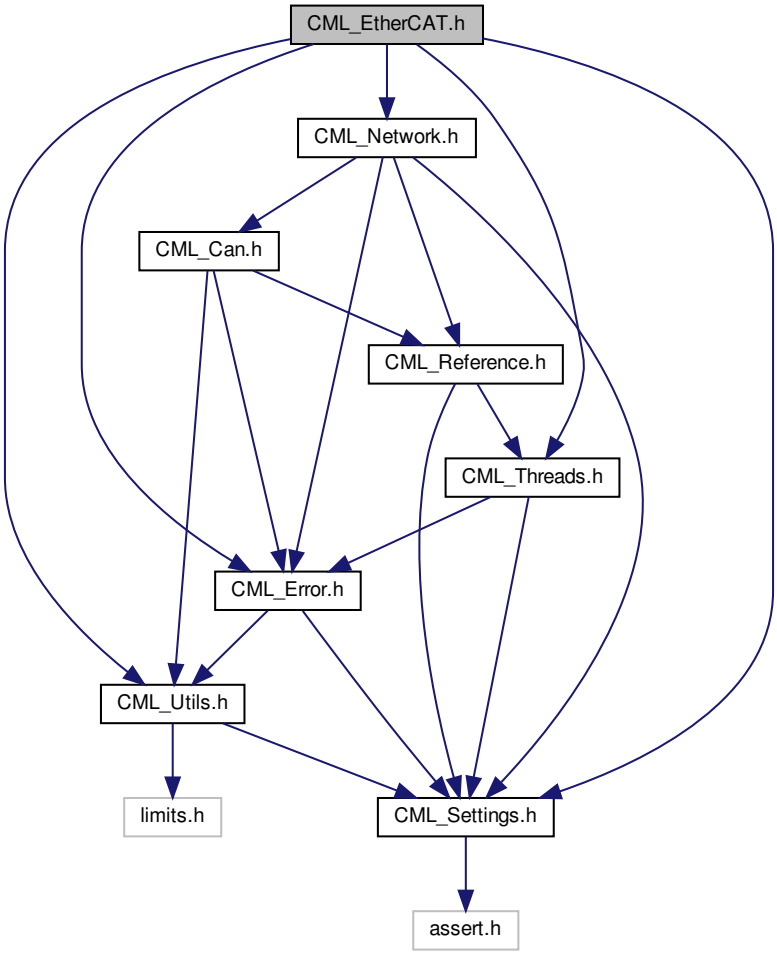
6.27.1 Detailed Description

This file defines the top level error class used throughout the library.

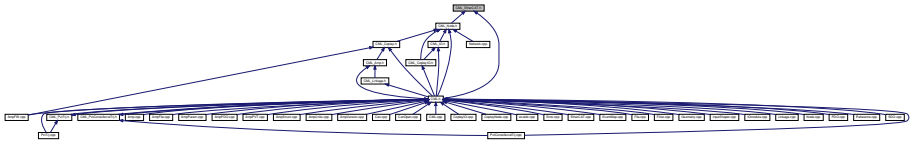
6.28 CML_EtherCAT.h File Reference

This header file defines the classes used to represent the top level of the EtherCAT network interface.

Include dependency graph for CML_EtherCAT.h:



This graph shows which files directly or indirectly include this file:



Classes

- class EtherCatError

  This class holds the error codes that describe EtherCAT error conditions.
- class EtherCatSettings

  Configuration object used to customize global settings for the EtherCAT network.
- class EtherCatHardware

  Low level Ethernet hardware interface.
- class EtherCAT

  The EtherCAT class is the top level interface into the EtherCAT network.
- class EcatDgram

  Generic EtherCAT datagram class.
- struct BRD

  Broadcast read.
- struct BWR

  Broadcast write. This type of datagram writes data to the same location on every node in the network.
- struct APRD

  Read by position in network (aka Auto Increment Physical Read) The read is performed on the node who's position matches the passed address.
- struct APWR

  Write by position in network (Auto Increment Physical Write) Like the APRD datagram, but a write version.
- struct ARMW

  Read by position in network and write to the same address of all following nodes.
- struct FPRD

  Read by assigned node ID (Configured Address Physical Read) The master assigns each node a unique 16-bit address at startup.
- struct FPWR

  Write by assigned node ID (Configured Address Physical Write)
- class EcatFrame

  EtherCAT frame class.

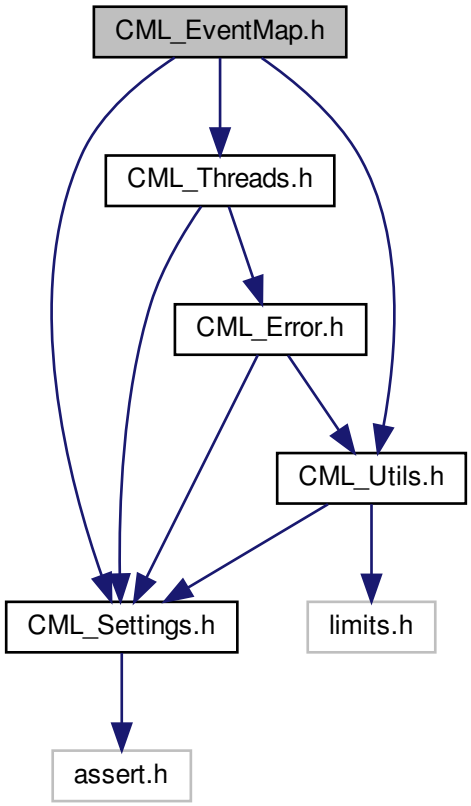
6.28.1 Detailed Description

This header file defines the classes used to represent the top level of the EtherCAT network interface.

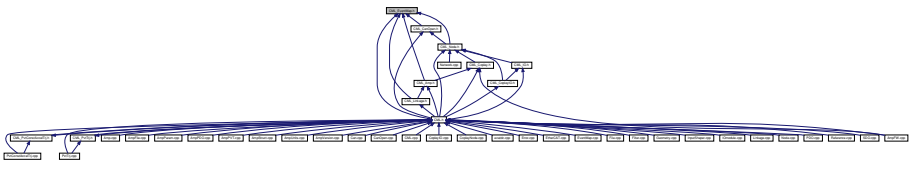
6.29 CML_EventMap.h File Reference

This file defines the Event Map class.

Include dependency graph for CML_EventMap.h:



This graph shows which files directly or indirectly include this file:



Classes

- class EventError

This class represents error conditions related to the Event object.

- class Event

  Events are a generic mechanism used to wait on some condition.

- class EventAny

  This is an event that matches if any of a group of bits are set in the EventMap mask.

- class EventAnyClear

  This is an event that matches if any of a group of bits are clear in the EventMap mask.

- class EventAll

  This is an event that matches if all of a group of bits are set in the EventMap mask.

- class EventNone

  This is an event that matches if none of a group of bits are set in the EventMap mask.

- class EventMap

  An event map is a mechanism that allows one or more threads to wait on some pre-defined event, or group of events.

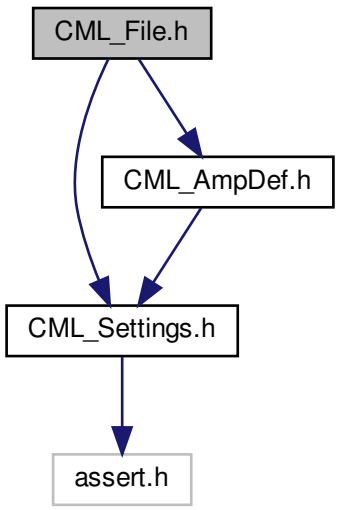
6.29.1 Detailed Description

This file defines the Event Map class.

6.30 CML_File.h File Reference

This file holds various handy functions for parsing files.

Include dependency graph for CML_File.h:



This graph shows which files directly or indirectly include this file:

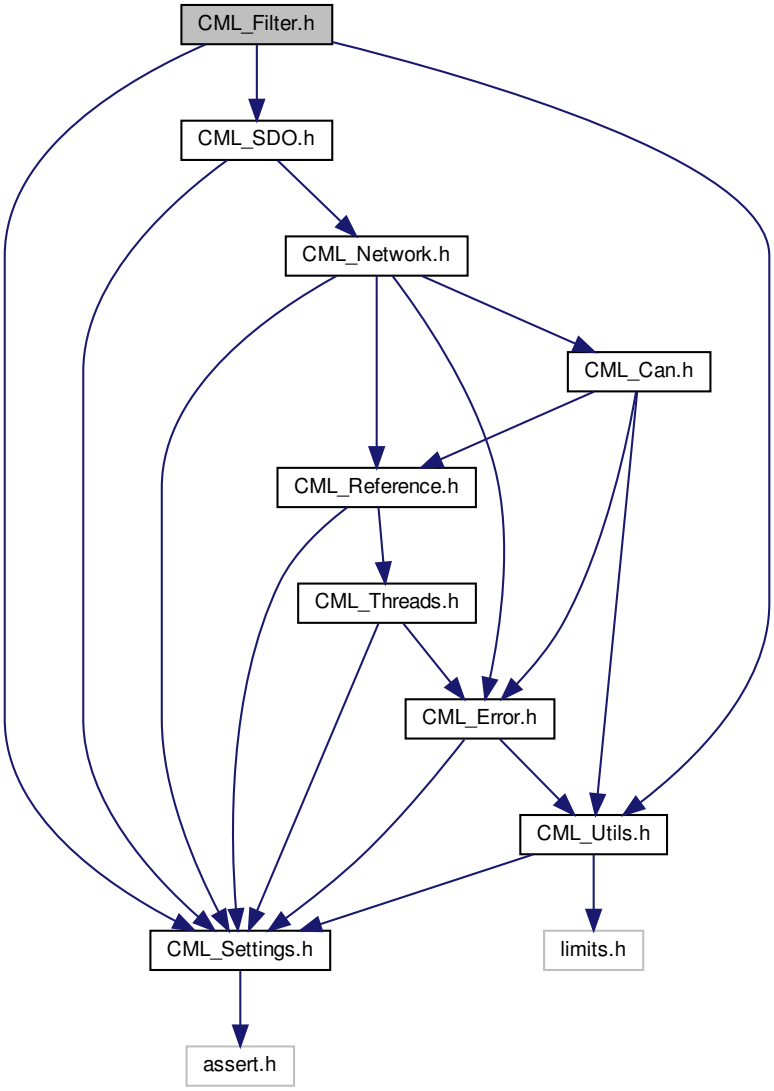
6.30.1 Detailed Description

This file holds various handy functions for parsing files.

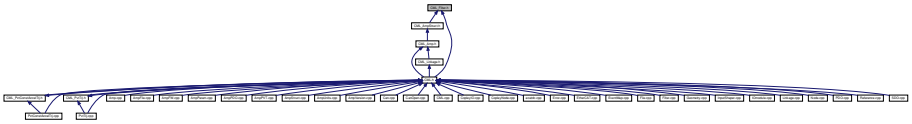
6.31 CML_Filter.h File Reference

This file defines the Filter object.

Include dependency graph for CML_Filter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Filter

  Generic filter structure.

6.31.1 Detailed Description

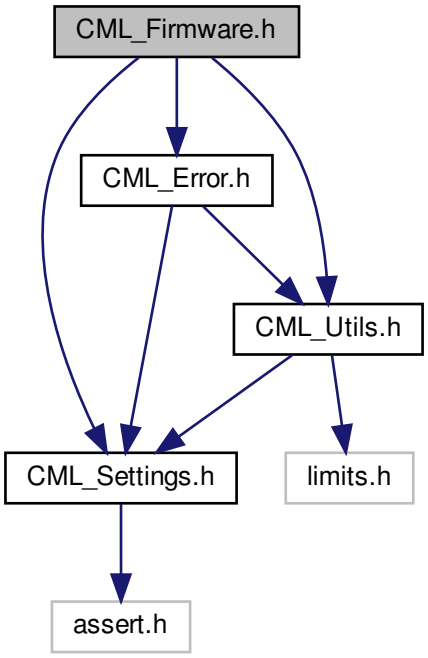
This file defines the Filter object.

The Filter object represents a two pole filter structure used in various locations within the amplifier.

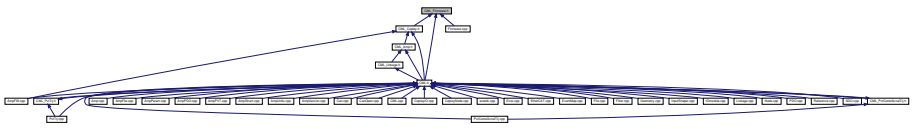
6.32 CML_Firmware.h File Reference

This file defines classes related to the Copley amplifier Firmware object.

Include dependency graph for CML_Firmware.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FirmwareError

  This class represents error conditions that can occur while accessing a Copley Controls amplifier firmware object.
- class Firmware

  Copley Controls amplifier firmware object.

6.32.1 Detailed Description

This file defines classes related to the Copley amplifier Firmware object.

