sFoundation 2.0

Namespace List

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

sFnd	

sFoundation 2.0

Classes | Typedefs

sFnd Namespace Reference

Classes

struct	**_mnErr** Error code and reason structure. More...
class	**IAttnNode** *Node Attention* Feature Interface. More...
class	**IAttnPort** Serial Port Attention Feature Interface. More...
class	**IBrakeControl** Brake Control Feature Interface. More...
class	**IGrpShutdown** Group Shutdown Feature Interface. More...
class	**IHoming** Homing Feature Interface. More...
class	**IInfo** Node Information Interface. More...
class	**IInfoEx**

	Expert Information Interface. More...
class	**ILimits** Torque and Position Limits Feature Interface. More...
class	**IMotion** Motion Control Feature Interface. More...
class	**IMotionAdv** Advanced Motion Control Interface. More...
class	**INode** Virtual Node Interface. More...
class	**INodeAdv** Advanced Node Features Interface. More...
class	**IObj** Feature Availability Interface. More...
class	**IObjWithNode** Feature Availability And Root Node Access Interface. More...
class	**IObjWithPort** Feature Availability And Port Access Interface. More...
class	**IPort** Serial Port Interface. More...
class	**IPortAdv** Advanced Port Features Interface. More...
class	**ISetup** Node Setup Features Interface. More...

class	**ISetupEx** Expert Setup Interface. More...
class	**IStatus** Node Status Features Interface. More...
class	**IStatusAdv** Advanced Status Features Interface. More...
class	**SysManager** sFoundation Top Level System Object. More...
class	**ValueAlert** Alert Type Register Node Parameter Access Object. More...
class	**ValueBase** Node Parameter Container Base Class. More...
class	**ValueDouble** This class object implements access to floating point parameters from a node. This class object implements access to floating point parameters within the node. More...
class	**ValuePowerReg** Power Status Register Node Parameter Access Object. More...
class	**ValueSigned** Signed 32-bit Integer Parameter Object. More...
class	**ValueStatus** Status Type Register Node Parameter Access Object.

More...

class **ValueString**
String Parameter Object. More...

class **ValueUnsigned**
Unsigned 32-bit Integer Parameter Object. More...

Typedefs

typedef struct **sFnd::_mnErr** **mnErr**
Error code and reason structure.
More...

Detailed Description

Teknic S-Foundation namespace.

Namespace for sFoundation class library.

Typedef Documentation

typedef struct sFnd::_mnErr sFnd::mnErr

Error code and reason structure. When the sFoundation class library encounters an error, this item will be thrown. This item can be used to identify the error, the address reporting the error, and get a brief description of the error.

Coding Example

```
try {
    // Do some library function
}
// If there is a error print out details
catch (mnErr theErr) {
    cout << theErr.TheAddr;
    cout << theErr.ErrorCode;
    cout << theErr.ErrMsg;
}
```

Examples:

Example-MultiThreaded/Axis.cpp, **Example-MultiThreaded/Example-MultiThreaded.cpp**, **Example-MultiThreaded/Supervisor.cpp**, **Example-SingleThreaded/Axis.cpp**, and **Example-SingleThreaded/Example-SingleThreaded.cpp**.

sFoundation 2.0

Here is a list of all documented namespace members with links to the namespaces they belong to:

- mnErr : **sFnd**

sFoundation 2.0

- mnErr : **sFnd**

sFoundation 2.0

Class List

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

[detail level 1 2 3]

▼ N sFnd	
C _mnErr	Error code and reason structure
C IAttnNode	*Node Attention* Feature Interface
C IAttnPort	Serial Port Attention Feature Interface
C IBrakeControl	Brake Control Feature Interface
C IGrpShutdown	Group Shutdown Feature Interface
C IHoming	Homing Feature Interface
C IInfo	Node Information Interface
C IInfoEx	Expert Information Interface
C ILimits	Torque and Position Limits Feature Interface
C IMotion	Motion Control Feature Interface
C IMotionAdv	Advanced Motion Control Interface
▼ C INode	Virtual Node Interface
C UseMutex	A Simple and Safe Mutex mechanism
C INodeAdv	Advanced Node Features Interface
C IObj	Feature Availability Interface
C IObjWithNode	Feature Availability And Root Node Access Interface
	Feature Availability And Port Access

IObjWithPort	Interface
IPort	Serial Port Interface
IPortAdv	Advanced Port Features Interface
ISetup	Node Setup Features Interface
ISetupEx	Expert Setup Interface
IStatus	Node Status Features Interface
IStatusAdv	Advanced Status Features Interface
SysManager	SFoundation Top Level System Object
ThreadLock	A safe thread locking mechanism
ValueAlert	Alert Type Register Node Parameter Access Object
ValueBase	Node Parameter Container Base Class
ValueDouble	This class object implements access to floating point parameters from a node. This class object implements access to floating point parameters within the node
ValuePowerReg	Power Status Register Node Parameter Access Object
ValueSigned	Signed 32-bit Integer Parameter Object
ValueStatus	Status Type Register Node Parameter Access Object
ValueString	String Parameter Object
ValueUnsigned	Unsigned 32-bit Integer Parameter Object
_alertReg	Universal Alert and Warning Register Type
_cpmAlertFlds	ClearPath-SC Alert and Warning Register Field Definitions

_cpmStatusRegFlds	ClearPath-SC Node Status Register Fields
_mnStatusReg	Universal Status Register Type
_ShutdownInfo	Information to setup group shutdowns at a node
commonAlertFlds	Alert bits common to all nodes

List of all members | Public Attributes

sFnd::_mnErr Struct Reference

Detailed Description

Error code and reason structure. When the sFoundation class library encounters an error, this item will be thrown. This item can be used to identify the error, the address reporting the error, and get a brief description of the error.

Coding Example

```
try {
    // Do some library function
}
// If there is a error print out details
catch (mnErr theErr) {
    cout << theErr.TheAddr;
    cout << theErr.ErrorCode;
    cout << theErr.ErrMsg;
}
```

Examples:

Example-GPIO/Example-GPIO.cpp, **Example-Homing/Example-Homing.cpp**, **Example-Motion/Example-Motion.cpp**, **Example-StatusAlerts/Example-StatusAlerts.cpp**, and **HelloWorld/HelloWorld.cpp**.

Public Attributes

multiaddr	**TheAddr**
cnErrCode	**ErrorCode**
char	**ErrorMsg** [512]

Member Data Documentation

cnErrCode sFnd::_mnErr::ErrorCode

The system Error Code number.

Examples:
Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-MultiThreaded/Axis.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

char sFnd::_mnErr::ErrorMsg[512]

A printable error string that includes the ErrorCode.

cnErrCode	Value	Description
MN_ERR_TIMEOUT,	0x80040002	Failed due to time-out while waiting for attention.
MN_ERR_DEV_ADDR,	0x80040004	Device addressing problems. • Check that you are not trying to send commands to a port that does not exist. • Only address ports [0-2].
MN_ERR_TOO_MANY,	0x80040005	There are too many nodes on the net(16 Max).
MN_ERR_RESP_FMT,	0x80040006	Response is garbled. This indicates serious com issues.

		• Ensure network power supply is stable. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_OFFLINE,	0x80040008	The node went offline. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_PARAM_RANGE,	0x80040009	Parameter is out of range.
MN_ERR_CLOSED,	0x8004000c	Attempting to use a closed port/node. • Ensure ports are opened before accessing. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_PKT_ERR,	0x80040011	Packet length does not equal buffer length. • Check for issues with the serial port.

		• Check conections, and wiring for intermittent connectivity issues.
MN_ERR_CMD_OFFLINE,	0x8004001b	Command attempt while net offline. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_ATTN_OVERRUN,	0x8004001e	Attention over-run error. This indicates one or more attentions may have been lost due to incorrect attention aetup handling. • Ensure attentions are enabled at the **IAttnPort** level.
MN_ERR_BADARG,	0x80040027	API function passed bad argument
MN_ERR_SEND_FAILED,	0x8004002a	The host serial port rejected a command. • Check that the host serial port is operating properly.
MN_ERR_RESP_TIMEOUT,	0x8004002e	Response from Node not found in time.

		• Ensure network power supply is stable. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_CANCELED,	0x80040030	Command canceled during Node restart.
MN_ERR_ADDR_RANGE,	0x80040034	Response from out of range address. A node likely went offline. • Ensure network power supply is stable. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_RESET_FAILED,	0x8004003b	Reset command failed, node isn't responding. • Ensure network power supply is stable. • Check conections, and wiring for intermittent connectivity issues.
MN_ERR_FILE_WRITE,	0x80040040	Configuration file failed to write to file

		location. • Check for correct directory. • Check for appropriate write-access for specified directory. • Check for adequate disk space.
MN_ERR_FILE_OPEN,	0x80040041	Configuration file failed to open. • Check for correct directory. • Check for appropriate read/write-access for specified directory.
MN_ERR_FILE_WRONG,	0x80040042	Configuration file is for wrong node. • Ensure the configuration file specified is for the correct Node type.
MN_ERR_FILE_BAD,	0x80040043	Configuration file corrupt
MN_ERR_FILE_ENABLED,	0x80040044	Configuration file load requires node to be disabled. • Ensure node is disabled before attempting to

		load configuration file.
MN_ERR_CMD_IN_ATTN,	0x80040045	Node command attempted from within an Attention Callback. • Ensure your registered attention callback functions do not attempt to send commands to any node.
MN_ERR_APP_NET_ONLY,	0x80040046	Feature only available on App Net
MN_ERR_24V_OFF,	0x80000047	Canceled due to lack of 24V supply. • Ensure communication power supply is stable, and turned on.
MN_ERR_CMD_ARGS,	0x80040102	Illegal or missing command args
MN_ERR_CMD_WR_FAIL,	0x80040103	Attempt to write to a read-only parameter.
MN_ERR_CMD_EEHW,	0x80040104	Non-Volatile memory failure. • Check conections, and wiring for intermittent connectivity issues.

		• If problem persists, replace motor.
MN_ERR_CMD_ACCESS_LVL,	0x80040105	Access level violation. • Make sure that the diagnostic USB channel is in moniter mode through ClearView.
MN_ERR_CMD_MV_FULL,	0x80040106	Move Buffers full. • Allow moves in buffer to complete before adding additional moves.
MN_ERR_CMD_MV_SPEC,	0x80040107	Move specification error. • Check that the commanded velocity is below the velocity limit for the node. • Ensure that the move distance is within the soft limits of the motor, and does not exxceed the numberspace (~8,000,000).
MN_ERR_CMD_MV_ESTOPPED,	0x80040108	Motion attempt in E-Stopped state.

		• Clear E-Stop state before commanding motion.
MN_ERR_CMD_MV_RANGE,	0x80040109	Motion attempt is out of range. • Ensure that the move distance is within the soft limits of the motor, and does not exxceed the numberspace (~8,000,000).
MN_ERR_CMD_MV_SHUTDOWN,	0x8004010a	Motion attempt when drive is in shutdown. • Appropriatly clear shutdown before commanding motion.
MN_ERR_CMD_MV_BLOCKED,	0x8004010c	Motion blocked non-disabling shutdown(ie: limit switches) present. • Remove cause of non-disabling shutdown. • Ensure commanded motion is in valid direction(ie: not attempting to move into an already asserted limit).

MN_ERR_NODE_IN_MOTION,	0x80040110	Drive is in motion. Some node commands cannot be called during motion(ie: Initiating homing).
MN_ERR_OFFLINE_XX,	0x800403a0-0x800403af	Node X detected offline Detected offline nodes are reported with this group. These are usually caused by a node restarting via communication power or firmware error.
MN_ERR_INIT_FAILED_PORTS_X,	0x80040600-0x80040607	Port X opened OK but no nodes were found attached to the link.
MN_ERR_PORT_FAILED_PORTS_X	0x80040700-0x80040707	Port X failed to open. • Make sure you specify a working port. • Check that the USB/Serial cable is properly inserted. • Ensure no other applications are activly using the port(ie: ClearView).
MN_ERR_BAUD_FAILED_PORTS_X	0x80040800 - 0x80040807	Port X opened OK but doesn't support the requested baud rate. • Check that the serial port supports your

		requested baud rate. • If using the SC-Hub, use a baud rate of MN_BAUD_12X, or MN_BAUD_24X.

Examples:
Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

multiaddr sFnd::_mnErr::TheAddr

Address reporting the error.

Examples:
Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

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

- C:/Git/inc/inc-pub/pubNetAPI.h

List of all members | Public Member Functions | Public Attributes

sFnd::IAttnNode Class Reference

Detailed Description

Node Attention Feature Interface.

A reference to this object allows your application to control which node status events will generate a *Network Attention* packet. These packets efficiently signal the host some noteworthy event, such as move done, has occurred. Using Attentions eliminates the overhead and latency involved with polling.

Note

This feature is only available on nodes that have the Advanced feature set.

See also

Attentions/Interupts for Event Driven Applications

cpmStatusRegFlds for a complete list of **Node** status events

Inherits **sFnd::IObjWithNode**.

Public Member Functions

mnStatusReg	**WaitForAttn** (**mnStatusReg** theAttn, int32_t timeoutMsec, bool autoClear=true) Wait for any of the indicated attention fields to assert. More...
void	**ClearAttn** (**mnStatusReg** attnClr) Clear the indicated fields from the current attention state. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Public Attributes

ValueStatus	**Mask** Attention Enabling Mask. More...
ValueAlert	**AlertMask** Selection of Alerts to include in the *Status Register* User Alert field. More...
ValueStatus	**StatusMask** Selection of Status Events to include in the *Status Register* Status Event field. More...

Member Function Documentation

void sFnd::IAttnNode::ClearAttn (mnStatusReg attnClr)

Parameters

[in] **attnClr** Set fields you wish to clear from the accumulating attentions.