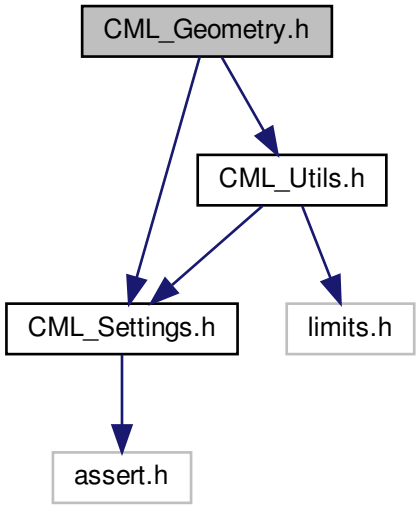
The firmware object is used to update the program within the amplifier. An object of this type must be passed to the Amp::FirmwareUpdate method to perform this task.

Note that firmware updates are likely to be rare, and are not part of normal operation.

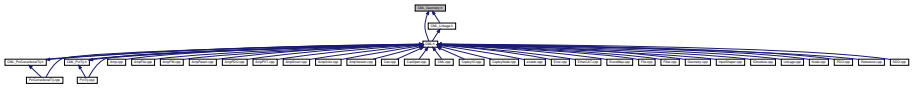
6.33 CML_Geometry.h File Reference

This file contains class definitions used to define multi-axis trajectory paths.

Include dependency graph for CML_Geometry.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PointN

  An N axis point.

- class Point< N >

  Template used for N dimensional objects.

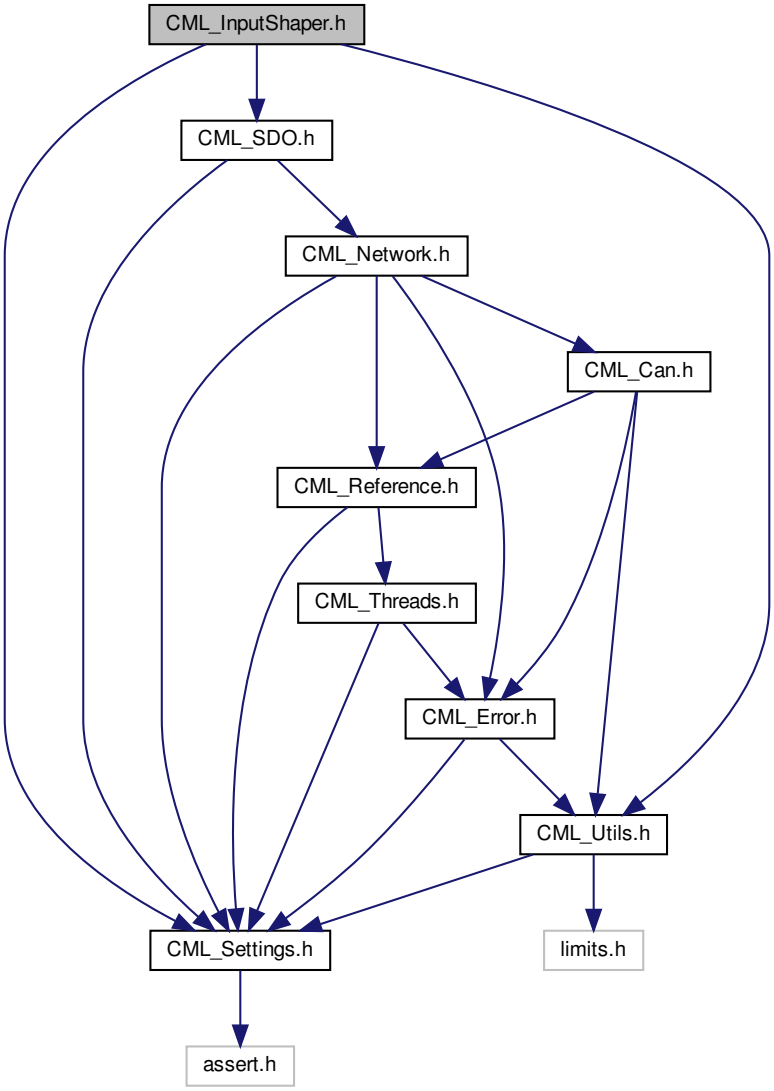
6.33.1 Detailed Description

This file contains class definitions used to define multi-axis trajectory paths.

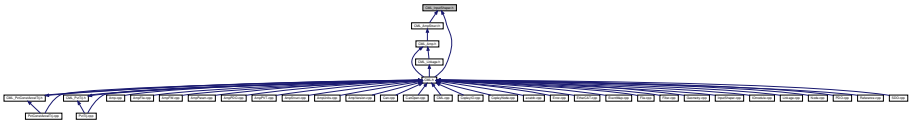
6.34 CML_InputShaper.h File Reference

This file defines the InputShaper object.

Include dependency graph for CML_InputShaper.h:



This graph shows which files directly or indirectly include this file:



Classes

- class InputShaper

  Generic input shaper structure.

6.34.1 Detailed Description

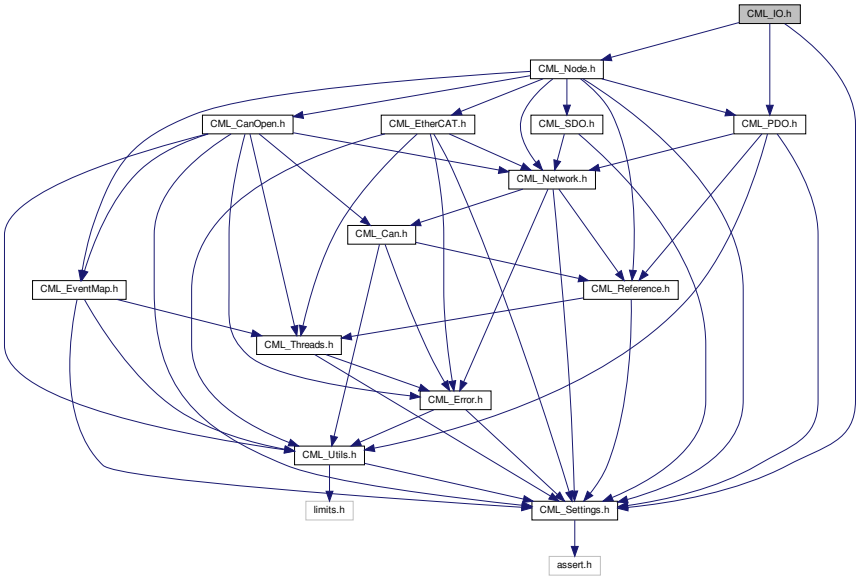
This file defines the InputShaper object.

The InputShaper represents a series of impulse functions convolved with an input function.

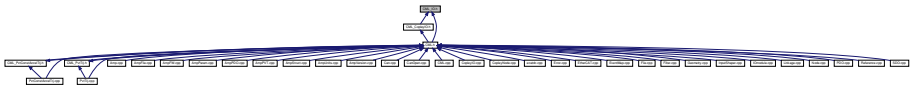
6.35 CML_IO.h File Reference

Standard CANopen I/O module support.

Include dependency graph for CML_IO.h:



This graph shows which files directly or indirectly include this file:



Classes

- class IOError

  I/O module errors.
- struct IOModuleSettings

  Standard CANopen I/O module settings.
- class IOModule

  Standard CANopen I/O module.
- class IOModule::DigOutPDO

  Receive PDO for mapping digital output pins.
- class IOModule::AlgOutPDO

  Receive PDO for mapping analog outputs.
- class IOModule::DigInPDO

  Transmit PDO for mapping digital inputs.
- class IOModule::AlgInPDO

  Transmit PDO for mapping analog inputs.

Enumerations

- enum IO_OBJID {
  IOOBJID_DIN_8_VALUE = 0x6000,
  IOOBJID_DIN_8_POL = 0x6002,
  IOOBJID_DIN_8_FILT = 0x6003,
  IOOBJID_DIN_INTENA = 0x6005,
  IOOBJID_DIN_8_MASK_ANY = 0x6006,
  IOOBJID_DIN_8_MASK_L2H = 0x6007,
  IOOBJID_DIN_8_MASK_H2L = 0x6008,
  IOOBJID_DIN_1_VALUE = 0x6020,
  IOOBJID_DIN_1_POL = 0x6030,
  IOOBJID_DIN_1_FILT = 0x6038,
  IOOBJID_DIN_1_MASK_ANY = 0x6050,
  IOOBJID_DIN_1_MASK_L2H = 0x6060,
  IOOBJID_DIN_1_MASK_H2L = 0x6070,
  IOOBJID_DIN_16_VALUE = 0x6100,
  IOOBJID_DIN_16_POL = 0x6102,
  IOOBJID_DIN_16_FILT = 0x6103,
  IOOBJID_DIN_16_MASK_ANY = 0x6106,
  IOOBJID_DIN_16_MASK_L2H = 0x6107,
  IOOBJID_DIN_16_MASK_H2L = 0x6108,
  IOOBJID_DIN_32_VALUE = 0x6120,
  IOOBJID_DIN_32_POL = 0x6122,
  IOOBJID_DIN_32_FILT = 0x6123,
  IOOBJID_DIN_32_MASK_ANY = 0x6126,
  IOOBJID_DIN_32_MASK_L2H = 0x6127,
  IOOBJID_DIN_32_MASK_H2L = 0x6128,
  IOOBJID_DOUT_8_VALUE = 0x6200,
  IOOBJID_DOUT_8_POL = 0x6202,
  IOOBJID_DOUT_8_ERRMODE = 0x6206,
  IOOBJID_DOUT_8_ERRVAL = 0x6207,
  IOOBJID_DOUT_8_FILT = 0x6208,
  IOOBJID_DOUT_1_VALUE = 0x6220,

IOOBJID_DOUT_1_POL = 0x6240,
IOOBJID_DOUT_1_ERRMODE = 0x6250,
IOOBJID_DOUT_1_ERRVAL = 0x6260,
IOOBJID_DOUT_1_FILT = 0x6270,
IOOBJID_DOUT_16_VALUE = 0x6300,
IOOBJID_DOUT_16_POL = 0x6302,
IOOBJID_DOUT_16_ERRMODE = 0x6306,
IOOBJID_DOUT_16_ERRVAL = 0x6307,
IOOBJID_DOUT_16_FILT = 0x6308,
IOOBJID_DOUT_32_VALUE = 0x6320,
IOOBJID_DOUT_32_POL = 0x6322,
IOOBJID_DOUT_32_ERRMODE = 0x6326,
IOOBJID_DOUT_32_ERRVAL = 0x6327,
IOOBJID_DOUT_32_FILT = 0x6328,
IOOBJID_AIN_8_VALUE = 0x6400,
IOOBJID_AIN_16_VALUE = 0x6401,
IOOBJID_AIN_32_VALUE = 0x6402,
IOOBJID_AIN_FLT_VALUE = 0x6403,
IOOBJID_AIN_MFG_VALUE = 0x6404,
IOOBJID_AOUT_8_VALUE = 0x6410,
IOOBJID_AOUT_16_VALUE = 0x6411,
IOOBJID_AOUT_32_VALUE = 0x6412,
IOOBJID_AOUT_FLT_VALUE = 0x6413,
IOOBJID_AOUT_MFG_VALUE = 0x6414,
IOOBJID_AIN_TRIG = 0x6421,
IOOBJID_AIN_INTSRC = 0x6422,
IOOBJID_AIN_INTENA = 0x6423,
IOOBJID_AIN_32_UPLIM = 0x6424,
IOOBJID_AIN_32_LWLIM = 0x6425,
IOOBJID_AIN_32_UDELTA = 0x6426,
IOOBJID_AIN_32_NDELTA = 0x6427,
IOOBJID_AIN_32_PDELTA = 0x6428,
IOOBJID_AIN_FLT_UPLIM = 0x6429,
IOOBJID_AIN_FLT_LWLIM = 0x642A,
IOOBJID_AIN_FLT_UDELTA = 0x642B,
IOOBJID_AIN_FLT_NDELTA = 0x642C,
IOOBJID_AIN_FLT_PDELTA = 0x642D,
IOOBJID_AIN_FLT_OFFSET = 0x642E,
IOOBJID_AIN_FLT_SCALE = 0x642F,
IOOBJID_AIN_UNIT = 0x6430,
IOOBJID_AIN_32_OFFSET = 0x6431,
IOOBJID_AIN_32_SCALE = 0x6432,
IOOBJID_AOUT_FLT_OFFSET = 0x6441,
IOOBJID_AOUT_FLT_SCALE = 0x6442,
IOOBJID_AOUT_ERRMODE = 0x6443,
IOOBJID_AOUT_32_ERRVAL = 0x6444,
IOOBJID_AOUT_FLT_ERRVAL = 0x6445,
IOOBJID_AOUT_32_OFFSET = 0x6446,
IOOBJID_AOUT_32_SCALE = 0x6447,
IOOBJID_AOUT_UNIT = 0x6450 }

Object dictionary ID values used on standard I/O modules.

- enum IO_AIN_TRIG_TYPE {
IOAINTRIG_UPPER_LIM = 0x0001,
IOAINTRIG_LOWER_LIM = 0x0002,

IOAINTRIG_UDELTA = 0x0004,
IOAINTRIG_NDELTA = 0x0008,
IOAINTRIG_PDELTA = 0x0010 }

This enumeration is used to define the types of events that may cause an analog input to generate an interrupt event.

- enum IOMODULE_EVENTS {
IOEVENT_DIN_PDO0 = 0x00000001,
IOEVENT_AIN_PDO0 = 0x00010000,
IOEVENT_AIN_PDO1 = 0x00020000,
IOEVENT_AIN_PDO2 = 0x00040000 }

This enumeration gives the various events that can be waited on.

6.35.1 Detailed Description

Standard CANopen I/O module support.

6.35.2 Enumeration Type Documentation

6.35.2.1 IO_AIN_TRIG_TYPE

`enum IO_AIN_TRIG_TYPE`

This enumeration is used to define the types of events that may cause an analog input to generate an interrupt event.

Enumerator

IOAINTRIG_UPPER_LIM	Input above upper limit.
IOAINTRIG_LOWER_LIM	Input below lower limit.
IOAINTRIG_UDELTA	Input changed by more then the unsigned delta amount.
IOAINTRIG_NDELTA	Input reduced by more then the negative delta amount.
IOAINTRIG_PDELTA	Input increased by more then the positive delta.

6.35.2.2 IO_OBJID

`enum IO_OBJID`

Object dictionary ID values used on standard I/O modules.

Enumerator

IOOBJID_DIN_8_VALUE	8-bit digital input value

Enumerator

IOOBJID_DIN_8_POL	8-bit digital input polarity
IOOBJID_DIN_8_FILT	8-bit digital input filter constant
IOOBJID_DIN_INTENA	Digital input interrupt enable.
IOOBJID_DIN_8_MASK_ANY	8-bit digital input int mask, any change
IOOBJID_DIN_8_MASK_L2H	8-bit digital input int mask, low to high
IOOBJID_DIN_8_MASK_H2L	8-bit digital input int mask, high to low
IOOBJID_DIN_1_VALUE	1-bit digital input value
IOOBJID_DIN_1_POL	1-bit digital input polarity
IOOBJID_DIN_1_FILT	1-bit digital input filter constant
IOOBJID_DIN_1_MASK_ANY	1-bit digital input int mask, any change
IOOBJID_DIN_1_MASK_L2H	1-bit digital input int mask, low to high
IOOBJID_DIN_1_MASK_H2L	1-bit digital input int mask, high to low
IOOBJID_DIN_16_VALUE	16-bit digital input value
IOOBJID_DIN_16_POL	16-bit digital input polarity
IOOBJID_DIN_16_FILT	16-bit digital input filter constant
IOOBJID_DIN_16_MASK_ANY	16-bit digital input int mask, any change
IOOBJID_DIN_16_MASK_L2H	16-bit digital input int mask, low to high
IOOBJID_DIN_16_MASK_H2L	16-bit digital input int mask, high to low
IOOBJID_DIN_32_VALUE	32-bit digital input value
IOOBJID_DIN_32_POL	32-bit digital input polarity
IOOBJID_DIN_32_FILT	32-bit digital input filter constant
IOOBJID_DIN_32_MASK_ANY	32-bit digital input int mask, any change
IOOBJID_DIN_32_MASK_L2H	32-bit digital input int mask, low to high
IOOBJID_DIN_32_MASK_H2L	32-bit digital input int mask, high to low
IOOBJID_DOUT_8_VALUE	8-bit digital output value
IOOBJID_DOUT_8_POL	8-bit digital output polarity
IOOBJID_DOUT_8_ERRMODE	8-bit digital output error mode
IOOBJID_DOUT_8_ERRVAL	8-bit digital output error value
IOOBJID_DOUT_8_FILT	8-bit digital output filter mask
IOOBJID_DOUT_1_VALUE	1-bit digital output value
IOOBJID_DOUT_1_POL	1-bit digital output polarity
IOOBJID_DOUT_1_ERRMODE	1-bit digital output error mode
IOOBJID_DOUT_1_ERRVAL	1-bit digital output error value
IOOBJID_DOUT_1_FILT	1-bit digital output filter mask
IOOBJID_DOUT_16_VALUE	16-bit digital output value
IOOBJID_DOUT_16_POL	16-bit digital output polarity
IOOBJID_DOUT_16_ERRMODE	16-bit digital output error mode
IOOBJID_DOUT_16_ERRVAL	16-bit digital output error value
IOOBJID_DOUT_16_FILT	16-bit digital output filter mask
IOOBJID_DOUT_32_VALUE	32-bit digital output value
IOOBJID_DOUT_32_POL	32-bit digital output polarity
IOOBJID_DOUT_32_ERRMODE	32-bit digital output error mode
IOOBJID_DOUT_32_ERRVAL	32-bit digital output error value

Enumerator

IOOBJID_DOUT_32_FILT	32-bit digital output filter mask
IOOBJID_AIN_8_VALUE	8-bit analog input value
IOOBJID_AIN_16_VALUE	16-bit analog input value
IOOBJID_AIN_32_VALUE	32-bit analog input value
IOOBJID_AIN_FLT_VALUE	floating point analog input value
IOOBJID_AIN_MFG_VALUE	manufacturer specific analog input value
IOOBJID_AOUT_8_VALUE	8-bit analog output value
IOOBJID_AOUT_16_VALUE	16-bit analog output value
IOOBJID_AOUT_32_VALUE	32-bit analog output value
IOOBJID_AOUT_FLT_VALUE	floating point analog output value
IOOBJID_AOUT_MFG_VALUE	manufacturer specific analog output value
IOOBJID_AIN_TRIG	Analog input trigger selection.
IOOBJID_AIN_INTSRC	Analog input interrupt source.
IOOBJID_AIN_INTENA	Analog input interrupt enable.
IOOBJID_AIN_32_UPLIM	32-bit analog input upper limit
IOOBJID_AIN_32_LWLIM	32-bit analog input lower limit
IOOBJID_AIN_32_UDELTA	32-bit analog input unsigned delta
IOOBJID_AIN_32_NDELTA	32-bit analog input negative delta
IOOBJID_AIN_32_PDELTA	32-bit analog input positive delta
IOOBJID_AIN_FLT_UPLIM	floating point analog input upper limit
IOOBJID_AIN_FLT_LWLIM	floating point analog input lower limit
IOOBJID_AIN_FLT_UDELTA	floating point analog input unsigned delta
IOOBJID_AIN_FLT_NDELTA	floating point analog input negative delta
IOOBJID_AIN_FLT_PDELTA	floating point analog input positive delta
IOOBJID_AIN_FLT_OFFSET	floating point analog input offset
IOOBJID_AIN_FLT_SCALE	floating point analog input scaling
IOOBJID_AIN_UNIT	analog input SI Unit
IOOBJID_AIN_32_OFFSET	32-bit analog input offset
IOOBJID_AIN_32_SCALE	32-bit analog input scaling
IOOBJID_AOUT_FLT_OFFSET	floating point analog output offset
IOOBJID_AOUT_FLT_SCALE	floating point analog output scaling
IOOBJID_AOUT_ERRMODE	analog output error mode
IOOBJID_AOUT_32_ERRVAL	32-bit analog output error value
IOOBJID_AOUT_FLT_ERRVAL	floating point analog output error value
IOOBJID_AOUT_32_OFFSET	32-bit analog output offset
IOOBJID_AOUT_32_SCALE	32-bit analog output scaling
IOOBJID_AOUT_UNIT	analog output SI Unit

6.35.2.3 IOMODULE_EVENTS

```
enum IOMODULE_EVENTS
```

This enumeration gives the various events that can be waited on.

The default events are simply the reception of one of the standard transmit PDO objects.

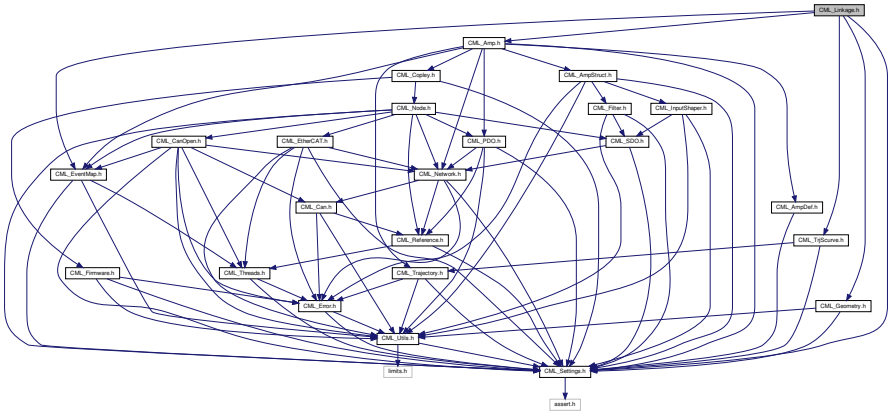
Enumerator

IOEVENT_DIN_PDO0	Digital input PDO 0 was received. By default, this PDO is transmitted by the module when any of the first 64 digital inputs changes state.
IOEVENT_AIN_PDO0	Analog input PDO 0 was received. By default, this PDO is transmitted by the module when any of the first 4 16-bit analog inputs generates an event. There are many different types of events that are programmable for analog inputs, however not all I/O module manufacturers support all (or any) of these events. The function IOmodule::AinSetTrigType can be used to set the type of event associated with an analog input. Consult the documentation provided with the I/O module to determine what types of analog input events are available for your module.
IOEVENT_AIN_PDO1	Analog input PDO 1 was received. This PDO is similar to analog input PDO 0, however it maps the second group of 4 16-bit analog inputs.
IOEVENT_AIN_PDO2	Analog input PDO 2 was received. This PDO is similar to analog input PDO 0, however it maps the third group of 4 16-bit analog inputs.

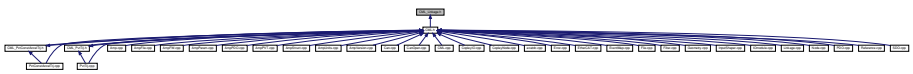
6.36 CML_Linkage.h File Reference

This file defines the Linkage object.

Include dependency graph for CML_Linkage.h:



This graph shows which files directly or indirectly include this file:



Classes

- class LinkError

  This class represents error conditions that can occur in the Linkage class.
- class RPDO_LinkCtrl

  Receive PDO used to update the control word of all amplifiers in the linkage.
- class LinkSettings

  Linkage object settings.
- class Linkage

  Linkage object, used for controlling a group of coordinated amplifiers.

Enumerations

- enum LINK_EVENT {
  LINKEVENT_MOVEDONE = 0x00000001,
  LINKEVENT_TRJDONE = 0x00000002,
  LINKEVENT_NODEGUARD = 0x00000004,
  LINKEVENT_FAULT = 0x00000010,
  LINKEVENT_ERROR = 0x00000020,
  LINKEVENT_POSWARN = 0x00000040,
  LINKEVENT_POSWIN = 0x00000080,
  LINKEVENT_VELWIN = 0x00000100,
  LINKEVENT_DISABLED = 0x00000200,
  LINKEVENT_POSLIM = 0x00000400,
  LINKEVENT_NEGLIM = 0x00000800,
  LINKEVENT_SOFTLIM_POS = 0x00001000,
  LINKEVENT_SOFTLIM_NEG = 0x00002000,
  LINKEVENT_QUICKSTOP = 0x00004000,
  LINKEVENT_ABORT = 0x00008000,
  LINKEVENT_TRSTP = 0x00100000 }

  Linkage events.

6.36.1 Detailed Description

This file defines the Linkage object.

A linkage is a group of two or more amplifiers which work together.

6.36.2 Enumeration Type Documentation

6.36.2.1 LINK_EVENT

`enum LINK_EVENT`

Linkage events.

This enumeration provides a list of events that can be used to wait on linkage conditions.

In general, linkage events parallel the amplifier events of all of the amplifiers attached to the linkage. For example, if any of the amplifiers is reporting an error event, then the linkage will be reporting an error event.