This member function is used to clear unwanted past Attentions that have been received by the host.

mnStatusReg sFnd::IAttnNode::WaitForAttn (mnStatusReg theAttn, int32_t timeoutMsec, bool autoClear = `true`)

Parameters

[in] **theAttn** The set of attentions to wait on.

Note

Only the lower 32-bits of the status register generate attentions.

Parameters

[in] **timeoutMsec** The maximum time to wait for these attentions.

[in] **autoClear** If set, the bits underlying last attention register cooresponding to theAttn will be atomically cleared.

Returns

A copy of the attentions that released this function. If the fields of this register is zero, the timeout occurred.

When called, this function blocks until the desired attention occurs. The returned mnStatusReg is set with the fields of the last matching attention. This is helpful when waiting on multiple attentions to find out the actual attention signaled. The *timeoutMS* parameter allows you limit the amount of waiting time until the return. This is typically required to prevent your application from hanging up when something fails to deliver the expected event(s).

Coding Example

```
//Enable attentions on the Port (typically this
    would be enabled once ate the beginning of
    the application)
myPort.Adv.Attn.Enable(true);

// Wait for node to be ready to take commands
attnReg attnReq;
attnReq.cpm.Ready = 1;
mynode.Adv.Attn.ClearAttn(attnReq);

mynode.EnableReq(true);

// If the node is not currently ready, wait for
    it to get there
mynode.Status.RT.Refresh();
if (!mynode.Status.RT.Value().cpm.Ready){
attnReg theAttn =
    mynode.Adv.Attn.WaitForAttn(attnReq, 3000);
if (!theAttn.cpm.Ready){
mnErr eInfo;
eInfo.ErrorCode = MN_ERR_TIMEOUT;
eInfo.TheAddr = mynode.Info.Ex.Addr;
snprintf(eInfo.ErrorMsg, sizeof(eInfo.ErrorMsg),
"Error: Timed out waiting for enable\n");
```

```
throw eInfo;
}
}
```

Member Data Documentation

ValueAlert sFnd::IAttnNode::AlertMask

Setup which fields of the *Alert Register* will set the **cpmStatusRegFlds::UserAlert** field in the *Status Register*.

See also

_cpmStatusReg For details about ClearPath-SC specific status

_mnStatusReg For details about the *Universal Status Register*

ValueStatus sFnd::IAttnNode::Mask

Setup which fields of the *Status Attn/Rise Register* cause the generation of an *Attention Packet* to the host. These packets are similar to a microprocessor interrupt.

See also

_cpmStatusRegFlds For details about ClearPath-SC specific status

_mnStatusReg For details about the *Universal Status Register*

Note

This is an advanced feature.

ValueStatus sFnd::IAttnNode::StatusMask

Setup which fields of the *Status Register* will set the **cpmStatusRegFlds::StatusEvent** field in the *Status Register*.

Note

Do not include the *StatusEvent* field in this mask to avoid latching this field in the status register.

See also

_cpmStatusRegFlds For details about ClearPath-SC specific status

_mnStatusReg For details about the *Universal Status Register*

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::IAttnPort Class Reference abstract

Detailed Description

This implements the feature object required to setup attentions and register an Attention handler callback function on this port.

See also
AttentionsPage

Inherits **sFnd::IObjWithPort**.

Public Member Functions

virtual void	**Enable** (bool newState)=0 Control the posting of attentions. More...
virtual bool	**Enabled** ()=0 Return the global attention posting switch. More...
bool	**HasAttnHandler** () Detect if handler is defined. More...
void	**AttnHandler** (mnAttnCallback theNewHandler) Register an attention callback function. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

void sFnd::IAttnPort::AttnHandler (mnAttnCallback theNewHandler)

Register an attention callback function.

Parameters

[in] **theNewHandler** The address of a handler function of type mnAttnCallback;

Note

To prevent system performance issues, the implemented callback function should not run for extended periods of time. The function is also restricted from running any network based command. This includes parameter accesses and node commands. This function should signal other threads to restart themselves or other lightweight signaling mechanisms.

Coding Example

```
// Example showing a Attention Handler function
// Enable the attentions for this port
myPort.Adv.Attn.Enable(true);
// The attentions will be handled by the individual
    nodes, but register
// a handler at the port level, just for illustrative purpos
myPort.Adv.Attn.AttnHandler(AttentionDetected);

void MN_DECL AttentionDetected(const mnAttnReqReg &detected)
{
// Make a local, non-const copy for printing purposes
mnAttnReqReg myAttns = detected;
// Create a buffer to hold the attentionReg information
char attnStringBuf[512];
// Load the buffer with the string representation of the
    attention information
myAttns.AttentionReg.StateStr(attnStringBuf, 512);
// Print it out to the console
printf("  --> ATTENTION: port %d, node=%d, attn=%s\n",
detected.MultiAddr >> 4, detected.MultiAddr, attnStringBuf);
}
```

virtual void sFnd::IAttnPort::Enable (bool newState) pure virtual

Parameters

[in] **newState** If set, sFoundation will process received *Attention Requests*.

Note

This does not turn off the generation of attentions by the node. Turn off Attention generation by clearing the **sFnd::IAttnNode::Mask** for each node on the Port. If attention posting is disabled, but attentions are still generated, an error will be generated.

virtual bool sFnd::IAttnPort::Enabled ( ) pure virtual

Returns

True if attentions can be posted.

bool sFnd::IAttnPort::HasAttnHandler ( )

Returns

true if a handler is installed.

Use the **IAttnPort::AttnHandler** function to register a handler callback function.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::IBrakeControl Class Reference abstract

Detailed Description

This class implements access to the brake control outputs of a SC-Hub Board. If multiple SC-Hub boards are chained together on a single port, only the first SC-Hub board's outputs will be controllable.

The Brake Control feature can be used to automatically control a brake whenever a node is disabled or shutdown, or they can be used as general purpose outputs. This object also allows the application to override the brake to turn the brake off even when the node is disabled, allowing the axis to move freely. This class provides functions to turn the outputs on and off, as well as to check the current state of the outputs.

Sometimes a brake can take time to engage so there is a Node Level **delay to disable** parameter which can be set to give the brake time to activate before the motor disables.

Note

When using **BRAKE_AUTOCONTROL** mode, Brake_0 will activate and deactivate based on the state of the node connected to the CP0 connector on the SC-Hub Board. Similarly, Brake_1 will correspond to CP1 on the SC-Hub Board.

Coding Example

```
// Configure the SC hub's Brake_0 to be in auto control
    mode (the brake will automatically activate, and
    deactivate based on the state of node 0)
myPort.BrakeControl.BrakeSetting(0,BRAKE_AUTOCONTROL);

//Determine if the current Brake Setting for the SC-Hub
    Board's Brake_0 is BRAKE_AUTOCONTROL
if(myPort.BrakeControl.BrakeSetting(0)==BRAKE_AUTOCONTRO
    L)
{  In auto control}
else
{ Not in auto control (The brake could be overridden)}

//Determine if the Brake_0 output on the SC-Hub Board's
    is engaged
if(myPort.BrakeControl.BrakeEngaged(0))
{  Brake Engaged }
else
```

```
{  Brake Not Engaged}

// Turn on the SC hub's Brake_1 output as a general
    purpose Output.
myPort.BrakeControl.BrakeSetting(1,GPO_ON);
```

See also

_BrakeControls for the complete list of brake modes

Inherits **sFnd::IObjWithPort**.

Public Member Functions

virtual void	**BrakeSetting** (size_t brakeNum, **BrakeControls** brakeMode)=0 Setup how brake will function. More...
virtual **BrakeControls**	**BrakeSetting** (size_t brakeNum)=0 Get the brake control state. More...
virtual bool	**BrakeEngaged** (size_t brakeNum)=0 Return the current brake state. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

virtual bool sFnd::IBrakeControl::BrakeEngaged (size_t brakeNum) pure virtual

Determine the curent state of the Brake. A Brake is considered engaged when the brake is holding the axis. This would correspond to zero current flow on a GPO.

Parameters

[in] **brakeNum** Brake number to use [0..1]; 0 corresponds to Brake_0 on a SC Hub, 1 corresponds to Brake_1.

Returns

true if the brake is engaged.

virtual void sFnd::IBrakeControl::BrakeSetting (size_t brakeNum, BrakeControls brakeMode) pure virtual

This function configures the configures the brake output.

Parameters

[in] **brakeNum** Brake number to use [0..1] 0 corresponds to Brake_0 on a SC Hub, 1 corresponds to Brake_1.

[in] **brakeMode** Brake mode to use.

See also

_BrakeControls for the complete list of modes

Examples:

Example-GPIO/Example-GPIO.cpp.

virtual BrakeControls sFnd::IBrakeControl::BrakeSetting (size_t brakeNum) pure virtual

Get the brake control state.

Parameters

[in] **brakeNum** Brake number to use [0..1] 0 corresponds to Brake_0 on a SC Hub, 1 corresponds to Brake_1.

See also

_BrakeControls for the complete list of modes

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::IGrpShutdown Class Reference abstract

Detailed Description

This class implements access to the group shutdown feature which allows an electrical connection or a change in a node's status to initiate a Group Shutdown Node Stop. This Group Shutdown will automatically propagate and initiate a Node Stop on all nodes in the port.

This class can configure how each node on the port will react to a Group Shutdown Node Stop, as well as whether certain fields in the Node's mnStatusdReg will issue a group shutdown. For instance, once a node goes **not ready** , a group shutdown can be called.

Once a group shutdown is triggered the shutdown is sent to each node on the port. A group shutdown is also automatically triggered upon the loss of power (negative edge) of the global stop input on the SC-Hub board.

Coding Example

```
// The following lines of code show an example of how to create
    a Shutdown info object.

// First we create the status mask object consisting of which
    status events on the node will trigger a group shutdown.
// In this case we are using "AlertPresent", "Disabled", and
    "Input A"

mnStatusReg myShutdownStatusMask;                        //Create
    Status mask
myShutdownStatusMask.cpm.AlertPresent = 1;               //Set
    desired fields
myShutdownStatusMask.cpm.Disabled = 1;
myShutdownStatusMask.cpm.InA = 1;

// Next we create the ShutdownInfo object we will use to
    configure the Node with
ShutdownInfo myShutdownInfo;                             //Create
    ShutdownInfo Object
myShutdownInfo.enabled = true;                           // This
    sets the node to respond to Group Shutdowns
myShutdownInfo.theStopType = STOP_TYPE_ABRUPT;           //Use an
    Abrupt Nodestop type for this node in response to a group shutd
myShutdownInfo.statusMask = myShutdownStatusMask;        //
    Configure the node to issue a group shutdown automatically
    based on our status mask

//Finally we configure the Port's first Node (Node 0) with our
    shutdown object
myport.GrpShutdown.ShutdownWhen(0,myShutdownInfo);
```

See also

CPM NodeStops and E-Stops to Cancel Motion

Inherits **sFnd::IObjWithPort**.

Public Member Functions

virtual void	**ShutdownWhen** (size_t nodeIndex, const **ShutdownInfo** &theInfo)=0 Setup the group shutdown for this node. More...
virtual void	**ShutdownWhenGet** (size_t nodeIndex, **ShutdownInfo** &theInfo)=0 Get the group shutdown settings for this node. More...
virtual void	**ShutdownInitiate** ()=0 Initiate a group shutdown. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

virtual void sFnd::IGrpShutdown::ShutdownInitiate ( ) pure virtual

Trigger the Group Shutdown to occur via an API call.

virtual void sFnd::IGrpShutdown::ShutdownWhen (size_t nodeIndex, const ShutdownInfo & theInfo) pure virtual

This function is used to setup how a node will react to a group shutdown, as well as which items in the node's status register will trigger a group shutdown.

Parameters

[in]	**nodeIndex**	Node index. This corresponds to the location from the host on the network.
[in]	**theInfo**	Settings to apply to this node.

See also

_ShutdownInfo for information on how to specify the stoptype, status mask, and whether the grpShutdown feature is enabled for this node

virtual void sFnd::IGrpShutdown::ShutdownWhenGet (size_t nodeIndex, ShutdownInfo & theInfo) pure virtual

Retrieve the current _ShutdownInfo for this node, including the stop type, status mask, and whether the feature is enabled

Parameters

[in]	**nodeIndex**	Node index. This corresponds to the location from the host on the network.
[out]	**theInfo**	Settings to apply to this node.

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

- C:/Git/inc/inc-pub/pubSysCls.h

List of all members | Public Member Functions

sFnd::IHoming Class Reference abstract

Detailed Description

This class object implements access to Teknic's homing engine. This object allows a user to initiate homing and test whether homing was complete. This object cannot configure homing parameters such as homing direction/speed. Homing parameters must be set using the ClearView software, and this object only provides the ability to start homing and check whether the axis has homed.

Homing is required to use soft limits and establish the node's number space for absolute type moves.

Inherits **sFnd::IObjWithNode**.

Public Member Functions

virtual void	**Initiate** ()=0 Initiate the homing sequence. More...
virtual bool	**IsHoming** ()=0 Update status real-time and return the state of the homing active flag. More...
virtual bool	**WasHomed** ()=0 Update status real-time and return the state of the homing complete flag. More...
virtual void	**SignalComplete** ()=0 Signal the completion of manual homing. More...
virtual void	**SignalInvalid** ()=0 Signal the node that our absolute position space is invalid. More...
virtual bool	**HomingValid** ()=0 Check Homing settings to see if they are set-up properly. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

virtual bool sFnd::IHoming::HomingValid ( ) pure virtual

Returns
True if homing has been set-up

Examples:
Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

virtual void sFnd::IHoming::Initiate ( ) pure virtual

This function is called to initiate a homing sequence. ClearView should be used to setup how homing is performed.

Coding Example

```
// Initiate Homing
mynode.Motion.Homing.Initiate();
```

Examples:
Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

virtual bool sFnd::IHoming::IsHoming ( ) pure virtual

Update **status real-time register** and return the state of the homing active flag.

Returns
True if homing is occurring.

Coding Example

```
// Check to see if the Node is still homing
if(mynode.Motion.Homing.IsHoming())
```

```
{ StillHoming }
else
{ No Longer Homing  }
```

virtual void sFnd::IHoming::SignalComplete ( ) pure virtual

This function is used to signal to the node that the homing sequence has completed. Typically the homing process automatically proceeds until completion. If a hard-stop or input cannot be used to terminate the homing sequence, this function can be used to signal the completion of a homing sequence.

Calling this function will cause soft limits to activate.

Note

This function is used when the automatic homing feature cannot be used to establish the absolute number space. For example when camera processing is used to establish a position.

See also

IHoming::SignalInvalid To disable soft limits and allow another homing sequence to occur.

IMotion::AddToPosition to manually adjust the number space during manual homing

Coding Example

```
// Manually zero position, and declare the number
     space valid
mynode.Motion.AddToPosition(-mynode.Motion.MeasPosition);
mynode.Motion.Homing.SignalComplete();
```

Examples:

Example-Homing/Example-Homing.cpp.

virtual void sFnd::IHoming::SignalInvalid ( ) pure virtual

Signal the node that our absolute position space is invalid.

This is useful when using soft limits and manual homing procedure.

Note

After calling this function the soft limit feature, if enabled, is disabled.

See also

IHoming::SignalComplete to reverse the affects of this function.

IMotion::AddToPosition to manually adjust the number space during manual homing

virtual bool sFnd::IHoming::WasHomed ( ) pure virtual

Update **status real-time register** and return the state of the homing complete flag. Homing complete will only assert after a successful homing sequence or if the user calls the **SignalComplete()** function indicating a successful manual homing.

Returns

True if homing has completed.

Coding Example

```
// Check to see if the Node is still homing
if(mynode.Motion.Homing.WasHomed())
{ Homed Successfully }
else
{ Not Successfuly Homed  }
```

Examples:

Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFnd::IInfo Class Reference abstract

List of all members | Public Types | Public Member Functions | Public Attributes

Detailed Description

This class object implements access to node information.

This includes node management items such as:

- Serial Number
- Node Type
- Firmware Version
- User ID management
- Configuration file loading and saving

Inherits **sFnd::IObjWithNode**.

Public Types

enum	**_nodeTypes** { **UNKNOWN**, **CLEARPATH_SC**, **CLEARPATH_SC_ADV** } This enumeration describes the type of node this object represents. More...
typedef enum **sFnd::IInfo::_nodeTypes**	**nodeTypes** This enumeration describes the type of node this object represents.

Public Member Functions

virtual **nodeTypes**	**NodeType** ()=0 Return the enumerated type of this node. More...
virtual void	**UserData** (size_t bank, const uint8_t m_lastUserData[MN_USER_NV_SIZE])=0 Set the Current User Defined Non-volatile Data. More...
virtual std::vector< uint8_t >	**UserData** (size_t bank)=0 Get the Current User Defined Non-volatile Data. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Public Attributes

ValueUnsigned	**SerialNumber** Get serial number of this node. More...
ValueString &	**FirmwareVersion** Return a firmware version string. More...
ValueString &	**Model** Get the model string. More...
ValueUnsigned	**PositioningResolution** Get the positioning resolution. More...
ValueString &	**UserID** Manage the UserID string for this node. More...
ValueUnsigned	**UserRAM** Manage User Volatile Data. More...
IInfoEx &	**Ex** Expert Information Access. More...

Member Enumeration Documentation

enum sFnd::IInfo::_nodeTypes

Enumerator	
UNKNOWN	The type cannot be determined.
CLEARPATH_SC	The node is a ClearPath-SC.
CLEARPATH_SC_ADV	The node is an Advanced ClearPath-SC.

Member Function Documentation

virtual nodeTypes sFnd::IInfo::NodeType ( ) pure virtual

This function is used to determine the type and base capabilities of this node.

Coding Example

```
if (myNode.Info.NodeType() == IInfo::CLEARPATH_SC) {
    printf( "My Node is a ClearPath-SC\n");
    return(2);
}
if(myNode.Info.NodeType() != IInfo::CLEARPATH_SC_ADV){
    printf( "My Node is an Advanced ClearPath-SC\n");
    return(2);
}
```

Examples:
Example-Homing/Example-Homing.cpp, and Example-Motion/Example-Motion.cpp.

virtual void sFnd::IInfo::UserData (size_t bank, const uint8_t m_lastUserData[MN_USER_NV_SIZE]) pure virtual

Parameters

[in]	**bank**	which bank of data to write to [0-3].
[in]	**m_lastUserData**	The data to write into the Node. MN_USER_NV_SIZE = 13 for ClearPath SC Nodes.

This feature allows the storage of calibration or other data at a node location. This information will be retained without power and survives node restarts, and configuration file changes. Up to 4 banks of 13 bytes are available

Note
Restoring the node to Factory Settings will clear this data.

Coding Example

```
// Data to write
uint8_t newNV[MN_USER_NV_SIZE] = {
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
    0x99, 0xaa, 0xbb, 0xcc, 0xdd
};
```

```
// Write new data to bank 0
myNode.Info.UserData(0, newNV);
```

virtual std::vector<uint8_t> sFnd::IInfo::UserData (size_t bank) pure virtual

Parameters

[in] **bank** which bank of data to access [0-3].

This function allows access to the non-volatile storage areas.

Note

Restoring the node to Factory Settings will clear this data.

Coding Example

```
// Get non-volatile data from node bank 0.
vector<uint8_t> myUserNV = myNode.Info.UserData(0);
```

Member Data Documentation

IInfoEx& sFnd::IInfo::Ex

This object allow access to expert information.

These include:

- The network address
- The node index for this node
- Generic Parameter maintenance

Examples:
Example-MultiThreaded/Axis.h, Example-SingleThreaded/Axis.h, and Example-StatusAlerts/Example-StatusAlerts.cpp.

ValueString& sFnd::IInfo::FirmwareVersion

The string is in the format of "x.y.z checksum" for example, "1.0.2 E113".

Coding Example

```
printf( "FW Ver=%s\n", myNode.Info.FirmwareVersion.Value());
// prints: 1.0.2 E113
```

Examples:
Example-Homing/Example-Homing.cpp, and Example-Motion/Example-Motion.cpp.

ValueString& sFnd::IInfo::Model

This reference allows access to the model string for this node

Coding Example

```
printf( "Model=%s\n", myNode.Info.Model.Value());
// prints: CPM-SCSK-3411P-ELSA-1-0-D
```

Examples:
Example-Homing/Example-Homing.cpp, and Example-Motion/Example-Motion.cpp.

ValueUnsigned sFnd::IInfo::PositioningResolution

This function allows access to the node's positioning resolution. The returned value is the number of position counts per revolution.

Coding Example

```
//Get the positioning resolution of the Node
uint32_t myPosnResolution = myNode.Info.PositioningResolution;
```

Examples:
Example-MultiThreaded/Axis.h.

ValueUnsigned sFnd::IInfo::SerialNumber

Access the node serial number via the Value function.

Coding Example

```
printf( "SN=%u\n", uint32_t(myNode.Info.SerialNumber));
```

Examples:
Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

ValueString& sFnd::IInfo::UserID

Manage the UserID string for this node.

Note
This parameter is saved to and overwritten by **loading configuration files**, or resetting to factory defaults

Coding Example

```
// Set User ID via the Value function
myNode.Info.UserID.Value("My Node");
// Set User ID via the operator= override
myNode.Info.UserID = "My Node";
// Get User ID via the Value function
cout << myNode.Info.UserID.Value() << endl;
// Get User ID via operator const char*()
const char *myParam = myNode.Info.UserID;
```

Examples:
Example-Homing/Example-Homing.cpp, **Example-Motion/Example-Motion.cpp**, and **Example-MultiThreaded/Axis.h**.

ValueUnsigned sFnd::IInfo::UserRAM

This feature is useful for your application to stash information in a node in case the host application crashes.

Note

This value is set to zero up power-up or restarting.

Coding Example

```
// Set the volatile value
myNode.Info.UserRAM = 0x1234
// Get the volatile value
cout << myNode.Info.UserRAM.Value() << endl;
// Alternate access way
uint16_t myVolatile = myNode.Info.UserRAM;
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFoundation 2.0

List of all members | Public Member Functions

sFnd::IInfoEx Class Reference abstract

Detailed Description

This class object implements access to expert node information.

Inherits **sFnd::IObjWithNode**.

Public Member Functions

multiaddr	**Addr** () Get our internal network address. More...
nodeaddr	**NodeIndex** () Get this node's network location. More...
virtual double	**Parameter** (nodeparam index)=0 Get a numeric parameter's current value. More...
virtual void	**Parameter** (nodeparam index, double newValue)=0 Set a numeric parameter's new value. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

multiaddr sFnd::IInfoEx::Addr ( )

Returns
The network address for this node.

This information allows the retrieval of the internal network address.

Examples:
Example-MultiThreaded/Axis.h, Example-SingleThreaded/Axis.h, and Example-StatusAlerts/Example-StatusAlerts.cpp.

nodeaddr sFnd::IInfoEx::NodeIndex ( )

Returns
The node index for this port.

This is the zero based index of the location of the node. The first node from the host starts the count.

virtual double sFnd::IInfoEx::Parameter (nodeparam index) pure virtual

Parameters
[in] **index** The parameter number for a numeric parameter.

Returns

Numeric value. The scaling and units depend on the requested parameter.

Note

This function should only be used under direction of Teknic support staff.

---

virtual void sFnd::IInfoEx::Parameter (nodeparam index, double newValue) pure virtual

Parameters

[in]	**index**	The parameter number for a numeric parameter.
[in]	**newValue**	The new value to write into this parameter.

Note

This function should only be used under direction of support staff.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Attributes

sFnd::ILimits Class Reference

Detailed Description

This class object implements the settings for torque limits and positional limits.

Inherits **sFnd::IObjWithNode**.

Public Attributes

ValueDouble	**TrqGlobal** Access the node's global torque limit. More...
ValueUnsigned	**PosnTrackingLimit** Access the tracking limit shutdown point. More...

Additional Inherited Members

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Data Documentation

ValueUnsigned sFnd::ILimits::PosnTrackingLimit

Access the tracking limit shutdown point. This parameter is a safety limit should not be changed by the application during operation except is rare application specific circumstances.

Tracking error at or beyond this point will cause a safety shutdown to occur.

ValueDouble sFnd::ILimits::TrqGlobal

Access the node's global torque limit in the current user **torque units.**. Typically this value is set through the ClearView application and is not changed during the application.

Coding Example

```
// Set the torque Limit to 100%
myNode.TrqUnit(myNode.PCT_MAX);
myNode.Limits.TrqGlobal = 100;
// Print out the torque limit
cout << myNode.Limits.TrqGlobal.Value() << endl;
// Alternate access way
double myTrqLimit = myNode.Limits.TrqGlobal;
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

Generated by doxygen 1.8.11

List of all members | Public Member Functions | Public Attributes

sFnd::IMotion Class Reference abstract

Detailed Description

This class object allows setup, initiation and cancellation of motion.

Inherits **sFnd::IObjWithNode**.

Public Member Functions

virtual double	**MovePosnDurationMsec** (int32_t target, bool targetIsAbsolute=false) Calculate the duration of a positional move. More...
virtual double	**MoveVelDurationMsec** (double target) Calculate the duration of a velocity move. More...
virtual size_t	**MovePosnStart** (int32_t target, bool targetIsAbsolute=false, bool addPostMoveDwell=false)=0 Initiate a positional move. More...
virtual size_t	**MoveVelStart** (double target)=0 Initiate a velocity move. More...
virtual bool	**MoveWentDone** ()=0 Update the status rise register and test and clear the move done field. More...
virtual bool	**MoveIsDone** ()=0 Update the real-time status register and test for move done. More...
virtual bool	**VelocityReachedTarget** ()=0 Update the status rise register and test and clear the At Target Velocity field. More...
virtual bool	**VelocityAtTarget** ()=0 Update the real-time status register and test for At Target Velocity. More...
virtual bool	**WentNotReady** ()=0 Update the status rise register and test and clear the node not ready field. More...
virtual bool	**IsReady** ()=0 Update the real-time status register and test for Node ready. More...
virtual void	**NodeStop** (nodeStopCodes howToStop)=0

Initiate a simple node stop at this node. More...

virtual void **GroupNodeStop** (nodeStopCodes howToStop)=0
Initiate a node stop on all nodes on this port. More...

virtual void **NodeStopClear** ()=0
Clear all latching Node Stop modifiers. More...

virtual void **AddToPosition** (double adjAmount)=0
Adjust number space. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode & **Node** ()
Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool **Supported** ()
Query if this feature is supported in this node. More...

Public Attributes

ValueDouble **AccLimit**
Current Node Acceleration Limit. More...

ValueDouble **VelLimit**
Current Node Velocity Limit. More...

ValueUnsigned **DwellMs**
Post-motion dwell time in milliseconds. More...

ValueDouble **PosnMeasured**
Access the current measured position. More...

ValueDouble **PosnCommanded**
Access the current commanded position. More...

ValueDouble **PosnTracking**
Access the current position tracking error. More...

ValueDouble **VelMeasured**
Access the current measured velocity. More...

ValueDouble **VelCommanded**

	Access the current commanded velocity. More...
ValueDouble	**TrqMeasured** Access the current measured torque. More...
ValueDouble	**TrqCommanded** Access the current commanded torque. More...
IMotionAdv &	**Adv** Access advanced features if available on the node's model.
IHoming &	**Homing** Access to the homing feature.

Member Function Documentation

virtual void sFnd::IMotion::AddToPosition (double adjAmount) pure virtual

Parameters

[in] **adjAmount** The amount to adjust the number space.