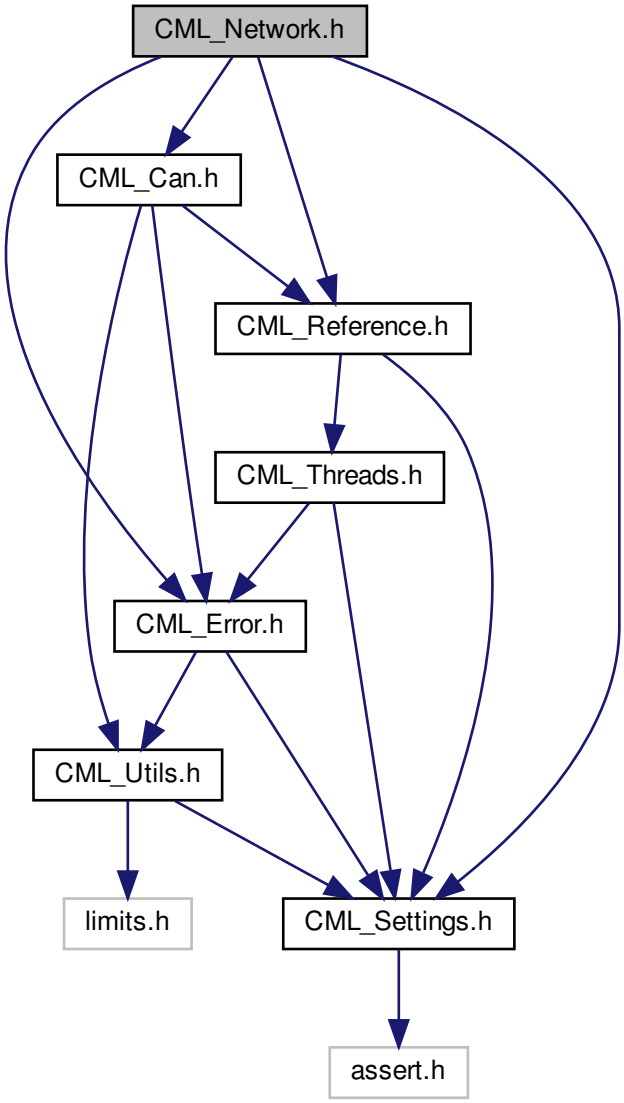
Enumerator

LINKEVENT_MOVEDONE	Set when all amplifiers attached to this linkage have finished their moves and have settled in to position at the end of the move. Cleared when a new move is started on any amplifier.
LINKEVENT_TRJDONE	Set when all amplifiers attached to the linkage have finished their moves, but have not yet settled into position at the end of the move. Cleared when a new move is on any amplifier started.
LINKEVENT_NODEGUARD	A node guarding (or heartbeat) error has occurred. This indicates that one of the amplifiers failed to respond within the expected amount of time for either a heartbeat or node guarding message.
LINKEVENT_FAULT	A latching fault has occurred on one of the amplifiers attached to this linkage.
LINKEVENT_ERROR	A non-latching error has occurred on one of the amplifiers.
LINKEVENT_POSWARN	One of the the amplifiers is reporting a position warning event.
LINKEVENT_POSWIN	One of the amplifiers is reporting a position window event.
LINKEVENT_VELWIN	One of the amplifiers is reporting a velocity window event.
LINKEVENT_DISABLED	One of the amplifiers is currently disabled.
LINKEVENT_POSLIM	The positive limit switch of one or more amplifier is currently active.
LINKEVENT_NEGLIM	The negative limit switch of one or more amplifier is currently active.
LINKEVENT_SOFTLIM_POS	The positive software limit of one or more amplifier is currently active.
LINKEVENT_SOFTLIM_NEG	The negative software limit of one or more amplifier is currently active.
LINKEVENT_QUICKSTOP	One of the linkage amplifiers is presently performing a quick stop sequence or is holding in quick stop mode. The amplifier must be disabled to clear this.
LINKEVENT_ABORT	One or more amplifier aborted the last profile without finishing.
LINKEVENT_TRSTP	One or more amplifier is trying to stop the motor.

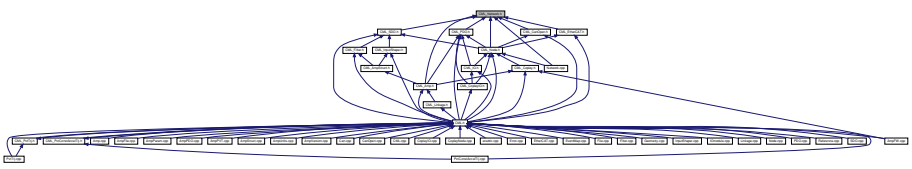
6.37 CML_Network.h File Reference

This header file defines the classes used for the generic top level network interface.

Include dependency graph for CML_Network.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NetworkError

  This class holds the error codes that describe various Netowrk error conditions.
- class NetworkNodeInfo

  Private data owned by the network object attached to every node.
- class Network

  Abstract network class.

Enumerations

- enum NodeState {
  NODESTATE_INVALID,
  NODESTATE_UNKNOWN,
  NODESTATE_GUARDERR,
  NODESTATE_STOPPED,
  NODESTATE_PRE_OP,
  NODESTATE_OPERATIONAL,
  NODESTATE_SAFE_OP }

  Enumeration used to identify a network node state.
- enum NetworkType {
  NET_TYPE_CANOPEN,
  NET_TYPE_ETHERCAT,
  NET_TYPE_INVALID }

  Enumeration used to identify a type of network architecture.
- enum GuardProtocol {
  GUARDTYPE_NONE,
  GUARDTYPE_HEARTBEAT,
  GUARDTYPE_NODEGUARD }

  Enumeration used to identify the various types of node guarding protocols.

6.37.1 Detailed Description

This header file defines the classes used for the generic top level network interface.

6.37.2 Enumeration Type Documentation

6.37.2.1 GuardProtocol

`enum GuardProtocol`

Enumeration used to identify the various types of node guarding protocols.

Enumerator

GUARDTYPE_NONE	No guarding protocol is in use.
GUARDTYPE_HEARTBEAT	The heartbeat protocol is being used.
GUARDTYPE_NODEGUARD	Node guarding protocol is being used.

6.37.2.2 NetworkType

`enum NetworkType`

Enumeration used to identify a type of network architecture.

Enumerator

NET_TYPE_CANOPEN	CANopen network.
NET_TYPE_ETHERCAT	EtherCAT network.
NET_TYPE_INVALID	Invalid network type.

6.37.2.3 NodeState

`enum NodeState`

Enumeration used to identify a network node state.

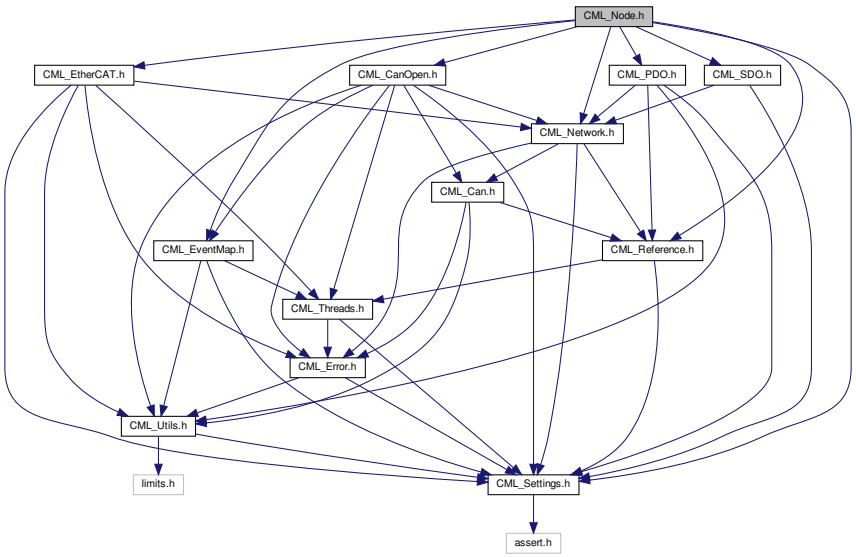
Enumerator

NODESTATE_INVALID	Invalid node state.
NODESTATE_UNKNOWN	Unknown state - the default state on node creation. The state will be changed when communication with the node is established.
NODESTATE_GUARDERR	On a node guarding or heartbeat timeout, the state will change to guard error.
NODESTATE_STOPPED	Stopped state (aka init state for EtherCAT)
NODESTATE_PRE_OP	Pre-operational state.
NODESTATE_OPERATIONAL	Operational state.
NODESTATE_SAFE_OP	Safe-operational state.

6.38 CML_Node.h File Reference

This header file defines the classes that define a generic node on the network.

Include dependency graph for CML_Node.h:



This graph shows which files directly or indirectly include this file:

Classes

- class NodeError

  This class represents node errors.
- struct NodeIdentity

  CANopen identity object.
- class Node

  Node class.

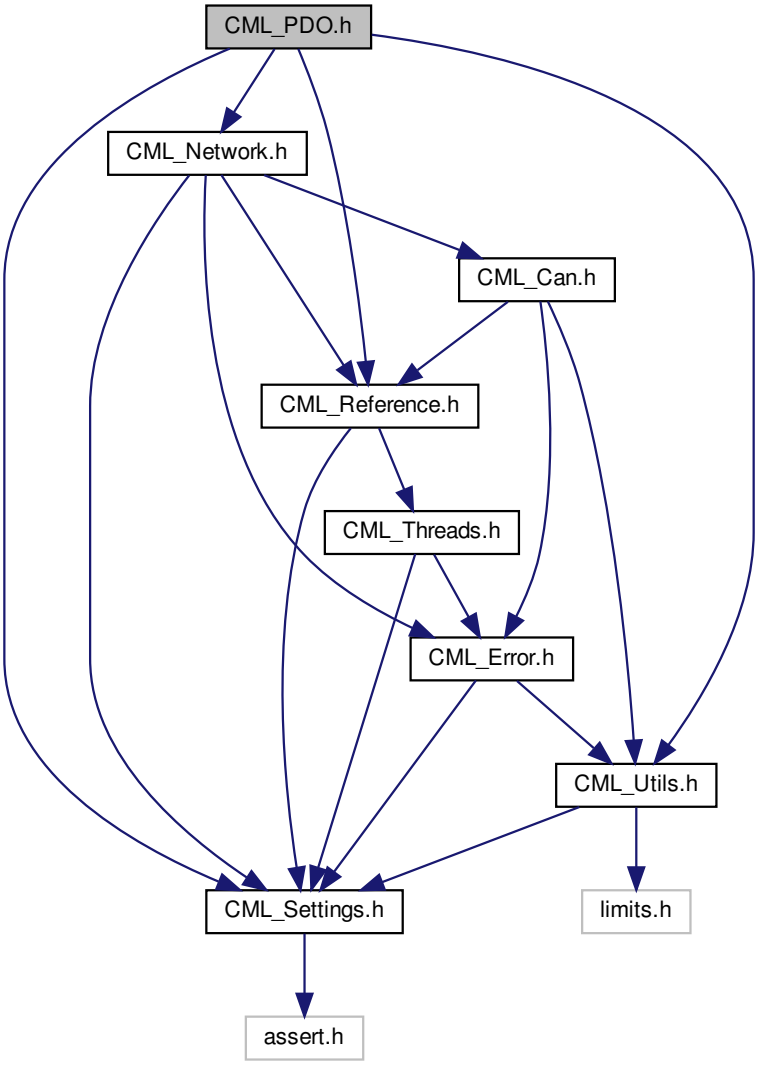
6.38.1 Detailed Description

This header file defines the classes that define a generic node on the network.

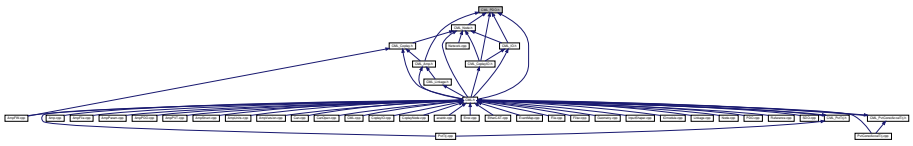
6.39 CML_PDO.h File Reference

This header file defines the classes used to communicate to CANopen nodes using Process Data Objects (PDOs).

Include dependency graph for CML_PDO.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PDO_Error

  This class represents error conditions related to PDOs.
- class Pmap

  This class allows variables to be mapped into a PDO.
- class PmapRaw

  This is the most generic PDO variable mapping class.
- class Pmap32

  This is a PDO variable mapping class that extends the virtual Pmap class to handle 32-bit integers.
- class Pmap24

  This is a PDO variable mapping class that extends the virtual Pmap class to handle 24-bit integers.
- class Pmap16

  This is a PDO variable mapping class that extends the virtual Pmap class to handle 16-bit integers.
- class Pmap8

  This is a PDO variable mapping class that extends the virtual Pmap class to handle 8-bit integers.
- class PDO

  PDO (Process Data Object) base class.
- class TPDO

  Transmit PDO (transmitted by node, received by this software).
- class RPDO

  Receive PDO (received by node, transmitted by this software).

Macros

- #define PDO_MAP_LEN 32

  Number of variables that may be added to a PDO's map.
- #define FLG_RTR_OK 0x01

  Flag used by transmit PDOs to indicate that RTR requests are allowed.

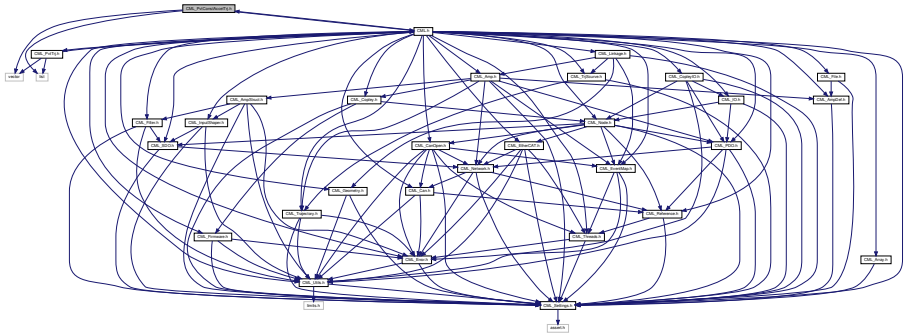
6.39.1 Detailed Description

This header file defines the classes used to communicate to CANopen nodes using Process Data Objects (PDOs).

6.40 CML_PvtConstAccelTrj.h File Reference

This header file defines the classes that define the PvtConstAccelTrj class.

Include dependency graph for CML_PvtConstAccelTrj.h:



This graph shows which files directly or indirectly include this file:

Classes

- class PvtConstAccelTrjError
- class PvtConstAccelTrj

6.40.1 Detailed Description

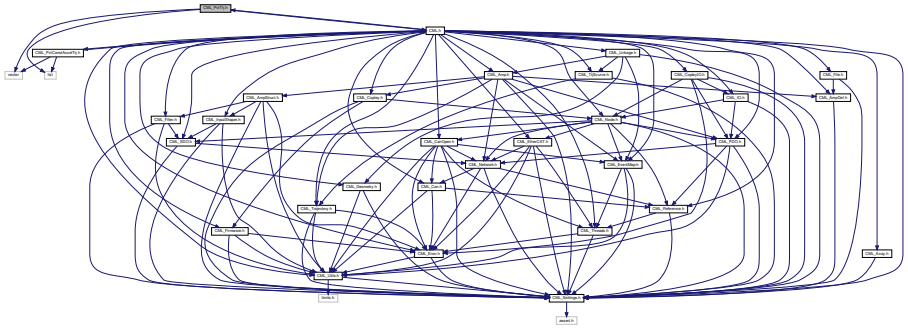
This header file defines the classes that define the PvtConstAccelTrj class.

The class is derived from the Linkage Trajectory class and is used to calculate a set of velocities that produce constant acceleration from a given set of positions and time constants.

6.41 CML_PvtTrj.h File Reference

This header file defines the classes that define the PvtTrj class.

Include dependency graph for CML_PvtTrj.h:



This graph shows which files directly or indirectly include this file:

Classes

- class PvtTrjError
- class PvtTrj

6.41.1 Detailed Description

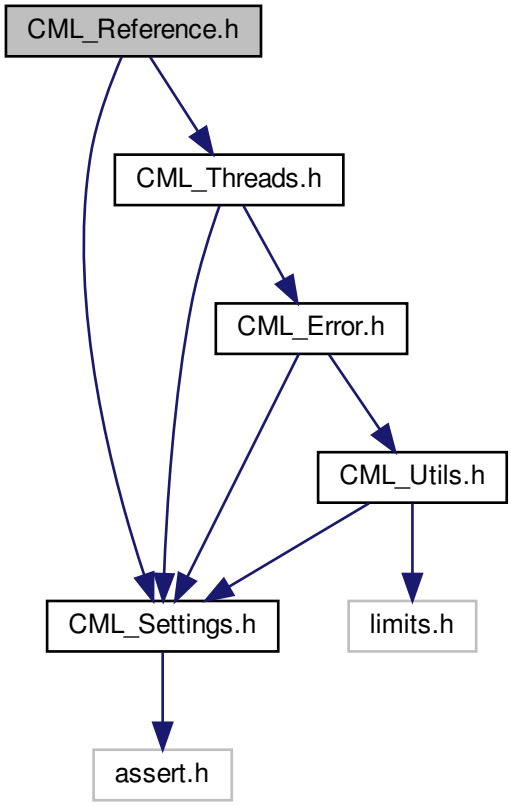
This header file defines the classes that define the PvtTrj class.

The class is derived from the Linkage Trajectory class and is send velocity, position, and time data to the trajectory generator.

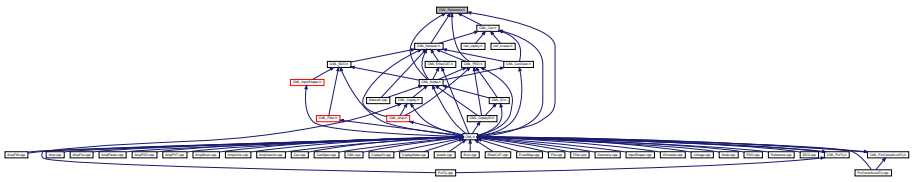
6.42 CML_Reference.h File Reference

This header file defines a set of classes used to handle reference counting within the CML library.

Include dependency graph for CML_Reference.h:



This graph shows which files directly or indirectly include this file:



Classes

- class RefObj

  This class is used to track object references in the CML library.
- class RefObjLocker< RefClass >

  This is a utility class that locks a reference in it's constructor, and unlocks it in it's destructor.

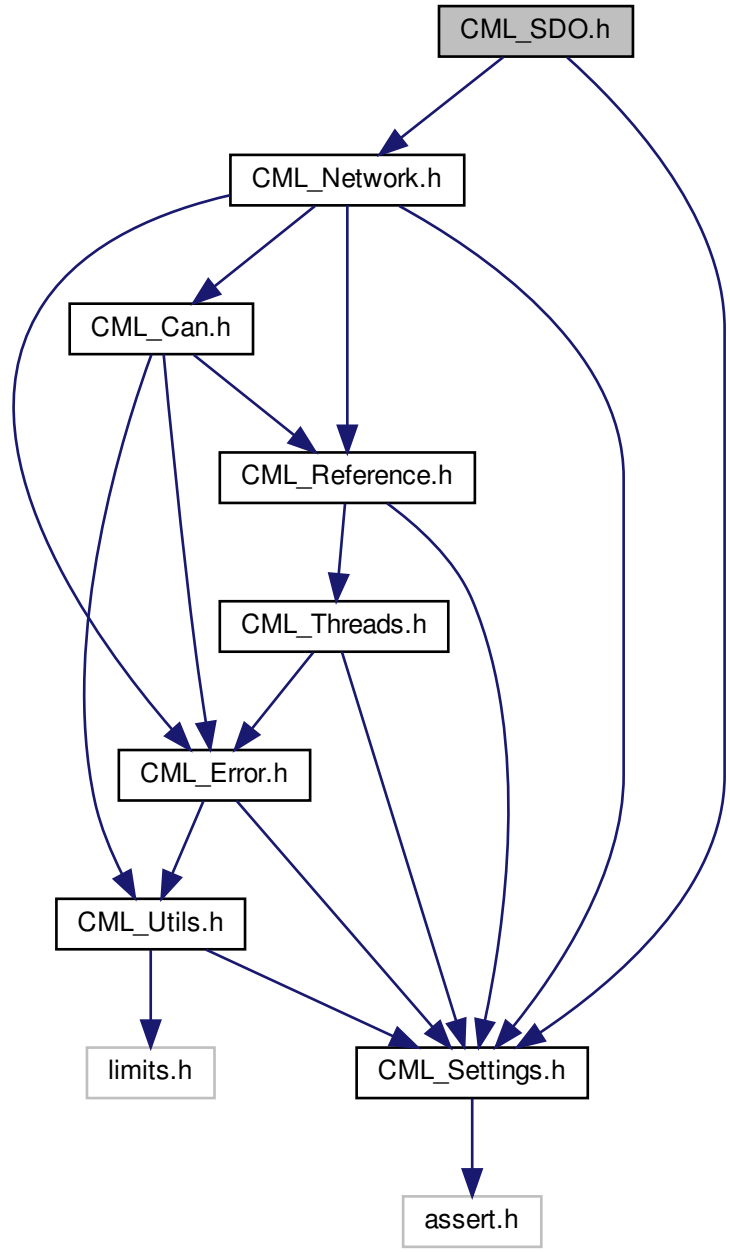
6.42.1 Detailed Description

This header file defines a set of classes used to handle reference counting within the CML library.

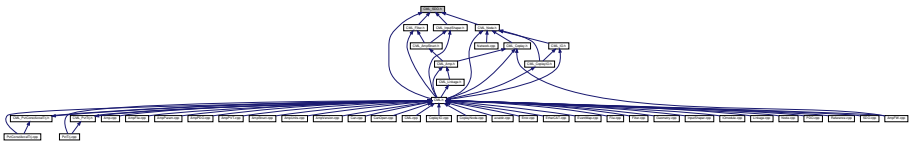
6.43 CML_SDO.h File Reference

This header file defines the classes used to communicate to CANopen nodes using Service Data Objects (SDOs).

Include dependency graph for CML_SDO.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SDO_Error

  This class represents SDO errors.
- class SDO

  CANopen Service Data Object (SDO).

Macros

- #define SDO_BLK_DNLD_THRESHOLD 15

  SDO downloads of this size (byte) or greater are more efficiently handled using a block download.
- #define SDO_BLK_UPLD_THRESHOLD 15

  SDO uploads of this size (byte) or greater are more efficiently handled using a block upload.

6.43.1 Detailed Description

This header file defines the classes used to communicate to CANopen nodes using Service Data Objects (SDOs).

6.43.2 Macro Definition Documentation

6.43.2.1 SDO_BLK_DNLD_THRESHOLD

```
#define SDO_BLK_DNLD_THRESHOLD 15
```

SDO downloads of this size (byte) or greater are more efficiently handled using a block download.

6.43.2.2 SDO_BLK_UPLD_THRESHOLD

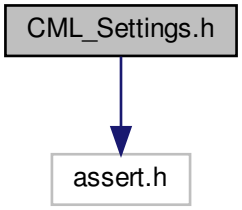
```
#define SDO_BLK_UPLD_THRESHOLD 15
```

SDO uploads of this size (byte) or greater are more efficiently handled using a block upload.

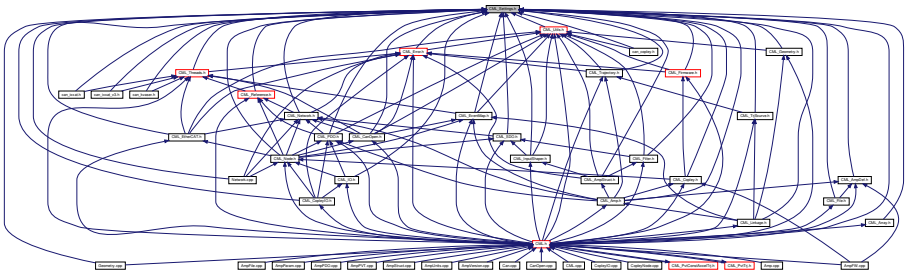
6.44 CML_Settings.h File Reference

This file provides some configuration options used to customize the Copley Motion Libraries.

Include dependency graph for CML_Settings.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define CML_NAMESPACE CML

  Library namespace.
- #define CML_HASH_SIZE 1483

  Size of the hash table used to associate CAN messages with their receivers.
- #define CML_FILE_ACCESS_OK

  Enable file access.
- #define CML_ALLOW_FLOATING_POINT

  Allow use of floating point math.
- #define CML_ENABLE_USER_UNITS

  Enable user units.
- #define CML_DEBUG_ASSERT

  Enable debug assertions.
- #define CML_MAX_AMPS_PER_LINK 32

This defines the maximum number of amplifiers that may be controlled by a single linkage object.

- #define CML_LINKAGE_TRJ_BUFFER_SIZE 50

  This parameter controls the size of the trajectory buffer used by the linkage object.

- #define CML_ERROR_HASH_SIZE 64

  Size of the hash table used by the Error::Lookup method.

- #define CML_ERROR_MESSAGES

  The CML::Error object includes a text message for each error type.

- #define CML_ENABLE_IOMODULE_PDOS

  This setting enables/disables the use of PDO objects within the IOModule class.

- #define CML_MAX_ECAT_FRAMES 100

  Size of extra private data for the Thread object.

- #define CML_NAMESPACE_START() namespace CML_NAMESPACE{

  Compiler native type to use for 64-bit integer.

6.44.1 Detailed Description

This file provides some configuration options used to customize the Copley Motion Libraries.

6.44.2 Macro Definition Documentation

6.44.2.1 CML_ALLOW_FLOATING_POINT

```
#define CML_ALLOW_FLOATING_POINT
```

Allow use of floating point math.

If this is defined, double precision floating point math will be used in some areas of the libraries. These areas include trajectory generation and unit conversions. If not defined, then no floating point math will be used, but some features will be disabled.

6.44.2.2 CML_DEBUG_ASSERT

```
#define CML_DEBUG_ASSERT
```

Enable debug assertions.

If this is defined, then some debug code will be added to the library which will use the standard C assert() function to test for some programming errors. Commenting this out will remove the checks. The standard C header file assert.h must be available if this feature is used.

6.44.2.3 CML_ENABLE_IOMODULE_PDOS

```
#define CML_ENABLE_IOMODULE_PDOS
```

This setting enables/disables the use of PDO objects within the IOModule class.

The IOModule class is used to access standard CANopen I/O modules on the network. Normally, these modules may be accessed using fast PDO transfers, however in very low memory embedded systems the extra RAM required to maintain the PDO objects may not be available. In such situations this setting may be commented out to reduce the memory footprint of the IOModule class.

6.44.2.4 CML_ENABLE_USER_UNITS

```
#define CML_ENABLE_USER_UNITS
```

Enable user units.

If this is defined, then all position, velocity, acceleration & jerk values will be specified in double precision floating point, and the units used for these values will be programmable. If not defined, then these parameters will all be specified as 32-bit integers using internal amplifier parameters. This is less convenient, but can be much faster for systems without a floating point processor.

6.44.2.5 CML_ERROR_HASH_SIZE

```
#define CML_ERROR_HASH_SIZE 64
```

Size of the hash table used by the Error::Lookup method.

This may be set to zero to disable this feature. Most systems will not require this feature and can safely set this parameter to zero.

6.44.2.6 CML_ERROR_MESSAGES

```
#define CML_ERROR_MESSAGES
```

The CML::Error object includes a text message for each error type.

If this setting is commented out then those messages will not be compiled in with the library. This can be useful for embedded environments where such messages are not used and represent a large amount of wasted memory.