This function is used mostly with a manual homing process, or anytime an application need to adjust the number space. It allows the **measured** and **commanded** position to be shifted by the specified amount.

Coding Example

```
// SAMPLE: Zeroing the number space around the current
    Measured Position
myNode.Motion.AddToPosition(-myNode.Motion.PosnMeasured);
```

Note

If using Soft Position limits, this command will reset the motor's **_cpmStatusRegFlds::WasHomed**Was Homed"" status causing the soft limit settings to be ignored. To re-enable soft limits see the **IHoming::SignalComplete** function.

Examples:

Example-Homing/Example-Homing.cpp.

virtual void sFnd::IMotion::GroupNodeStop (nodeStopCodes howToStop) pure virtual

Parameters

[in] **howToStop** One of the commonly used *Node Stop codes.*

This is the simplest way to broadcast a group node stop. This issues a stop of the specified type to all nodes on the initiating node's port.

See also

sFnd::IMotion::NodeStopClear To clear latching attributes

virtual bool sFnd::IMotion::IsReady ( ) pure virtual

Refresh the **real-time status register** and test for **Node Ready**. This allows for a simple script-like interface.

Returns

true if the Node is **ready** for motion command.

Coding Example

```
// SAMPLE: This checks if the Node is ready for motion commands
if(mynode.Motion.IsReady())
{
    printf("the Node is ready");
}
else
{
    printf("the Node is not ready");
}
```

Note

For performance reasons it may be necessary to use the **Status reference** and manually perform a Refresh in a periodic place if your application and check the results via the Status object.

Examples:

Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

virtual bool sFnd::IMotion::MoveIsDone ( ) pure virtual

Refresh the **real-time status register** and test for **Move Done**. This allows for a simple script-like interface.

Returns

true if **Move Done** is true.

Coding Example

```
// SAMPLE: This checks if the Node is move done.
if(mynode.Motion.MoveIsDone())
{
printf("the Node is move done");
}
else
{
printf("the Node is not move done");
```

```
}
```

Note

For performance reasons it may be superior to use the **real-time status register** reference and perform a Refresh in a periodic place in your application and check the results via the **Status object**.

Examples:

Example-Motion/Example-Motion.cpp.

virtual double sFnd::IMotion::MovePosnDurationMsec (int32_t target, bool targetIsAbsolute = false) virtual

Parameters

[in] **target**	Target position. This is either the absolute position or relative to the current position depending on the *targetIsAbsolute* argument.
[in] **targetIsAbsolute**	The target is an absolute destination.

Using the currently specified kinematic constraints, return the expected duration (in milliseconds) of the move from when it starts to when the move command completes.

This function is useful for setting up timeouts or helping to schedule other events in your application.

Coding Example

```
// SAMPLE: Using MovePosnDurationMsec to set up a reasonable
    timeout for a move to complete

//define a timeout in case the node is unable to complete
    the move
double timeout =  mynode.Motion.MoveDurationMsec(1000) +
    100;
double StartTime = myMgr.TimeStampMsec();

//This will loop checking to see if the node went Move done
    with the defined timeout
while (!mynode.Motion.MoveWentDone() &&
    myMgr.TimeStampMsec() < StartTime+timeout)
{//Wait for move to complete
}
```

Examples:

Example-Motion/Example-Motion.cpp.

virtual size_t sFnd::IMotion::MovePosnStart (int32_t target, bool targetIsAbsolute = false, bool addPostMoveDwell = false) pure virtual

Parameters

[in]	**target**	Target position. This is either the absolute position of relative to the current position depending on the *targetIsAbsolute* argument.
[in]	**targetIsAbsolute**	the target is an absolute destination.
[in]	**addPostMoveDwell**	delay the next move being issued by DwellMs after this move profile completes.

Returns

Number of additional moves the node will accept.

The move is kinematically limited by the current **acceleration**, and **velocity** limits.

Move requests are buffered and requests initiated during an active move will execute immediately after the active move completes. The return value will tell you the number of additional moves that may be queued(up to 16) when this request is processed. You can also use the *Status Register's* **_cpmStatusRegFlds::MoveBufAvail** field, which will be deasserted when the buffer is full.

Examples:

Example-Motion/Example-Motion.cpp.

virtual double sFnd::IMotion::MoveVelDurationMsec (double target) virtual

Parameters

[in] **target** Target velocity.

Using the currently specified kinematic constraints, return the expected duration (in milliseconds) of the move's acceleration to the target velocity.

This function is useful for setting up timeouts or helping to schedule other events in your application.

Coding Example

```
// SAMPLE: Using MoveVelDurationMsec to set up a reasonable
     timeout for a velocity move to complete

//define a timeout in case the node is unable to complete
     the move
double timeout = mynode.Motion.MoveVelDurationMsec() + 100;
double StartTime = myMgr.TimeStampMsec();

//This will loop checking to see if the node went Move done
     with the defined timeout
while (!mynode.Motion.VelocityReachedTarget() &&
     myMgr.TimeStampMsec() < StartTime+timeout)
{//Wait for move to complete
}
```

virtual size_t sFnd::IMotion::MoveVelStart (double target) pure virtual

Parameters

[in] **target** Target velocity.

Returns

Number of additional moves the node will accept.

The move is kinematically limited by the current **acceleration limit**.

Move requests are buffered and requests initiated during a active move will execute immediately after the active move completes. The return value will tell you the number of additional moves that may be queued(up to 16) when this request is processed. You can also use the *Status Register's* **_cpmStatusRegFlds::MoveBufAvail** field, which will be deasserted when the buffer is full.

virtual bool sFnd::IMotion::MoveWentDone ( ) pure virtual

Update the **status rise register** and test and clear the node **Move Done** field.

Returns

true if **Move Done** went true since last call.

Coding Example

```
// SAMPLE: This checks if the Node has reached move done
     since the last time the application checked.
if(mynode.Motion.VelocityReachedTarget())
{
```

```
printf("the Node reached move done");
}
else
{
printf("the Node has not reached move done");
}
```

Note

For performance reasons it may be superior to use the **Status reference** and perform a Refresh in a periodic place in your application and check the results via the **Status object**.

Examples:

Example-Motion/Example-Motion.cpp.

virtual void sFnd::IMotion::NodeStop (nodeStopCodes howToStop) pure virtual

Parameters

[in] **howToStop** One of the commonly used *Node Stop codes.*

This is the simplest way to command a nodestop.

Coding Example

```
// SAMPLE: This issues an abrupt nodestop on this node
mynode.Motion.NodeStop(STOP_TYPE_ABRUPT);

// SAMPLE: This issues a nodestop to ramp to zero velocity
    using the current decel parameter's on this node
mynode.Motion.NodeStop(STOP_TYPE_RAMP_WITH_DECEL);
```

See also

Node Stop Page for a complete list of stop types

sFnd::IMotion::NodeStopClear To clear latching attributes

virtual void sFnd::IMotion::NodeStopClear ( ) pure virtual

This function will clear the MotionLock, E-Stop, Controlled, Disabled, etc. latching conditions from the node. This will allow normal operations to continue, unless the motor is shutdown state.

Coding Example

```
// SAMPLE: Clear any Current NodeStops
myNode.Motion.NodeStopClear();
```

See also

sFnd::IMotion::NodeStop(nodeStopCodes) For initiating **Node** Stops

Examples:

Example-Homing/Example-Homing.cpp, **Example-Motion/Example-Motion.cpp**, and **Example-StatusAlerts/Example-StatusAlerts.cpp**.

virtual bool sFnd::IMotion::VelocityAtTarget ( ) pure virtual

Refresh the **real-time status register** and test for **At Target Velocity**. This allows for a simple script-like interface.

Returns

true if **At Target Velocity** is true.

Coding Example

```
// SAMPLE: This checks if the Node is at the commanded velocity
if(mynode.Motion.VelocityAtTarget())
{
printf("the Node is at velocity");
}
else
{
printf("the Node is not at velocity");
}
```

Note

For performance reasons it may be necessary to use the **Status reference** and manually perform a Refresh in a periodic place if your application and check the results.

virtual bool sFnd::IMotion::VelocityReachedTarget ( ) pure virtual

Update the **status rise register** and test and clear the node **At Target Velocity** field.

Returns

true if *At Target Velocity* went true since last call.

Coding Example

```
// SAMPLE: This checks if the Node has reached the target
    velocity since the last time the application checked.
```

```
if(mynode.Motion.VelocityReachedTarget())
{
printf("the Node reached target velocity");
}
else
{
printf("the Node has not reached the target velocity");
}
```

Note

For performance reasons it may be superior to use the **status rise register** reference and perform a Refresh in a periodic place in your application and check the results via the **Status object**.

virtual bool sFnd::IMotion::WentNotReady ( ) pure virtual

Update the **status rise register** and test and clear the node **Not Ready** field.

Coding Example

```
// SAMPLE: This checks if the Node has gone not ready since
    the last time the application checked.
if(mynode.Motion.WentNotReady())
{
printf("the Node went not ready");
}
else
{
printf("the Node has been and still is ready");
}
```

Note

For performance reasons it may be superior to use the **status rise register** reference and perform a Refresh in a periodic place in your application and check the results via the **Status object**.

Member Data Documentation

ValueDouble sFnd::IMotion::AccLimit

Access the acceleration limit value for the next issued move. The value is specified using this node's user acceleration units. Changed values will apply to the next move. The default units are RPM/sec

Coding Example

```
// SAMPLE: Getting and setting the Acceleration Limit
myNode.Motion.AccLimit = 10000.;                // Sets
double accLim = myNode.Motion.AccLimit; // Gets
```

See also

INode::AccUnit(enum _accUnits newUnits) Changing Acceleration Units

Examples:

Example-Motion/Example-Motion.cpp.

ValueUnsigned sFnd::IMotion::DwellMs

Access to the post move dwell time used in positional moves with a after-move dwell.

Coding Example

```
// SAMPLE: Getting and setting the Post-motion dwell time.
myNode.Motion.DwellMs = 100.;               // Sets
double myDwell = myNode.Motion.DwellMs; // Gets
```

See also

MovePosnStart() for more information on how to command a positional move with dwell

ValueDouble sFnd::IMotion::PosnCommanded

Access the current commanded position.

Coding Example

```
// SAMPLE: Reading the current commanded position.
double myPosnCmd = myNode.Motion.PosnCommanded;
```

Note

The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "R" 6400cnts/rev for "E"). This means that for step and direction modes, the position may be scaled differently from the step and direction input based on the ratio of the input resolution set up through ClearView, and the motor's native positioning resolution. For example, if you are using a "R" ClearPath-SC motor with a native positioning resolution of 800 cnts/rev, and you set the Input resolution to 200 steps/rev to drop in for a 200 cnt stepper motor, each step sent will change the position 4 cnts.

Examples:

Example-SingleThreaded/Axis.h.

ValueDouble sFnd::IMotion::PosnMeasured

Access the current measured position.

Coding Example

```
// SAMPLE: Reading the current measured position.
double myPosn = myNode.Motion.PosnMeasured;
```

Note

The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "Regular" 6400cnts/rev for "Enhanced"). This means that for step+direction modes the position may be scaled differently from the step and direction input based on the ratio of the input resolution set up through ClearView, and the motor's native positioning resolution. For example, if you are using a "Regular" ClearPath-SC motor with a native positioning resolution of 800 cnts/rev, and you set the Input resolution to 200 steps/rev to drop in for a 200 cnt stepper motor, each step sent will change the position 4 cnts.

Examples:

Example-Homing/Example-Homing.cpp, **Example-Motion/Example-Motion.cpp**, and **Example-MultiThreaded/Axis.h**.

ValueDouble sFnd::IMotion::PosnTracking

Access the current position tracking error. Tracking error is defined as commanded position - measured position.

Coding Example

```
// SAMPLE: Reading the current position tracking error.
double myPosnErr = myNode.Motion.PosnTracking;
```

Note

The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "Regular" 6400cnts/rev for "Enhanced"). This means that for step+direction modes the position may be scaled differently from the step and direction input based on the ratio of the input resolution set up through ClearView, and the motor's native positioning resolution. For example, if you are using a "Regular" ClearPath-SC motor with a native positioning resolution of 800 cnts/rev, and you set the Input resolution to 200 steps/rev to drop in for a 200 cnt stepper motor, each step sent will change the position 4 cnts.

ValueDouble sFnd::IMotion::TrqCommanded

Access the current commanded torque. The torque is scaled to the current node's torque unit. The default units are percentage of maximum.

Coding Example

```
// SAMPLE: Reading the current commanded Torque.
double myTrqCmd = myNode.Motion.TrqCommanded;
```

See also

INode::TrqUnit(_trqUnits)

ValueDouble sFnd::IMotion::TrqMeasured

Access the current measured torque. The torque is scaled to the current node's torque unit. The default units are percentage of maximum.

Coding Example

```
// SAMPLE: Reading the current measured torque.
double myTrq = myNode.Motion.TrqMeasured;
```

See also

INode::TrqUnit(_trqUnits)

ValueDouble sFnd::IMotion::VelCommanded

Access the current commanded velocity. The velocity is scaled to the current node's velocity unit. The default units are RPM.

Coding Example

```
// SAMPLE: Reading the current commanded velocity.
double myVelCmd = myNode.Motion.VelCommanded;
```

See also

INode::VelUnit(_velUnits) To set velocity units.

ValueDouble sFnd::IMotion::VelLimit

Access the current velocity limit value specified in the current velocity units. Changed values will apply to the next issued move. The default units are RPM.

Coding Example

```
// SAMPLE: Getting and setting the Velocity Limit
myNode.Motion.VelLimit = 1000.;              // Sets
double velLim = myNode.Motion.VelLimit; // Gets
```

See also

INode::VelUnit(enum _velUnits newUnits)

Examples:

Example-Motion/Example-Motion.cpp.