6.44.2.7 CML_FILE_ACCESS_OK

```
#define CML_FILE_ACCESS_OK
```

Enable file access.

The libraries have some features which require the standard C library functions to open, read, and write files. Some embedded systems do not support a file system, so these features may be disabled by commenting out the define. For systems which do support the standard C file access functions, this should be enabled.

6.44.2.8 CML_HASH_SIZE

```
#define CML_HASH_SIZE 1483
```

Size of the hash table used to associate CAN messages with their receivers.

Larger tables give faster access, but use more memory.

The following values have been selected as good options for a typical CANopen system: 2053, 1483, 1097, 683, 409.

6.44.2.9 CML_LINKAGE_TRJ_BUFFER_SIZE

```
#define CML_LINKAGE_TRJ_BUFFER_SIZE 50
```

This parameter controls the size of the trajectory buffer used by the linkage object.

The linkage object uses this buffer when streaming multi-axis PVT profiles. If multi-axis PVT profiles are not required then setting may be commented out. Doing so will significantly reduce the size of the linkage object. If multi-axis PVTs are required, set this to the length of the buffer. A value of 50 is a reasonable choice.

6.44.2.10 CML_MAX_AMPS_PER_LINK

```
#define CML_MAX_AMPS_PER_LINK 32
```

This defines the maximum number of amplifiers that may be controlled by a single linkage object.

The absolute maximum value that this can accept is 32, however it can be lowered to reduce the memory requirements of the library. This setting also limits the number of independent axes/link. Normally, the number of amps & axes is the same, but the Linkage object may be extended for conditions where this isn't true. In any case, there can be no more then this many axes/link.

6.44.2.11 CML_MAX_ECAT_FRAMES

```
#define CML_MAX_ECAT_FRAMES 100
```

Size of extra private data for the Thread object.

This setting should be left undefined for most systems. It's provided to allow greater flexibility when porting the libraries to another operating system. Size of extra private data for the Mutex object. This setting should be left undefined for most systems. It's provided to allow greater flexibility when porting the libraries to another operating system. Size of extra private data for the Semaphore object. This setting should be left undefined for most systems. It's provided to allow greater flexibility when porting the libraries to another operating system. Number of EtherCAT frames that each EtherCAT interface can keep track of at a time.

6.44.2.12 CML_NAMESPACE

```
#define CML_NAMESPACE CML
```

Library namespace.

This gives the name of the C++ namespace which will be used to contain the library. If no namespace is desired, just comment out the define

6.44.2.13 CML_NAMESPACE_START

```
#define CML_NAMESPACE_START( ) namespace CML_NAMESPACE{
```

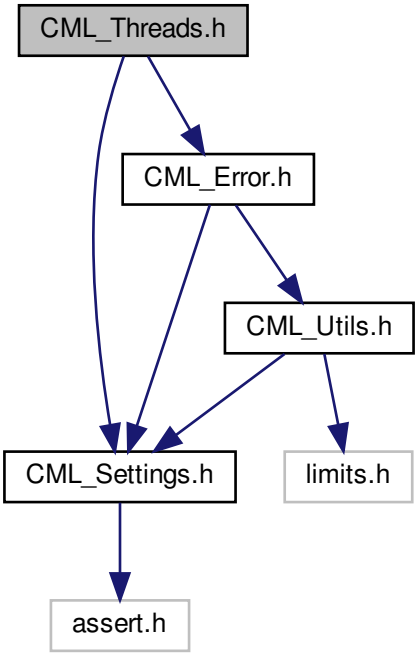
Compiler native type to use for 64-bit integer.

Normally this doesn't need to be defined, but if you're getting compiler errors related to the int64 type you can set this Some C++ compilers support the C99 style stdint.h header which defines variables of standard sizes. Uncomment this line to use that header to define the 64-bit integer type. If this line is uncommented, then CML_INT64_TYPE is ignored.

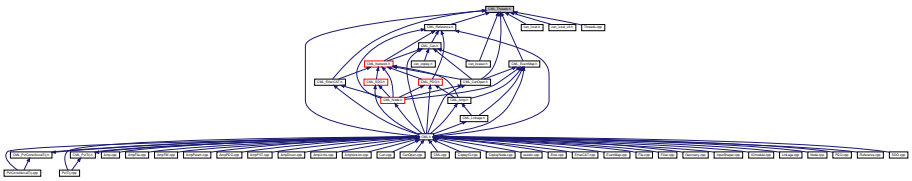
6.45 CML_Threads.h File Reference

The classes defined in this file provide an operating system independent way of accessing multi-tasking system features.

Include dependency graph for CML_Threads.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ThreadError

  Errors related to the multi-threaded libraries.

- class Thread

  Virtual class which provides multi-tasking.

- class Mutex

  This class represents an object that can be used by multiple threads to gain safe access to a shared resource.

- class MutexLocker

  This is a utility class that locks a mutex in it's constructor, and unlocks it in it's destructor.

- class Semaphore

  Generic semaphore class.

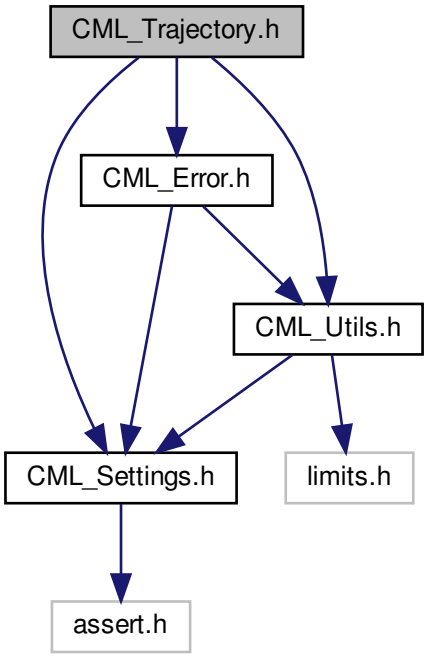
6.45.1 Detailed Description

The classes defined in this file provide an operating system independent way of accessing multi-tasking system features.

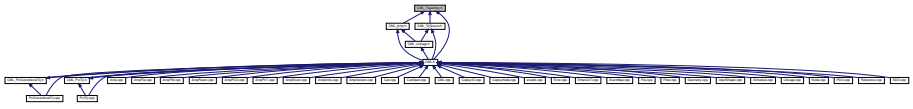
The implementation of these classes will be different for different supported platforms.

6.46 CML_Trajectory.h File Reference

Include dependency graph for CML_Trajectory.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TrjError

  This class represents error conditions reported by the trajectory classes.
- class Trajectory

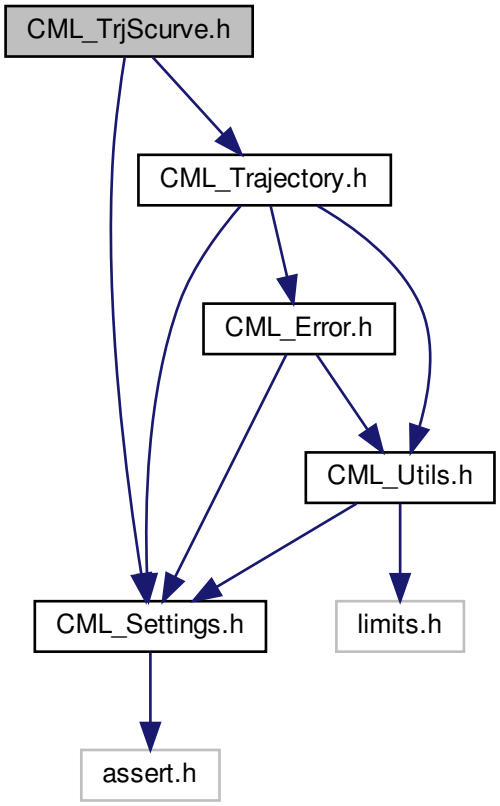
  Trajectory information class.
- class LinkTrajectory

  Linkage trajectory.

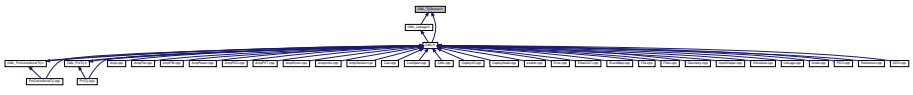
6.47 CML_TrjScurve.h File Reference

This file defines the TrjScurve class.

Include dependency graph for CML_TrjScurve.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ScurveError

  This class represents error conditions that can occur in the TrjScurve class.

- class TrjScurve

  Asymmetric S-curve profile generator.
- class LinkTrjScurve

  Multi-axis s-curve profile.

6.47.1 Detailed Description

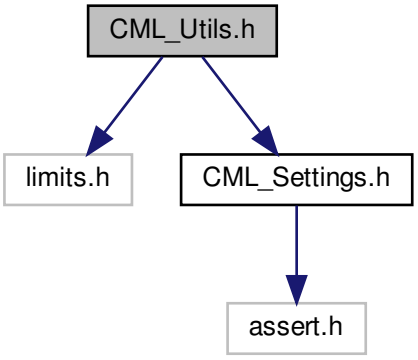
This file defines the TrjScurve class.

This class is used to calculate asymmetric S-curve trajectory profiles for use by the Amp or Linkage objects.

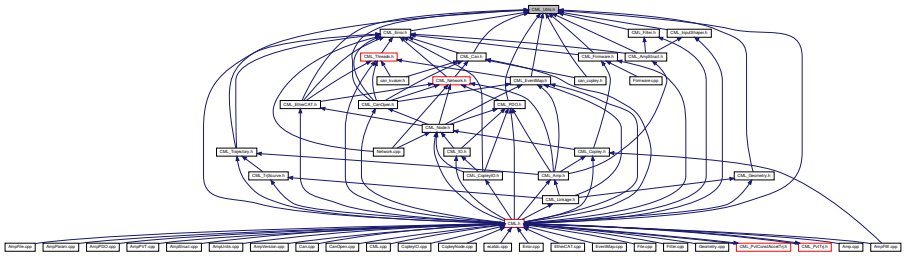
6.48 CML_Utils.h File Reference

This file holds various handy utility types and functions.

Include dependency graph for CML_Utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ByteCast(x) ((byte)(x))

  The ByteCast() macro is used to cast a value to a byte type (unsigned char).

Typedefs

- typedef unsigned char uchar

  unsigned character
- typedef unsigned char byte

  unsigned character
- typedef unsigned int uint

  unsigned integer type
- typedef unsigned long ulong

  unsigned long type
- typedef signed char int8

  8-bit integer type
- typedef unsigned char uint8

  8-bit unsigned integer type
- typedef short int16

  Signed 16-bit integer type.
- typedef unsigned short uint16

  16-bit unsigned integer type
- typedef int int32

  Signed 32-bit integer type.
- typedef unsigned int uint32

  Unsigned 32-bit integer type.
- typedef CML_INT64_TYPE int64

  Signed 64-bit integer type.
- typedef double uunit

  User programmable unit.

6.48.1 Detailed Description

This file holds various handy utility types and functions.

6.48.2 Macro Definition Documentation

6.48.2.1 ByteCast

```
#define ByteCast(
              x ) ((byte)(x))
```

The ByteCast() macro is used to cast a value to a byte type (unsigned char).

The reason that a macro is used rather then a simple cast is that some processors have characters that are more then 8 bits long. This is particularly common with 16-bit or 32-bit microcontrollers and DSPs. On such systems the ByteCast macro will strip off any upper bits and then cast the result to a byte. On systems with 8-bit bytes the ByteCast macro simply casts the passed value to a byte.

6.48.3 Typedef Documentation

6.48.3.1 int16

```
int16
```

Signed 16-bit integer type.

Note that the actual definition of this type will depend on the compiler being used. The standard C language header file limits.h will be used to determine how to create the type definition.

6.48.3.2 int32

```
int32
```

Signed 32-bit integer type.

Note that the actual definition of this type will depend on the compiler being used. The standard C language header file limits.h will be used to determine how to create the type definition.

6.48.3.3 int64

```
int64
```

Signed 64-bit integer type.

If this isn't working correctly, check the library configuration in the file CML_Settings.h

6.48.3.4 uint16

`uint16`

16-bit unsigned integer type

Unsigned 16-bit integer type.

Note that the actual definition of this type will depend on the compiler being used. The standard C language header file limits.h will be used to determine how to create the type definition.

6.48.3.5 uint32

`uint32`

Unsigned 32-bit integer type.

Note that the actual definition of this type will depend on the compiler being used. The standard C language header file limits.h will be used to determine how to create the type definition.

6.48.3.6 uunit

`uunit`

User programmable unit.

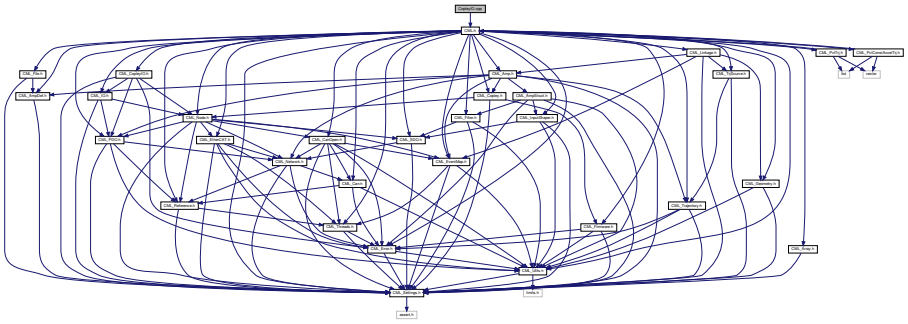
If user units are enabled in the CML_Settings.h file, then this type will resolve to double precision floating point. If not, then this type will resolve to a 32-bit integer.

User units are used for all position, velocity, acceleration, and jerk values passed to/from an Amp object.

6.49 CopleyIO.cpp File Reference

This file contains the CopleyIO object methods used to upload / download structures containing groups of module parameters.

Include dependency graph for CopleyIO.cpp:



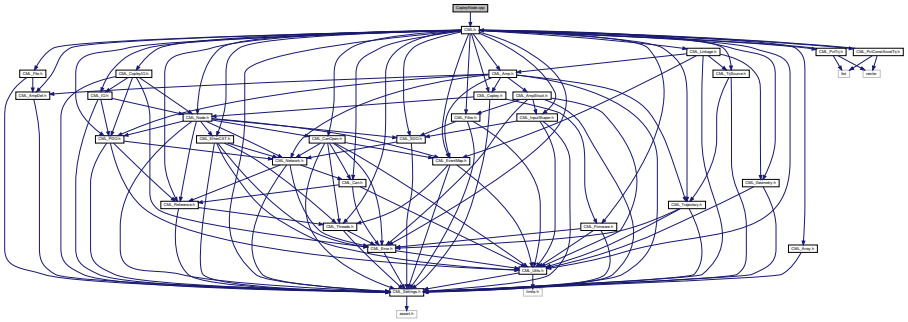
6.49.1 Detailed Description

This file contains the CopleyIO object methods used to upload / download structures containing groups of module parameters.

6.50 CopleyNode.cpp File Reference

This file holds code to implement the CopleyNode object.

Include dependency graph for CopleyNode.cpp:



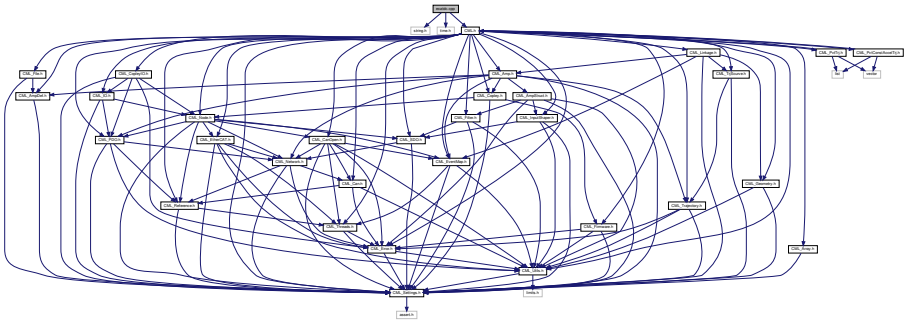
6.50.1 Detailed Description

This file holds code to implement the CopleyNode object.

6.51 ecatdc.cpp File Reference

This file holds some utility code used by the EtherCAT network when initializing it's distributed clock.

Include dependency graph for ecatdc.cpp:



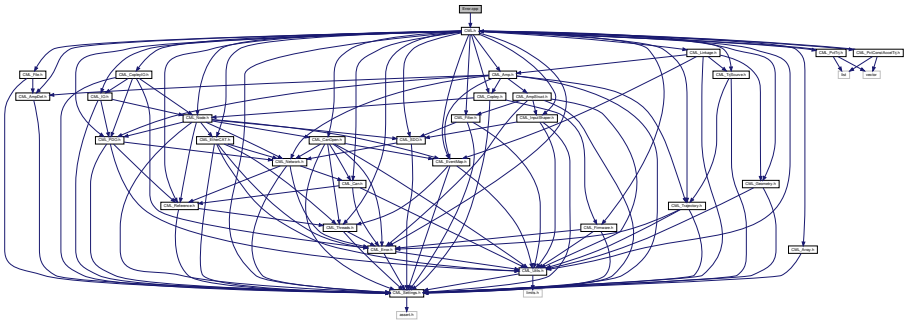
6.51.1 Detailed Description

This file holds some utility code used by the EtherCAT network when initializing it's distributed clock.

6.52 Error.cpp File Reference

This file handles initializing the static data objects used by the Error class.

Include dependency graph for Error.cpp:



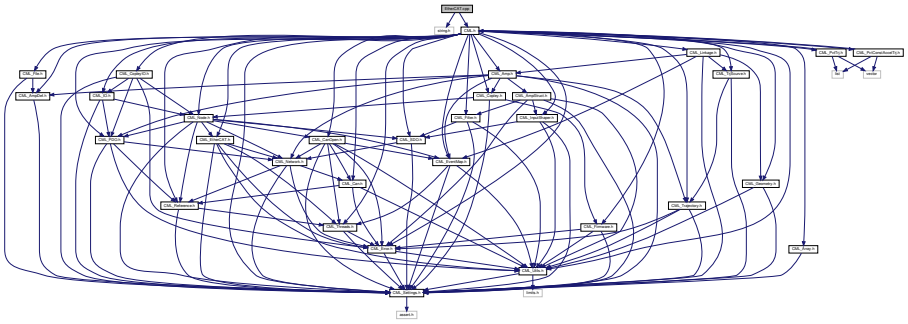
6.52.1 Detailed Description

This file handles initializing the static data objects used by the Error class.

6.53 EtherCAT.cpp File Reference

This file holds code for the top level EtherCAT class.

Include dependency graph for EtherCAT.cpp:



Macros

- #define SM_RXMBX 0

  The EtherCatNodeInfo structure holds some data required by the EtherCAT network interface which is present in every node it manages.

6.53.1 Detailed Description

This file holds code for the top level EtherCAT class.

This class manages the EtherCAT network.

6.53.2 Macro Definition Documentation

6.53.2.1 SM_RXMBX

```
#define SM_RXMBX 0
```

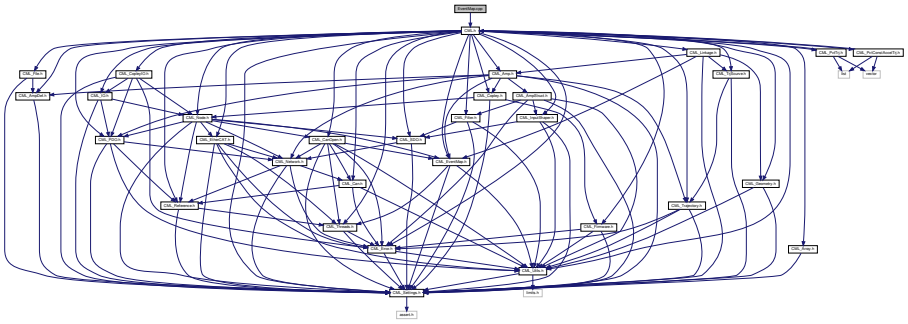
The EtherCatNodeInfo structure holds some data required by the EtherCAT network interface which is present in every node it manages.

The contents of this structure should be considered the private property of the EtherCAT class.

6.54 EventMap.cpp File Reference

This file contains the implementation of the EventMap class.

Include dependency graph for EventMap.cpp:



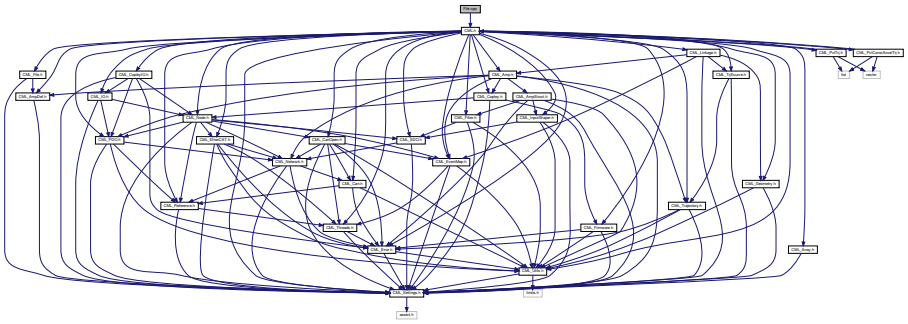
6.54.1 Detailed Description

This file contains the implementation of the EventMap class.

6.55 File.cpp File Reference

This file contains code used to parse CME-2 type files.

Include dependency graph for File.cpp:



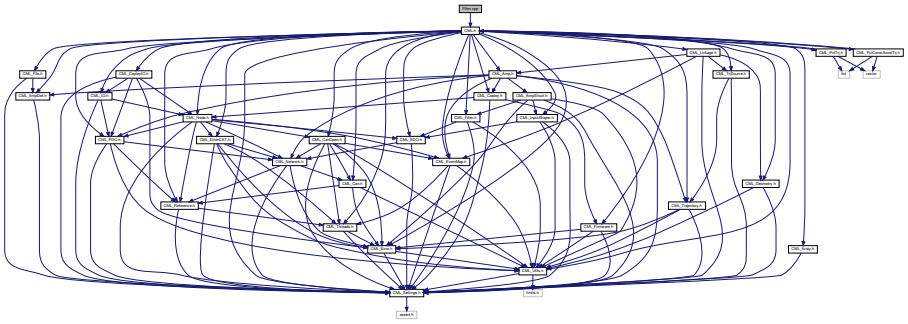
6.55.1 Detailed Description

This file contains code used to parse CME-2 type files.

6.56 Filter.cpp File Reference

Implementation of the Filter class.

Include dependency graph for Filter.cpp:

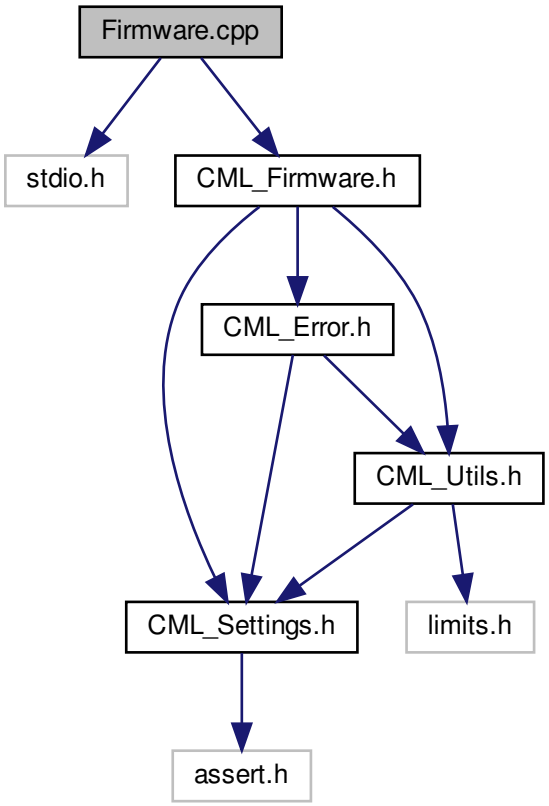


6.56.1 Detailed Description

Implementation of the Filter class.

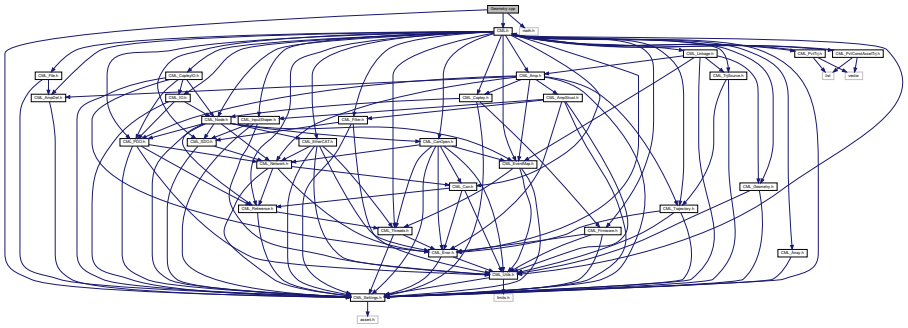
6.57 Firmware.cpp File Reference

Include dependency graph for Firmware.cpp:



6.58 Geometry.cpp File Reference

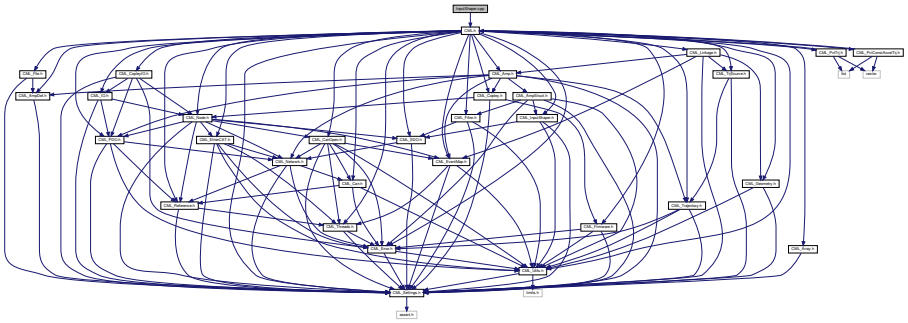
Include dependency graph for Geometry.cpp:



6.59 InputShaper.cpp File Reference

Implementation of the InputShaper class.