ValueDouble sFnd::IMotion::VelMeasured

Access the current measured velocity. The velocity is scaled to the current node's velocity unit. The default units are RPM.

Coding Example

```
// SAMPLE: Reading the current measured velocity.
double myVel = myNode.Motion.VelMeasured;
```

See also

INode::VelUnit(_velUnits) To set velocity units.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions | Public Attributes

sFnd::IMotionAdv Class Reference abstract

Detailed Description

This class object allows setup and initiation of advanced motion features only available in advance models of Clearpath SC. These features include:

- **Head, and Tail Moves**
- **Asymmetric moves**
- **Triggered Moves**
- **A after start**/ **B after end events**

See also
Advanced Move Types for more information.

Inherits **sFnd::IObjWithNode**.

Public Member Functions

double	**MovePosnDurationMsec** (int32_t targetPosn, bool targetIsAbsolute=false) Calculate the duration of a positional move. More...
virtual double	**MovePosnHeadTailDurationMsec** (int32_t targetPosn, bool targetIsAbsolute=false, bool hasHead=true, bool hasTail=true) Returns the duration of a head-tail positional move. More...
virtual double	**MovePosnAsymDurationMsec** (int32_t targetPosn, bool targetIsAbsolute=false) Returns the duration of an asymmetric positional move. More...
double	**MoveVelDurationMsec** (double targetCountsPerSec) Returns the duration of a velocity move. More...
virtual size_t	**MovePosnStart** (int32_t targetPosn, bool targetIsAbsolute=false, bool isTriggered=false, bool hasDwell=false)=0 Initiate a trapezoid/triangle positional move. More...
virtual size_t	**MovePosnHeadTailStart** (int32_t targetPosn, bool targetIsAbsolute=false, bool isTriggered=false, bool hasHead=true, bool hasTail=true, bool hasDwell=false)=0 Initiate a head-tail positional move. More...
virtual size_t	**MovePosnAsymStart** (int32_t targetPosn, bool targetIsAbsolute=false, bool isTriggered=false, bool hasDwell=false)=0 Initiate a asymmetric positional move. More...
virtual size_t	**MoveVelStart** (double targetPosn, bool isTriggered=false)=0 Initiate a velocity move. More...
virtual void	**TriggerMove** ()=0 Trigger a waiting move. More...
virtual void	**TriggerMovesInMyGroup** ()=0 Trigger a group of waiting moves. More...
virtual size_t	**TriggerGroup** ()=0

Get our current trigger group setting. More...

virtual void **TriggerGroup** (size_t groupNumber)=0
Set our trigger group. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode & **Node** ()
Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool **Supported** ()
Query if this feature is supported in this node. More...

Public Attributes

ValueDouble **DecelLimit**
Access the deceleration limit for the next issued Asymmetric move. More...

ValueDouble **HeadTailVelLimit**
Access the velocity limit for the head-tail sections of the next issued head-tail move. More...

ValueUnsigned **HeadDistance**
Access the head distance for the next issued head-tail move. More...

ValueUnsigned **TailDistance**
Access the tail distance for the next issued head-tail move. More...

ValueUnsigned **AfterStartDistance**
Access the *A after Start* event distance. More...

ValueUnsigned **BeforeEndDistance**
Access the *B before End* event distance. More...

Member Function Documentation

virtual double sFnd::IMotionAdv::MovePosnAsymDurationMsec (**int32_t targetPosn,** **bool targetIsAbsolute =** `false`) virtual

Parameters

`[in]` **targetPosn** Target position. This is either the absolute position of relative to the current position depending on the *theType* argument. The triggered type move will prevent execution of the motion until a trigger event is detected at the node.

`[in]` **targetIsAbsolute** the target is an absolute destination.

Returns

motion duration (milliseconds)

Using the currently specified kinematic constraints, return the expected duration (in milliseconds) of the move from when it starts to when the move command completes.

This function is useful for setting up timeouts or helping to schedule other events in your application.

Coding Example

```
// SAMPLE: Using MovePosnAsymDurationMsec to set up a reasonable
    timeout for an asymmetric move to complete

//define a timeout in case the node is unable to complete the move
double timeout =  mynode.Motion.Adv.MovePosnAsymDurationMsec(1000) +
    100;
double StartTime = myMgr.TimeStampMsec();

//This will loop checking to see if the node went Move done with the
    defined timeout
while (!mynode.Motion.MoveWentDone() && myMgr.TimeStampMsec() <
    StartTime+timeout)
{//Wait for move to complete
}
```

virtual size_t sFnd::IMotionAdv::MovePosnAsymStart (**int32_t targetPosn,** **bool targetIsAbsolute =** `false`**,** **bool isTriggered =** `false`**,** **bool hasDwell =** `false`) pure virtual

Parameters

`[in]` **targetPosn** Target position. This is either the absolute position of relative to the

current position depending on the *theType* argument. The triggered type move will prevent execution of the motion until a trigger event is detected at the node.

[in] **targetIsAbsolute** the target is an absolute destination.

[in] **isTriggered** The move awaits a trigger event to start the motion if true.

[in] **hasDwell** The move will wait the dwell parameter amount of time before starting the next move.

Returns

Number of additional moves the node will accept.

The move is kinematically limited by the current **acceleration**, **deceleration**, and **velocity**.

Move requests are buffered and requests initiated during an active move will execute immediately after the active move completes. Triggered moves will wait for a trigger event before executing. The return value will tell you the number of additional moves that may be queued(up to 16) when this request is processed. You can also use the *Status Register's* **_cpmStatusRegFlds::MoveBufAvail** field, which will be deasserted when the buffer is full.

See also

sFnd::IPortAdv::TriggerMovesInGroup For starting these moves

sFnd::IMotionAdv::TriggerMove For starting these moves

double sFnd::IMotionAdv::MovePosnDurationMsec (int32_t targetPosn, bool targetIsAbsolute = `false`)

Parameters

[in] **targetPosn** Target position. This is either the absolute position of relative to the current position depending on the *targetIsAbsolute* argument.

[in] **targetIsAbsolute** the target is an absolute destination.

Returns

duration (milliseconds)

Using the currently specified kinematic constraints, return the expected duration (in milliseconds) of the move from when it starts to when the move command completes.

This function is useful for setting up timeouts or helping to schedule other events in your application.

Coding Example

```
// SAMPLE: Using MovePosnDurationMsec to set up a reasonable timeout
    for a move to complete

//define a timeout in case the node is unable to complete the move
double timeout =  mynode.Motion.Adv.MoveDurationMsec(1000) + 100;
double StartTime = myMgr.TimeStampMsec();

//This will loop checking to see if the node went Move done with the
    defined timeout
```

```
while (!mynode.Motion.MoveWentDone() && myMgr.TimeStampMsec() <
     StartTime+timeout)
{//Wait for move to complete
}
```

virtual double sFnd::IMotionAdv::MovePosnHeadTailDurationMsec (**int32_t targetPosn,**
bool targetIsAbsolute = `false`**,**
bool hasHead = `true`**,**
bool hasTail = `true`
)

virtual

Parameters

[in] **targetPosn**	Target position. This is either the absolute position of relative to the current position depending on the *theType* argument. The triggered type move will prevent execution of the motion until a trigger event is detected at the node.
[in] **targetIsAbsolute**	the target is an absolute destination.
[in] **hasHead**	Do not exceed the HeadTailVelLimit until after HeadDistance steps have been sent.
[in] **hasTail**	Do not exceed the HeadTailVelLimit during the last TailDistance steps of the move.

Returns

motion duration (milliseconds)

Using the currently specified kinematic constraints, return the expected duration (in milliseconds) of the move from when it starts to when the move command completes.

This function is useful for setting up timeouts or helping to schedule other events in your application.

Coding Example

```
// SAMPLE: Using MovePosnHeadTailDurationMsec to set up a reasonable
     timeout for a head-tail move to complete

//define a timeout in case the node is unable to complete the move
double timeout =  mynode.Motion.Adv.MovePosnHeadTailDurationMsec(1000)
     + 100;
double StartTime = myMgr.TimeStampMsec();

//This will loop checking to see if the node went Move done with the
     defined timeout
while (!mynode.Motion.MoveWentDone() && myMgr.TimeStampMsec() <
     StartTime+timeout)
{//Wait for move to complete
}
```

virtual size_t sFnd::IMotionAdv::MovePosnHeadTailStart (**int32_t targetPosn,**
bool targetIsAbsolute = `false`**,**

pure virtual

```
                                    bool    isTriggered = false,
                                    bool    hasHead = true,
                                    bool    hasTail = true,
                                    bool    hasDwell = false
                                )
```

Parameters

[in]	**targetPosn**	The target position. This is either the absolute position of relative to the current position depending on the *theType* argument. The triggered type move will prevent execution of the motion until a trigger event is detected at the node.
[in]	**targetIsAbsolute**	the target is an absolute destination.
[in]	**isTriggered**	The move awaits a trigger event to start the motion if true.
[in]	**hasHead**	Do not exceed the HeadTailVelLimit until after HeadDistance steps have been sent.
[in]	**hasTail**	Do not exceed the HeadTailVelLimit during the last TailDistance steps of the move.
[in]	**hasDwell**	delay the next move being issued by DwellMs after this move profile completes.

Returns

Number of additional moves the node will accept.

The move is kinematically limited by the current **acceleration**, **velocity**, as well as **Head** and **tail** distances.

Move requests are buffered and requests initiated during an active move will execute immediately after the active move completes. Triggered moves will wait for a trigger event before executing. The return value will tell you the number of additional moves that may be queued(up to 16) when this request is processed. You can also use the *Status Register's* **_cpmStatusRegFlds::MoveBufAvail** field, which will be deasserted when the buffer is full.

See also

sFnd::IPortAdv::TriggerMovesInGroup For starting these moves

sFnd::IMotionAdv::TriggerMove For starting these moves

```
virtual size_t
sFnd::IMotionAdv::MovePosnStart     ( int32_t  targetPosn,
                                      bool     targetIsAbsolute = false,
                                      bool     isTriggered = false,
                                      bool     hasDwell = false
                                    )                                    pure virtual
```

Parameters

[in]	**targetPosn**	Target position. This is either the absolute position of relative to the current position depending on the *theType* argument. The triggered type move will prevent execution of the motion until a trigger event is detected at the node.
[in]	**targetIsAbsolute**	the target is an absolute destination.
[in]	**isTriggered**	The move awaits a trigger event to start the motion if true.
[in]	**hasDwell**	The move will wait the dwell parameter amount of time before

starting the next move.

Returns
Number of additional moves the node will accept.

The move is kinematically limited by the current **acceleration**, and **velocity** limits.

Move requests are buffered and requests initiated during an active move will execute immediately after the active move completes. Triggered moves will wait for a trigger event before executing. The return value will tell you the number of additional moves that may be queued(up to 16) when this request is processed. You can also use the *Status Register's* **_cpmStatusRegFlds::MoveBufAvail** field, which will be deasserted when the buffer is full.

See also
sFnd::IPortAdv::TriggerMovesInGroup For starting these moves
sFnd::IMotionAdv::TriggerMove For starting these moves

double sFnd::IMotionAdv::MoveVelDurationMsec (double targetCountsPerSec)

Parameters
[in] **targetCountsPerSec** Target velocity. (counts/sec)

Returns
time to reach target velocity (milliseconds)

Using the currently specified kinematic constraints, return the expected duration (in milliseconds) of the move's acceleration to the target velocity.

This function is useful for setting up timeouts or helping to schedule other events in your application.

Coding Example

```
// SAMPLE: Using MoveVelDurationMsec to set up a reasonable timeout
    for a velocity move to complete

//define a timeout in case the node is unable to complete the move
double timeout = mynode.Motion.MoveVelDurationMsec() + 100;
double StartTime = myMgr.TimeStampMsec();

//This will loop checking to see if the node went Move done with the
    defined timeout
while (!mynode.Motion.VelocityReachedTarget() &&
    myMgr.TimeStampMsec() < StartTime+timeout)
{//Wait for move to complete
}
```

virtual size_t sFnd::IMotionAdv::MoveVelStart (double targetPosn, bool isTriggered = false)

pure virtual

Parameters
[in] **targetPosn** Target velocity. If the *triggered* argument is true, this move will wait for a

trigger event to be detected at the node.

[in] **isTriggered** The move awaits a trigger event to start the motion if true.

Returns

Number of additional moves the node will accept.

The move is kinematically limited by the current **acceleration limit**.

Move requests are buffered and requests initiated during an active move will execute immediately after the active move completes. Triggered moves will wait for a trigger event before executing. The return value will tell you the number of additional moves that may be queued(up to 16) when this request is processed. You can also use the *Status Register's* **_cpmStatusRegFlds::MoveBufAvail** field, which will be deasserted when the buffer is full.

virtual size_t sFnd::IMotionAdv::TriggerGroup ( ) pure virtual

Returns

This node's current group number

If zero, there is no trigger group assigned and the group trigger will be ignored.

virtual void sFnd::IMotionAdv::TriggerGroup (size_t groupNumber) pure virtual

Parameters

[in] **groupNumber** Set to non-zero to assign this node to a triggerable group. To remove from a group set to zero.

Remarks

The trigger group allows multiple nodes to start motion with low latency. For example a machine with multiple XYZ axes could assign *groupNumber* 1 to one set and 2 to another. Releasing one group with a **sFnd::IPortAdv::TriggerMovesInGroup(size_t)** call, or by calling **TriggerMovesInMyGroup** on one of the Nodes.

Coding Example

```
// SAMPLE: This will set two Nodes to have the same trigger group
mynode.Motion.Adv.TriggerGroup(1);
myothernode.Motion.Adv.TriggerGroup(1);

//set the last node to have a different trigger group
myThirdNode.Motion.Adv.TriggerGroup(2);
```

virtual void sFnd::IMotionAdv::TriggerMove ( ) pure virtual

Call this function to release a waiting move.

Note

If there are no waiting moves or if the Node is currently executing a move, this command will be ignored.

Coding Example

```
// SAMPLE: This loads a relative triggered move into the move buffer
mynode.Motion.Adv.MovePosnStart(1000,false,true);

//No motion has occurred yet...

mynode.Motion.Adv.TriggerMove();
//Motion now begins
```

virtual void sFnd::IMotionAdv::TriggerMovesInMyGroup ( ) pure virtual

Call this function to release waiting moves in this trigger group.

Note

If there are no waiting moves, or if the node is currently executing a move this command will be ignored on that node.

/see CPMtriggerPage

Coding Example

```
// SAMPLE: This loads a relative triggered move into each node
mynode.Motion.Adv.MovePosnStart(1000,false,true);
myothernode.Motion.Adv.MovePosnStart(1000,false,true);

//No motion has occurred yet...

mynode.Motion.Adv.TriggerMovesInMyGroup();
//If both Nodes share a trigger group, then both node will start
    motion now
//If the nodes do not share a trigger group only mynode will begin motion
```

Member Data Documentation

ValueUnsigned sFnd::IMotionAdv::AfterStartDistance

Access the *A after Start* event distance

Note
The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "Regular" 6400cnts/rev for "Enhanced").

See also
cpmStatusRegFlds::AFromStart for the associated Status event

Coding Example

```
// SAMPLE: Getting and setting the A after start distance
myNode.Motion.Adv.AfterStartDistance = 10000.;              // Sets
double Adist = myNode.Motion.Adv.AfterStartDistance;    // Gets
```

ValueUnsigned sFnd::IMotionAdv::BeforeEndDistance

Access the *B before End* event distance in steps.

Note
The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "Regular" 6400cnts/rev for "Enhanced").

See also
cpmStatusRegFlds::BFromEnd for the associated Status event

Coding Example

```
// SAMPLE: Getting and setting the B before end distance
myNode.Motion.Adv.BeforeEndDistance = 10000.;               // Sets
double Bdist = myNode.Motion.Adv.BeforeEndDistance; // Gets
```

ValueDouble sFnd::IMotionAdv::DecelLimit

Access the decceleration limit value for the next asymmetric move. The value is specified using this node's user acceleration units. Changed values will apply to the next move. The default units are RPM/sec

Coding Example

```
// SAMPLE: Getting and setting the Decceleration Limit
myNode.Motion.Adv.DecelLimit = 10000.;              // Sets
double accLim = myNode.Motion.Adv.DecelLimit;   // Gets
```

See also
INode::AccUnit(enum _accUnits newUnits) Changing Acceleration Units

ValueUnsigned sFnd::IMotionAdv::HeadDistance

Access the head distance for the next issued head-tail move in steps.

Note
The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "Regular" 6400cnts/rev for "Enhanced").

Coding Example

```
// SAMPLE: Getting and setting the Head Distance
myNode.Motion.Adv.HeadDistance = 10000.;                // Sets
double headDist = myNode.Motion.Adv.HeadDistance;   // Gets
```

Examples:
Example-MultiThreaded/Axis.h.

ValueDouble sFnd::IMotionAdv::HeadTailVelLimit

Access the current head/tail velocity limit for the next head-tail move. The value is specified in the current velocity units. Changed values will apply to the next issued move. The default units are RPM.

Coding Example

```
// SAMPLE: Getting and setting the Head/Tail Velocity Limit
myNode.Motion.Adv.HeadTailVelLimit = 1000.;             // Sets
double velLim = myNode.Motion.Adv.HeadTailVelLimit; // Gets
```

See also
INode::VelUnit(enum _velUnits newUnits)

ValueUnsigned sFnd::IMotionAdv::TailDistance

Access the tail distance for the next issued head-tail move in steps.

Note
The resolution of this value will be the same as the positioning resolution of the motor (800cnts/rev for "Regular" 6400cnts/rev for "Enhanced").

Coding Example

```
// SAMPLE: Getting and setting the Tail Distance
myNode.Motion.Adv.TailDistance = 10000.;                // Sets
double tailDist = myNode.Motion.AdvTaildDistance;   // Gets
```

Examples:
Example-MultiThreaded/Axis.h.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFoundation 2.0

sFnd::INode Class Reference abstract

List of all members | Classes | Public Types | Public Member Functions | Public Attributes

Detailed Description

This class defines a "super" node whose framework exposes the Node Level features available for your application by a series of public attribute references. There are common member functions for these feature objects to determine if the node supports the feature.

This class also implements:

- Velocity Units
- Acceleration Units
- Torque Units
- Enable Requests

Coding Example

```
// Create instance of sFoundation
SysManager myMgr;
// Define a port SC4-HUB attached to
myMgr.ComHubPort(0, 13);
// Tell system to open this one port
myMgr.PortsOpen(1);
// Create useful reference to the first port object
IPort &myPort = myMgr.Ports(0);
// Test for presence of nodes
if (myPort.NodeCount() == 0) {
    // Bail out, no nodes
    return;
}
// If there are nodes present, get a reference to
    the first one
INode &myNode =  myPort.Nodes(0);
// You can now interact with myNode
myNode.EnableReq(true);
```

See also

sFnd::SysManager For the root object for interacting with ClearPath-SC motors.

sFnd::SysManager::Ports To get references to the **IPort** objects created when **SysManager::PortsOpen** has successfully opened.

sFnd::IPort::Nodes To get references to **INode** objects detected on a port.

Examples:

Example-GPIO/Example-GPIO.cpp, **Example-Homing/Example-Homing.cpp**, **Example-Motion/Example-Motion.cpp**, **Example-MultiThreaded/Axis.h**, **Example-SingleThreaded/Axis.h**, and **Example-StatusAlerts/Example-StatusAlerts.cpp**.

Classes

class **UseMutex**
A Simple and Safe Mutex mechanism. More...

Public Types

enum **_accUnits** { **RPM_PER_SEC**, **COUNTS_PER_SEC2** }
Units for Accelerations parameters and status. More...

enum **_velUnits** { **RPM**, **COUNTS_PER_SEC** }
Units for velocity parameters and status. More...

enum **_trqUnits** { **PCT_MAX**, **AMPS** }
Units for torque parameters and status. More...

Public Member Functions

virtual void **AccUnit** (enum **_accUnits** newUnits)=0
Change the acceleration units. More...

enum **_accUnits** **AccUnit** ()
Get the current velocity units. More...

virtual void	**VelUnit** (enum **_velUnits** newUnits)=0 Change the velocity units. More...
enum **_velUnits**	**VelUnit** () Get the current velocity units. More...
virtual void	**TrqUnit** (enum **_trqUnits** newUnits)=0 Change the torque units. More...
enum **_trqUnits**	**TrqUnit** () Get the current torque units. More...
void	**EnableReq** (bool newState) Thread safe enable request. More...
bool	**EnableReq** () Get the current *Enable Request* state. More...

Public Attributes

IPort &	**Port** Get a reference to the port object this node is attached to.
IInfo &	**Info** This reference allows for access about generic information about a node.
IMotion &	**Motion** Get a reference to the motion feature interface object.
IStatus &	**Status** Get a reference to the status information object.
ILimits &	**Limits**

	Get a reference to the limits feature object.
ISetup &	**Setup** Get a reference to our Setup object.
INodeAdv &	**Adv** Get a reference to the advanced features object.

Member Enumeration Documentation

enum sFnd::INode::_accUnits

Enumerator	
RPM_PER_SEC	Revolutions/Minute/Second.
COUNTS_PER_SEC2	Counts/Second^2.

enum sFnd::INode::_trqUnits

Enumerator	
PCT_MAX	Percentage of drive maximum.
AMPS	Drive delivery amps.

enum sFnd::INode::_velUnits

Enumerator	
RPM	Revolutions/Minute.
COUNTS_PER_SEC	Counts/Second.

Member Function Documentation

virtual void sFnd::INode::AccUnit (enum _accUnits newUnits) pure virtual

Parameters

[in] **newUnits** Change the acceleration units to *newUnits*.

Coding Example

```
// Change the Units of Acceleration to RPM/Sec
mynode.AccUnit(mynode.RPM_PER_SEC);

// Change the Units of Acceleration to Cnts/sec/sec
mynode.AccUnit(mynode.COUNTS_PER_SEC2);
```

Examples:

Example-Motion/Example-Motion.cpp.

enum _accUnits sFnd::INode::AccUnit ( ) inline

Returns

Current acceleration unit

void sFnd::INode::EnableReq (bool newState) inline

Parameters

[in] **newState** Change the state of the *Enable Request*.

This function allows you to assert the user *Enable Request*. If *Enable Request* is asserted the drive will enable unless an overriding disable nodeStop condition or a shutdown persists. If a

shutdown, or nodestop prevents the Node from enabling, then as soon as the condition is removed by **clearing the Nodestop** or **clearing the alert** The Node will immediately enable(there is no need to re-send the enable request).

Examples:

Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

bool sFnd::INode::EnableReq ( ) inline

This function will return the state of the user *Enable Request*.

Note

The Node may not be Enabled even if this request is asserted. A **NodeStop**, or **Shutdown Alert** can prevent the enable request from executing.

virtual void sFnd::INode::TrqUnit (enum _trqUnits newUnits) pure virtual

Parameters

[in] **newUnits** New torque unit

Coding Example

```
// Change the Units of torque to percent of max
mynode.AccUnit(mynode.PCT_MAX);

// Change the Units of torque to Amps
mynode.AccUnit(mynode.AMPS);
```

enum _trqUnits sFnd::INode::TrqUnit ( ) inline

Returns

Current torque unit

enum _velUnits sFnd::INode::VelUnit ( ) inline

Returns

Current velocity unit

virtual void sFnd::INode::VelUnit (enum _velUnits newUnits) pure virtual

Parameters

[in] **newUnits** Set new velocity unit

Coding Example

```
// Change the Units of Velocity to RPM
mynode.VelUnit(mynode.RPM);

// Change the Units of Velocity to Cnts/Sec
mynode.VelUnit(mynode.COUNTS_PER_SEC);
```

Examples:

Example-Motion/Example-Motion.cpp.

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

- C:/Git/inc/inc-pub/pubSysCls.h

List of all members | Public Member Functions

sFnd::INode::UseMutex Class Reference

Detailed Description

This mechanism is used to create critical sections of code when multiple threads are accessing a node, in order to perform read-modify-write operations by one of the threads, for example. An instance of this simplifies error handling whether node mutex is utilized.

Note

This instance should generally be created in a C++ code block that requires the node mutex. It must go out of scope before the code stream after the mutually exclusive tasks are complete. This ensures the release of the mutex in the case of operational failures that causes the called function to throw an error.

Coding Example

```
// This function will lock out any other parallel
    attempts to
// access the node until it returns.
void myFuncWithLock(INode &myNode) {
    // Lock is taken automatically
    INode::UseMutex myLock(myNode);

    // Perform read-modify-write action

    // Exiting this function guarantees unlock occurs
}
```

Note

Care should be taken to hold this instance as short as possible to avoid starving competing threads from executing.

Public Member Functions

UseMutex (**INode** &ourNode)
Create a safe and simple mutex. More...

Constructor & Destructor Documentation

sFnd::INode::UseMutex::UseMutex (INode & ourNode)

Parameters

[in,out]	**ourNode**	This is passed a reference to the node you wish to perform the mutual exclusion on.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFoundation 2.0

sFnd::INodeAdv Class Reference

Detailed Description

This implements the objects required to access any advanced features of a node.

Provides access to the following advanced features:

- Attention Generation

Inherits **sFnd::IObjWithNode**.

Public Attributes

IAttnNode &	**Attn** Attention Generation Feature Setup. More...

Additional Inherited Members

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Data Documentation

IAttnNode& sFnd::INodeAdv::Attn

Access the node's *Attention Generation* feature.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFnd::IObj Class Reference

List of all members | Public Member Functions

Detailed Description

This is the base object for all feature objects. It provides the function for determining whether a "feature object" is supported.

Inherited by **sFnd::IObjWithNode**, and **sFnd::IObjWithPort**.

Public Member Functions

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

virtual bool sFnd::IObj::Supported ( ) virtual

Returns
Returns true if this object is supported by this node.

A programmatic way of determining if this node supports this feature. Some features are not available on certain types of nodes.

Reimplemented in **sFnd::IPortAdv**.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFnd::IObjWithNode Class Reference

List of all members | Public Member Functions

Detailed Description

This is the base object for all Node Level feature Objects, and provides a common function each feature to have a back link to the owner node.

Inherits **sFnd::IObj**.

Inherited by **sFnd::IAttnNode**, **sFnd::IHoming**, **sFnd::IInfo**, **sFnd::IInfoEx**, **sFnd::ILimits**, **sFnd::IMotion**, **sFnd::IMotionAdv**, **sFnd::INodeAdv**, **sFnd::ISetup**, **sFnd::ISetupEx**, **sFnd::IStatus**, **sFnd::IStatusAdv**, and **sFnd::ValueBase**.

Public Member Functions

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

INode& sFnd::IObjWithNode::Node ( ) inline

Returns
Reference to our node object.

This member function is useful when you want to return the root of your node when you are using a feature object. For example, your are interacting with a status object **IStatus** and you wish to disable the node.