Include dependency graph for InputShaper.cpp:



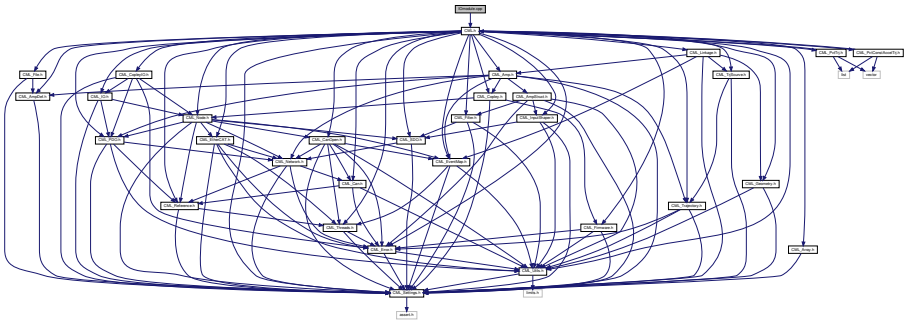
6.59.1 Detailed Description

Implementation of the InputShaper class.

6.60 IOmodule.cpp File Reference

I/O module object support.

Include dependency graph for IOmodule.cpp:



6.60.1 Detailed Description

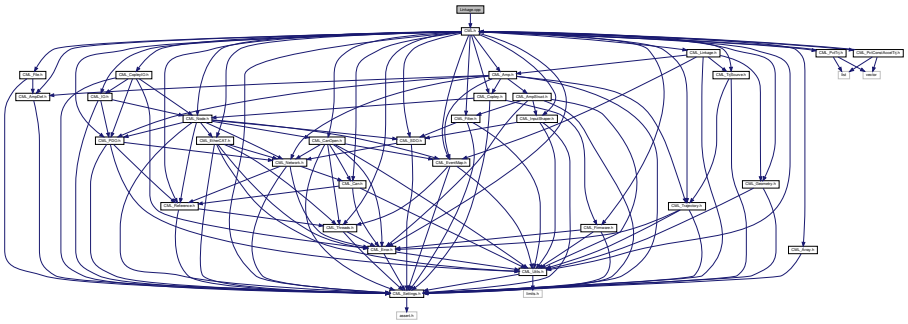
I/O module object support.

This file holds the code used to implement a standard DS401 I/O module.

6.61 Linkage.cpp File Reference

Implementation of the Linkage class.

Include dependency graph for Linkage.cpp:



Macros

- #define ERROR_EVENTS

  Update the status event map used by this linkage.

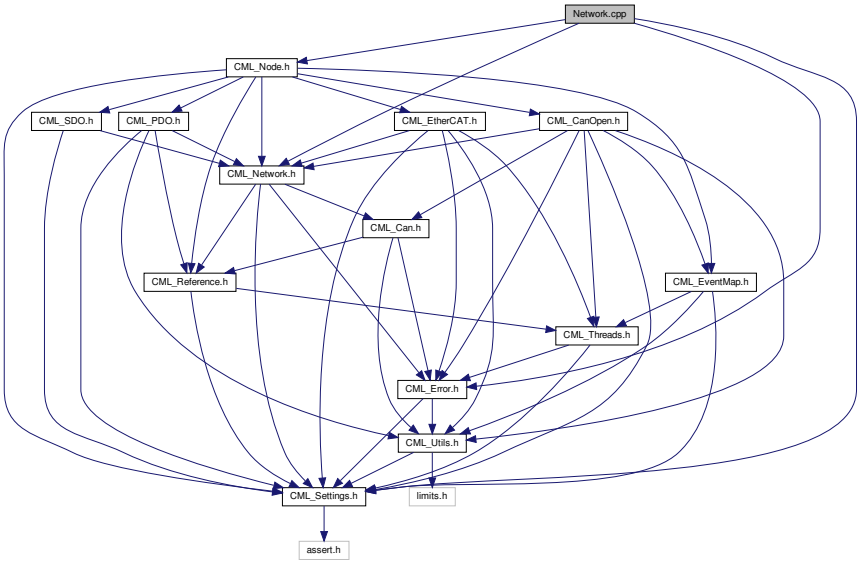
6.61.1 Detailed Description

Implementation of the Linkage class.

6.62 Network.cpp File Reference

This file holds code for the top level CANopen class.

Include dependency graph for Network.cpp:



6.62.1 Detailed Description

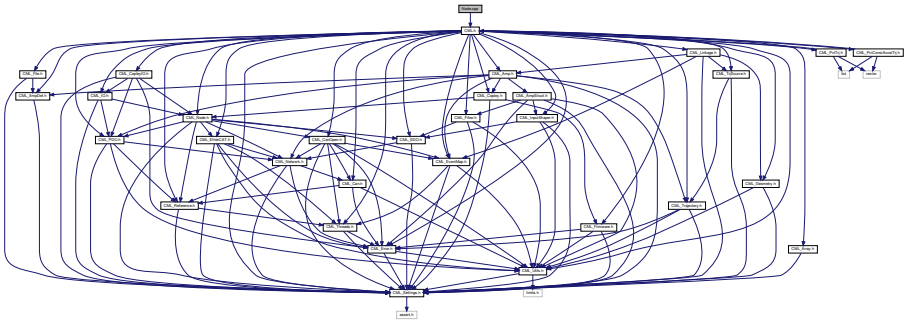
This file holds code for the top level CANopen class.

This class is used for over all control of the CANopen network.

6.63 Node.cpp File Reference

This file holds code to implement the CANopen node related objects.

Include dependency graph for Node.cpp:



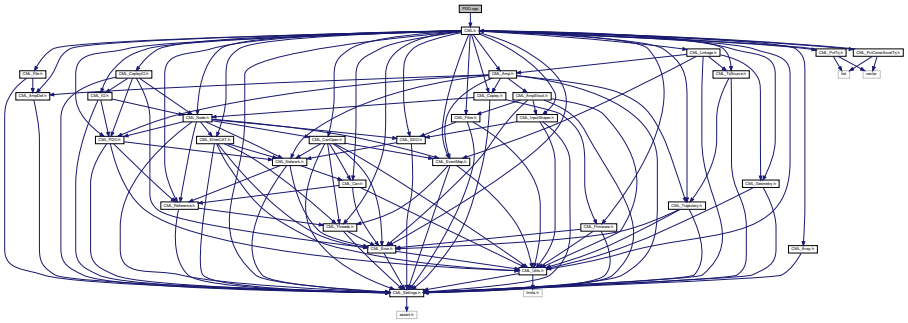
6.63.1 Detailed Description

This file holds code to implement the CANopen node related objects.

6.64 PDO.cpp File Reference

This file holds the code needed to implement CANopen Process Data Objects (PDOs).

Include dependency graph for PDO.cpp:



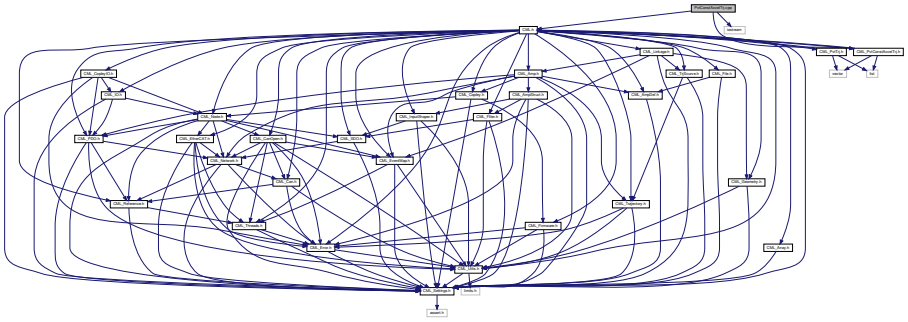
6.64.1 Detailed Description

This file holds the code needed to implement CANopen Process Data Objects (PDOs).

6.65 PvtConstAccelTrj.cpp File Reference

This file holds code to implement the PvtConstAccelTrj class.

Include dependency graph for PvtConstAccelTrj.cpp:



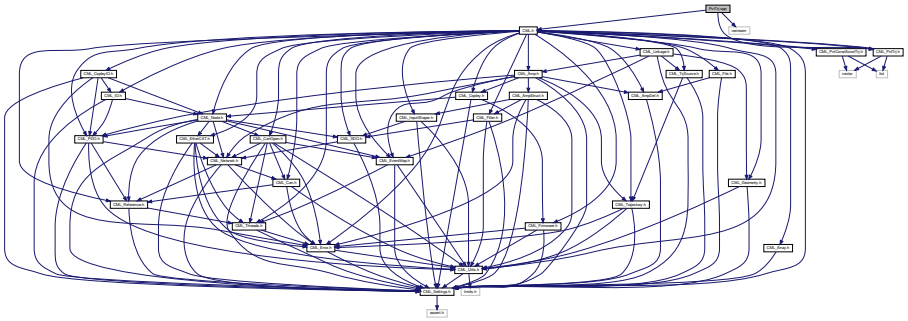
6.65.1 Detailed Description

This file holds code to implement the PvtConstAccelTrj class.

6.66 PvtTrj.cpp File Reference

This file holds code to implement the PvtTrj class.

Include dependency graph for PvtTrj.cpp:



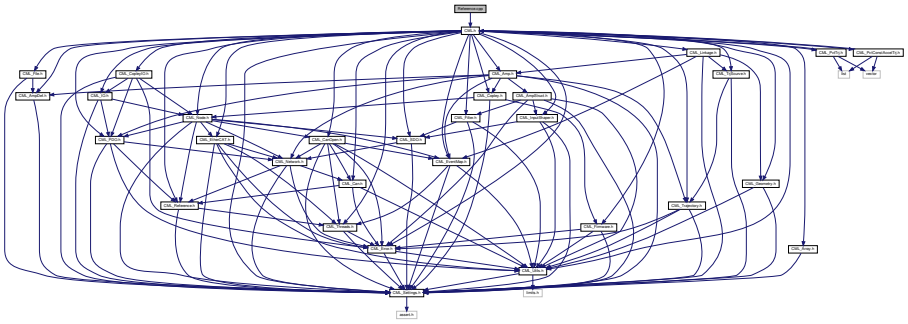
6.66.1 Detailed Description

This file holds code to implement the PvtTrj class.

6.67 Reference.cpp File Reference

This file holds the code needed to implement the CML reference counting objects.

Include dependency graph for Reference.cpp:



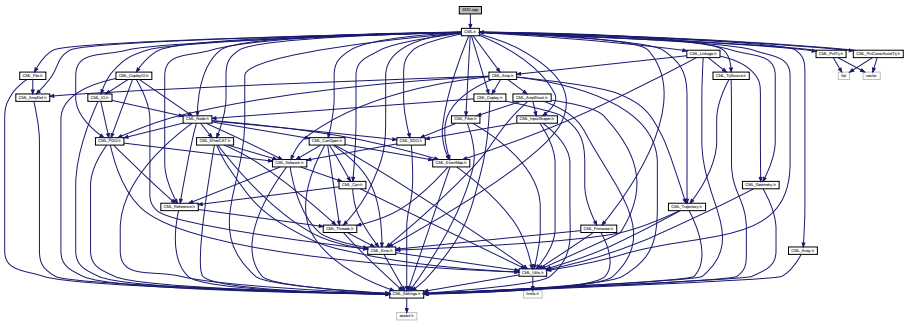
6.67.1 Detailed Description

This file holds the code needed to implement the CML reference counting objects.

6.68 SDO.cpp File Reference

This file contains the code used to implement the CANopen SDO objects.

Include dependency graph for SDO.cpp:



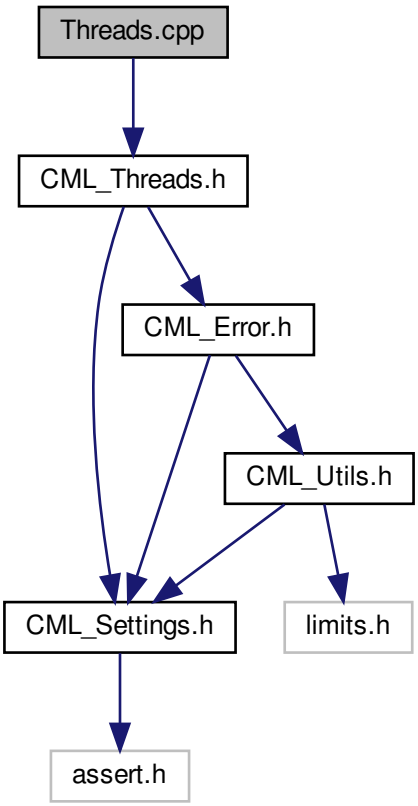
6.68.1 Detailed Description

This file contains the code used to implement the CANopen SDO objects.

6.69 Threads.cpp File Reference

This file only contains definitions for the generic thread error objects.

Include dependency graph for Threads.cpp:



6.69.1 Detailed Description

This file only contains definitions for the generic thread error objects.

The code used to implement the OS specific thread methods is located in Operating system specific files such as Threads_posix.cpp and Threads_w32.cpp.

Index