Coding Example

```
// Access another feature from another feature
myStatusObj.Node().EnableReq(false));
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members

sFnd::IObjWithPort Class Reference

Detailed Description

This is the base object for all Port Level feature Objects.

Inherits **sFnd::IObj**.

Inherited by **sFnd::IAttnPort**, **sFnd::IBrakeControl**, **sFnd::IGrpShutdown**, and **sFnd::IPortAdv**.

Additional Inherited Members

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions | Public Attributes

sFnd::IPort Class Reference abstract

Detailed Description

This class defines a "super" Port whose framework exposes the Port Level features available for your application by a series of public attribute references. There are common member functions for these feature objects to determine if the node supports the feature.

This class also implements:

- References to Node object on the Port
- Functions to restart the Port, and determine if the Port is currently Open
- A function to issue a Port wide Nodestop to every Node on this Port
- A function to record the command history of all communication on this Port

Coding Example

```
// Create instance of sFoundation
SysManager myMgr;
// Define a port SC4-HUB attached to
myMgr.ComHubPort(0, 13);
// Tell system to open this one port
myMgr.PortsOpen(1);
// Create useful reference to the first port object
IPort &myPort = myMgr.Ports(0);
// Test for presence of nodes
if (myPort.NodeCount() == 0) {
// Bail out, no nodes
    return;
}
```

See also

sFnd::SysManager For the root object for interacting with ClearPath-SC motors.

sFnd::SysManager::Ports To get references to the **IPort** objects created when **SysManager::PortsOpen** has successfully opened.

Examples:

Example-GPIO/Example-GPIO.cpp, **Example-Homing/Example-Homing.cpp**, **Example-Motion/Example-Motion.cpp**, **Example-StatusAlerts/Example-StatusAlerts.cpp**, and **HelloWorld/HelloWorld.cpp**.

Public Member Functions

INode &	**Nodes** (size_t index) Return a reference to the indexed node on this port. More...
virtual Uint16	**NodeCount** ()=0 Return the count of nodes active on this port. More...

virtual void	**RestartCold** ()=0 Restart this nodes to its powered on state and re-connect. More...
virtual **openStates**	**OpenState** ()=0 Get port state. More...
virtual bool	**WaitForOnline** (int32_t timeoutMsec=15000)=0 Wait for the port to go online. More...
virtual void	**CommandTraceSave** (const char *filePath)=0 Create command trace file. More...
virtual void	**NodeStop** (mgNodeStopReg stopType=STOP_TYPE_IGNORE)=0 Send a high-priority *Node Stop* command to all the nodes attached to this serial port. If stopType is STOP_TYPE_IGNORE, the nodes will use their default stop type instead of the specific type. More...

Public Attributes

IGrpShutdown &	**GrpShutdown** Group Shutdown Feature.
IBrakeControl &	**BrakeControl** Brake Control Feature of SC-Hub.
IPortAdv &	**Adv** Access to advanced features for the port.

Member Function Documentation

virtual void sFnd::IPort::CommandTraceSave (const char * filePath) pure virtual

This function creates a file which contains a detailed transaction log of the all the commands send and data received by the host. It will remember the last 4096 commands and/or responses.

Teknic support staff may ask for the file created via this function to determine where a programming error occurred.

virtual Uint16 sFnd::IPort::NodeCount ( ) pure virtual

Returns

Return the count of nodes present on this port.

Use this function to find out how many active nodes are attached to this port. If the port if offline, this will be zero.

Coding Example

```
// Check to see if there are nodes on the Port.
if (myPort.NodeCount() == 0) {
// Bail out, no nodes
return;
}
```

Examples:

Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

INode& sFnd::IPort::Nodes (size_t index)

Parameters

[in] **index** Get a reference for the node. This is the zero based index, meaning zero is the first node. The first node is the node closest to the host serial port in the ring.

Examples:

Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, and Example-

StatusAlerts/Example-StatusAlerts.cpp.

virtual void sFnd::IPort::NodeStop (mgNodeStopReg stopType = STOP_TYPE_IGNORE) pure virtual

Parameters

[in] **stopType** Will force this stop type to all nodes on this port.

This is a fast acting command that can be used to stop all motion on a port. If a modifier field is set, such as E-Stop, all motion will be blocked until this condition is cleared.

See also

sFnd::IMotion::NodeStop For initiating node stops at a single node.

Coding Example

```
myport.NodeStop(STOP_TYPE_ABRUPT);              //This issues
    an abrupt nodestop on all the nodes on this port
myport.NodeStop(STOP_TYPE_RAMP_WITH_DECEL);     //This issues
    a nodestop to ramp to zero velocity using the current
    decel parameter's on all the nodes on this port
```

virtual openStates sFnd::IPort::OpenState ( ) pure virtual

Returns

Current port operational state.

Use this function to ascertain the operation state of this port.

Note

Ports operate independently from each other. In other words, a failure on connections on one port should not affect another port.

Examples:

Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

virtual void sFnd::IPort::RestartCold ( ) pure virtual

This causes the nodes on this port to restart and re-establish contact with the detected node configuration.

virtual bool (int32_t timeoutMsec = 15000) pure virtual

sFnd::IPort::WaitForOnline

Parameters

`[in]` **timeoutMsec** Time to wait for port to return or go online

Returns

True if the port came back online.

This function is useful after restarting the network after power is lost.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions | Public Attributes

sFnd::IPortAdv Class Reference abstract

Detailed Description

This implements the objects required to access any advanced features at the port level. Currently, this class provides a function to trigger groups of Nodes on this Port.

Inherits **sFnd::IObjWithPort**.

Public Member Functions

virtual void	**TriggerMovesInGroup** (size_t groupNumber)=0 Trigger a group of nodes to initiate their moves. More...
bool	**Supported** () Query if this feature is supported in this node. More...

Public Attributes

IAttnPort &	**Attn** Access the Attention Handling feature object. More...

Member Function Documentation

bool sFnd::IPortAdv::Supported ( ) virtual

Returns

Returns true if this object is supported by this node.

A programmatic way of determining if this node supports this feature. Some features are not available on certain types of nodes.

Reimplemented from **sFnd::IObj**.

virtual void sFnd::IPortAdv::TriggerMovesInGroup (size_t groupNumber) pure virtual

Parameters

[in] **groupNumber** Group to signal on the network attached to this port.

If there are no waiting moves or nodes setup at this *groupNumber*, this command will be ignored with no actions taken.

See also

Triggered Moves for more infomation on triggered moves

sFnd::IMotionAdv for configuring/loading triggered moves

sFnd::IMotionAdv::TriggerGroup to set/get a Node's trigger Group

Coding Example

```
// This example will load some triggered moves onto a
     group of Nodes, then Trigger them at the Port Level

myNode1.Motion.Adv.MovePosnStart(1000,false,true);
     /Load a 1000 cnt relative triggered move onto Node
     1
myNode2.Motion.Adv.MovePosnStart(1000,false,true);
     /Load a 1000 cnt relative triggered move onto Node
     2
```

```
myNode3.Motion.Adv.MovePosnStart(1000,false,true);
     /Load a 1000 cnt relative triggered move onto Node
     3

//Set the Trigger group of Node's 1, 2, and 3
myNode1.Motion.Adv.TriggerGroup(1);
myNode2.Motion.Adv.TriggerGroup(1);
myNode3.Motion.Adv.TriggerGroup(2);

//Trigger the Loaded Moves
myPort.Adv.TriggerMovesInGroup(1);  //This will
     trigger Node 1 and 2 (Group 1)
myPort.Adv.TriggerMovesInGroup(2);  //This will only
     trigger Node 3 (Group 2)
```

Member Data Documentation

IAttnPort& sFnd::IPortAdv::Attn

This object allows access to the advanced *Attention Request* feature. This feature allows for non-polled asynchronous event handling.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions | Public Attributes

sFnd::ISetup Class Reference abstract

Detailed Description

This class accesses and manages node setup features. These include:

- Loading Configuration files from your application
- Saving Configuration files

Inherits **sFnd::IObjWithNode**.

Public Member Functions

virtual void	**ConfigLoad** (const char *filePath, bool doReset=false)=0 Load configuration File. More...
virtual void	**ConfigSave** (const char *filePath)=0 Save the node's configuration. More...
virtual bool	**AccessLevelIsFull** ()=0 Returns true if the node is in full access mode on this network. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Public Attributes

ValueDouble	**DelayToDisableMsecs** Manage the delay to actual disable. More...
ISetupEx	**Ex** Expert setup features access.

Member Function Documentation

virtual bool sFnd::ISetup::AccessLevelIsFull ( ) pure virtual

Returns true if the node is in full access mode on this network. The only time a node may not be in full access mode is if ClearView is connected through the USB hub on the motor and the user has taken Full Access through the USB diagnostic port via ClearView.

Coding Example

```
// This example will check to see if the application
    has full access to the Node
if(myNode.Setup.AccessLevelIsFull())
{ ...we have full access. Normal operation can continue...}
else
{ ...we don't have full access. Set ClearView to
    Monitor Mode before continuing...}
```

virtual void sFnd::ISetup::ConfigLoad (const char * filePath, bool doReset = false) pure virtual

Parameters

- [in] **filePath** Pointer to C-String file path to the file to load.
- [in] **doReset** Set this to true to force the node to reset at the end of load.

Load the configuration file specified by *filePath* to this node.

Coding Example

```
// This function will load the Config file. You will
    have to edit the file path for your application
char* myconfigFile = "...\MyConfigFolder\MyConfigFile.mtr";
mynode.Setup.ConfigLoad(myconfigFile);
```

virtual void sFnd::ISetup::ConfigSave (const char * filePath) pure virtual

Parameters

[in] **filePath** Pointer to C-String file path to save the configuration at.

Save the node's configuration to the *filePath* specified. This would typically be performed through ClearView right after tuning each axis.

Coding Example

```
//Save the config file before starting to the windows
    Temp file path
char tempPath[256];
GetTempPathA(256, tempPath);
char myconfigFile[269];
sprintf(myconfigFile, "%s\myConfig.mtr", tempPath);
mynode.Setup.ConfigSave(myconfigFile);
```

Member Data Documentation

ValueDouble sFnd::ISetup::DelayToDisableMsecs

This allows an electrical brake to engage prior to the actual disable. The units are milliseconds.

See also

sFnd::IBrakeControl for more information on configuring, and controlling brakes

Coding Example

```
// SAMPLE: Getting and setting the delay to disable time.
myNode.Setup.DelayToDisableMsecs = 100.;
     // Sets
double myDwell = myNode.Setup.DelayToDisableMsecs;  // Gets
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Attributes

sFnd::ISetupEx Class Reference

Detailed Description

This class allows access to the more advanced setup options for this node. Items accessed in this class may be overridden by ClearView or configuration file loads.

Inherits **sFnd::IObjWithNode**.

Public Attributes

ValueDouble	**NetWatchdogMsec** Network watchdog timer. More...

Additional Inherited Members

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Data Documentation

ValueDouble sFnd::ISetupEx::NetWatchdogMsec

This allows the adjustment of a network watchdog timer that will force a ramped Node Stop if the host stops communicating with the node. This is a safety feature and should not be changed except in rare application specific circumstances.

The ramp stop acceleration is setup via the E-Stop deceleration rate in ClearView. This is found in the details dropdown menu in ClearView.

Set this to zero to disable. Units are milliseconds.

Note

This setting can be **saved into a config file**, and is overwritten when a **config file is loaded**.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions | Public Attributes

sFnd::IStatus Class Reference abstract

Detailed Description

This Node Level class accesses and manages the node status information. Node Status information is divided into two types of registers:

- Status Registers
- Alert/Shutdown Registers

Status Registers contain current information about the node's dynamic state such as whether a node: is in motion, has gone move done, etc. The main Realtime Status Register is maintained by the node and is updated each sample-time to reflect the node's state. To reduce latency and allow the capture of short transient events, the Realtime Status Register feeds a series of accumulating registers. These registers store transitional events until they are accessed by the application program. These are:

- The deassertions are captured by the Status Fallen Register.
- The assertions are captured by the Status Risen & Attention Source Register. This register is the source of the autonomous Attention Packets when enabled on "Advanced" motors.
- The Status Accumulated Register is a bitwise OR accumulation of the Realtime Status Register. This register is useful for determining if an assertion has occurred since the last access.

The Alert Register contains information regarding a shutdown the Node may have experienced during normal operation. Depending on the type of alert, the application may be able to recover and/or indentify the root cause of the failure.

Access to these registers is provided through the **ValueAlert** and **ValueStatus** classes.

Coding Example

```
//This example code will check a node's status for
    any current alerts and
//print out the state string of any alerts found
```

```
//This creates an Empty String for us to put our
    Alert State String into
char alertList[256];

// Refresh the Real-Time Status, and Alert Register
theNode.Status.RT.Refresh();
myNode.Status.Alerts.Refresh();


// Check the Real-Time Status Register's
    "AlertPresent" bit
// The bit is set true if there are alerts in the
    alert register
if (theNode.Status.RT.Value().cpm.AlertPresent)
{
printf("Node has alerts!\n");
}
else
{
printf("Node has no alerts!\n");
}

// get an alertReg reference, and check directly
    for alerts
if (myNode.Status.Alerts.Value().isInAlert())
{
// Put a text description of all alert bits set
    into our empty string, then print the string
myNode.Status.Alerts.Value().StateStr(alertList, 256);
printf("Node has alerts! Alerts: %s", alertList);

// Check for specific alerts by accessing the
    cpmAlertFlds within the alertReg Struct
if (myNode.Status.Alerts.Value().cpm.TrackingShutdown)
{
printf("Tracking Shutdown\n");
}
}
```

See also

- **sFnd::ValueAlert** for more information on accessing the **Node**'s Alert Register
- **sFnd::ValueStatus** for more information on accessing the **Node**'s Status Register
- **cpmStatusRegFlds** for a complete list of ClearPath-SC Status Fields
- **cpmAlertFlds** for a complete list of ClearPath-SC **Alerts**

Inherits **sFnd::IObjWithNode**.

Public Member Functions

virtual void	**AlertsClear** ()=0 Clear non-serious alerts. More...
virtual bool	**IsReady** ()=0 If true, the node is ready for motion. More...
virtual bool	**HadTorqueSaturation** ()=0 If true, the node is ready for motion. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Public Attributes

ValueStatus	**RT** Real Time Status Register Reference. More...
ValueStatus	**Rise** Status Risen Register Reference. More...

ValueStatus	**Fall** Status Fallen Register Reference. More...
ValueStatus	**Accum** Status Accumulated Register Reference. More...
ValueAlert	**Alerts** Alerts/Shutdowns Registers Reference. More...
ValueDouble	**RMSlevel** RMS level Reference. More...
ValuePowerReg	**Power** RMS level Reference. More...
IStatusAdv &	**Adv** Advanced status information. More...

Member Function Documentation

virtual void sFnd::IStatus::AlertsClear ( ) pure virtual

Clear all non-serious alerts and allow node to enable if an *Enable Request* is still pending.

Examples:
Example-Homing/Example-Homing.cpp, **Example-Motion/Example-Motion.cpp**, and **Example-StatusAlerts/Example-StatusAlerts.cpp**.

virtual bool sFnd::IStatus::HadTorqueSaturation ( ) pure virtual

Returns
True if the node is ready for motion.

This function is used to determine if a Torque saturation event has occurred since the last query.

Note
The *Warnings* register is queried when this function is called

Examples:
Example-StatusAlerts/Example-StatusAlerts.cpp.

virtual bool sFnd::IStatus::IsReady ( ) pure virtual

Returns
True if the node is ready for motion.

This function is used to determine the availability of the move generator to begin operations.

Member Data Documentation

ValueStatus sFnd::IStatus::Accum

The *Status Accumulated Register* reference.

IStatusAdv& sFnd::IStatus::Adv

Access to extra status information only available in advanced ClearPath-SC nodes.

ValueAlert sFnd::IStatus::Alerts

The *Alerts/Shutdowns Register* reference.

Examples:
Example-StatusAlerts/Example-StatusAlerts.cpp.

ValueStatus sFnd::IStatus::Fall

The *Status Fallen Register* reference.

ValuePowerReg sFnd::IStatus::Power

A reference to the Node's bus and backup power status.

ValueStatus sFnd::IStatus::Rise

The *Status Risen Register* reference.

ValueDouble sFnd::IStatus::RMSlevel

A reference to the Node's *RMS Level*.

ValueStatus sFnd::IStatus::RT

The *Real Time Status Register* reference

Examples:
Example-GPIO/Example-GPIO.cpp, and Example-StatusAlerts/Example-StatusAlerts.cpp.

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

- C:/Git/inc/inc-pub/pubSysCls.h

List of all members | Public Attributes

sFnd::IStatusAdv Class Reference

Detailed Description

This class accesses and manages the advanced node status information. It provides access to:

- Capture Registers

See also

Position Capture for information on configuring inputs to capture position

Inherits **sFnd::IObjWithNode**.

Public Attributes

ValueSigned	**CapturedHiResPosn** Access the last captured high resolution position value. More...
ValueSigned	**CapturedPos** Return the last at sample-rate position captured. More...

Additional Inherited Members

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Data Documentation

ValueSigned sFnd::IStatusAdv::CapturedHiResPosn

Access the last captured high resolution position value.

Note
This input corresponds to a change of state of the Node's input A

ValueSigned sFnd::IStatusAdv::CapturedPos

Return the last at sample-rate position captured.

Note
This input corresponds to a change of state of the Node's input A

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFnd::SysManager Class Reference

List of all members | Classes | Public Member Functions | Static Public Member Functions

Detailed Description

This class defines the manager object whose framework implements system level functions including:

- Functions to open, close, and restart ports in the system
- A thread safe system dwell
- The system timestamp
- Direct access to ports and nodes within the system

Only one instance of this object should be used in any application.

Coding Example

```
// Create instance of sFoundation
SysManager myMgr;
// Define a port SC4-HUB attached to
myMgr.ComHubPort(0, 13);
// Tell system to open this one port
myMgr.PortsOpen(1);
// Access Node objects through the port
myMgr.Ports(0).Nodes(0);
```

See also

sFnd::IPort to reference port level features

sFnd::INode to reference node level features

Class Library Overview for a high level overview of how these libraries interact

Examples:

Example-GPIO/Example-GPIO.cpp, **Example-Homing/Example-Homing.cpp**, **Example-Motion/Example-Motion.cpp**, **Example-SingleThreaded/Axis.h**, **Example-StatusAlerts/Example-StatusAlerts.cpp**, and **HelloWorld/HelloWorld.cpp**.

Classes

class **ThreadLock**
A safe thread locking mechanism. More...

Public Member Functions

void **ComHubPort** (size_t netNumber, int portNumber, netRates portRate=MN_BAUD_12X)
Simplified setup for ClearPath-SC COM Hub. More...

void **PortsOpen** (size_t portCount)
Open ports to start operations. More...

void **PortsOpen** (int portCount)
Open ports to start operations. More...

void	**PortsClose** () Close all operations down and close the ports. More...
void	**RestartCold** () Close and re-open ports with node restarts and perform system initialization. More...
void	**RestartWarm** () Close and re-open ports and perform system initialization. More...
double	**TimeStampMsec** () Get a time-stamp. More...
std::string &	**TimeStampMsecStr** (std::string &toUpdString) Format a std:string with a time-stamp string. More...
std::string	**TimeStampMsecStr** () Format a std:string with a time-stamp string. More...
void	**Delay** (uint32_t msec) Suspend thread execution. More...
IPort &	**Ports** (size_t index) Get reference to our port object. More...
INode &	**NodeGet** (**multiaddr** theMultiAddr) Convert multi-address to an **INode** reference. More...

Static Public Member Functions

static void	**FindComHubPorts** (std::vector< std::string > &comHubPorts) Search for ClearPath-SC COM Hubs on this machine. More...

Member Function Documentation

void sFnd::SysManager::ComHubPort (size_t netNumber, int portNumber, netRates portRate = MN_BAUD_12X)

Parameters

[in] **netNumber** This is the index of the port to setup. The first port is numbered zero. The range of the netNumbers is from 0 to 2.

[in] **portNumber** This is Windows oriented port number. i.e. COM1 would set *portNumber* = 1.

[in] **portRate** This is set to the desired network rate to operate at. If using a SC-HUB, you can specify the speed as MN_BAUD_24X, else you should use MN_BAUD_12X for 115200 fields/second ports typically found on a PC motherboard.

This function is used to define the relationship between the system's serial port number and the index number(netNumber) in your system. **Coding Example**

```
// This example configures the first port(netNumber=0) to refer to
    windows COM5, using the default baud rate of 115200
myMgr.ComHubPort(0,5);

// This example configures the 2nd port(netNumber=1) to refer to
    windows COM10, using the higher baud rate of 230400
myMgr.ComHubPort(1,10,MN_BAUD_24X);
```

Examples:
Example-GPIO/Example-GPIO.cpp, **Example-Homing/Example-Homing.cpp**, **Example-Motion/Example-Motion.cpp**, **Example-StatusAlerts/Example-StatusAlerts.cpp**, and **HelloWorld/HelloWorld.cpp**.

void sFnd::SysManager::Delay (uint32_t msec)

Parameters

[in] **msec** Delay in milliseconds.

This calls the underlying operating system's thread suspension function to delay execution for this thread.

Coding Example

```
// Create our system instance
SysManager myMgr;
// Wait 100 milliseconds
cout << myMgr.TimeStampMsecStr() << endl;
```

```
myMgr.Delay(100);
cout << myMgr.TimeStampMsecStr() << endl;
```

Examples:
Example-Motion/Example-Motion.cpp.

static void sFnd::SysManager::FindComHubPorts (std::vector< std::string > & comHubPorts) static

Parameters
[out] **comHubPorts** This will hold a vector of strings which can later be passed to ComHubPort to open the given port(s).

Examples:
Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

INode& sFnd::SysManager::NodeGet (multiaddr theMultiAddr)

Parameters
[in] **theMultiAddr** This a system's internal node address.

The returned node reference may exist on any defined ports.

Remarks
A _mnErr object is thrown if the requested address does not exist in your system or the port currently is offline.

Coding Example

```
// Get a reference to a node from a specific address
// 0-15 correspond to Nodes on port 0
// 16-31 correspond to Nodes on port 1
// 32-47 correspond to Nodes on port 2
INode &mynode = myMgr.NodeGet(3);   // Get the 4th node on port 0
INode &mynode2 = myMgr.NodeGet(18); // Get the 3rd node on port 1
```

IPort& sFnd::SysManager::Ports (size_t index)

Parameters
[in] **index** The port index used when initializing the system.

This function is used to get a reference to the IPort object associated with serial port. This would be the same as argument used with the SysManager::ComHubPort function.

See also
SysManager::ComHubPort For setting up a serial port,

Examples:

Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

void sFnd::SysManager::PortsClose ( )

This function will close the serial ports in the system and release all resources related to them.

Note

Any references to Ports, Nodes, or ValueXXXX parameters will no longer be valid after this function is called

Examples:

Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

void sFnd::SysManager::PortsOpen (size_t portCount)

Parameters

[in] **portCount** Number of ports specified

Call this function to start network operations after the ports have been specified.

Note

Ports should be assigned from 0 without skipping any.

Coding Example

```
// Create our system instance
SysManager myMgr;
// Set the first port = COM10
myMgr.ComHubPort(0, 10);
// Set the second port = COM12
myMgr.ComHubPort(1, 12);
// Start system with two ports.
myMgr.PortsOpen(2);
```

See also

IPort::OpenState For information about starting success.

IPort::WaitForOnline To synchronize waiting for online state.

Examples:

Example-GPIO/Example-GPIO.cpp, Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, Example-StatusAlerts/Example-StatusAlerts.cpp, and HelloWorld/HelloWorld.cpp.

void sFnd::SysManager::PortsOpen (int portCount)

Parameters

[in] **portCount** Number of ports specified

Call this function to start network operations after the ports have been specified.

Note

Ports should be assigned from 0 without skipping any.

Coding Example

```
// Create our system instance
SysManager myMgr;
// Set the first port = COM10
myMgr.ComHubPort(0, 10);
// Set the second port = COM12
myMgr.ComHubPort(1, 12);
// Start system with two ports.
myMgr.PortsOpen(2);
```

See also

IPort::OpenState For information about starting success.

IPort::WaitForOnline To synchronize waiting for online state.

void sFnd::SysManager::RestartCold ( )

This function will send a reset node command to restart them to their powered-up state. The serial ports are re-inventoried for nodes and the **SysManager** refreshed for a new startup.

Note

Any references to Ports, Nodes, or ValueXXXX parameters will no longer be valid after this function is called

See also

IPort::OpenState For information about starting success.

IPort::WaitForOnline To synchronize waiting for online state.

void sFnd::SysManager::RestartWarm ( )

This function will close and re-open the serial ports are re-inventoried for nodes and the **SysManager** refreshed for a new startup.

Note

Any references to Ports, Nodes, or ValueXXXX parameters will no longer be valid after this function is called

See also

IPort::OpenState For information about starting success.

IPort::WaitForOnline To synchronize waiting for online state.

double sFnd::SysManager::TimeStampMsec ( )

Returns
High precision time-stamp

This time-stamp will have at least millisecond resolution. For the Windows platform this is actually microsecond resolution.

Note
This time stamp correlates to the time stamp in the **Command Trace** file.

Coding Example

```
//This example show how you can use the TimeStampMsec function to
     establish a reasonable timout for a polling loop.

 double timeout;
timeout = myMgr.TimeStampMsec() + TimeToWait;         //define a
     timeout in case the node is unable to enable

//This will loop checking on some status, and will timeout if the
     status is not found
while (!mynode.SomeStatus()) {
    mynode.SomeStatus.Refresh();
    if (myMgr.TimeStampMsec() > timeout) {
        printf("Error: Timed out waiting for some status\n");
        return -2;
    }
}
```

Examples:
Example-Homing/Example-Homing.cpp, and **Example-Motion/Example-Motion.cpp**.

std::string& sFnd::SysManager::TimeStampMsecStr (std::string & toUpdString)

Updates *toUpdString* with a time-stamp string that correlates to the **Command Trace** feature.

Parameters
`[in,out]` **toUpdString** Reference to string to update

Returns
toUpdString the incoming string reference

Coding Example

```
// Create our system instance
SysManager myMgr;
// String to store results
string myString;
// Print time stamp to cout
```

```
cout << myMgr.TimeStampMsecStr(myString) << endl;
cout << myString << endl;
printf("%s\n", myString.c_str());
```

std::string sFnd::SysManager::TimeStampMsecStr ( )

Returns a time-stamp string that correlates to the **Command Trace** feature.

Coding Example

```
// Create our system instance
SysManager myMgr;
// Print time stamp to cout
cout << myMgr.TimeStampMsecStr() << endl;
printf("%s\n", myMgr.TimeStampMsecStr().c_str());
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::SysManager::ThreadLock Class Reference

Detailed Description

This mechanism is used to allow debugging in a multi-threaded application. It locks a gate to prevent commands from being issued, then waits for all outstanding commands to finish and receive their responses from the Nodes. It then locks the mutex to the current thread, preventing other threads from executing. Doing this prevents response timeouts from occurring while stepping through code.

Note

This instance should generally be created in a C++ code block prior to a placed breakpoint within the same scope. When stepping through code, leaving the scope of where the **ThreadLock** was taken will unlock the other threads; possibly resulting in a timeout.

It may also be advisable when debugging to turn off the **watchdog timers** for each Node.

Coding Example

```
// This example shows how you can use ThreadLocks in
// conjuction with breakpoints to debug your application.
void myFunction() {

<<<<<<  function code Here <<<<<<<<<<<<<<

// Insert this just before the Break point to lock
    other threads to prevent response timeouts
SysManager::ThreadLock lock;

<<<<  insert breakpoint Here <<<<<<<<

<<<<<< more function code Here <<<<<<<<<<<<<<
}
//Once the scope of the function is left, the
    ThreadLock is released allowing other Threads to
    function normally.
```

Public Member Functions

ThreadLock ()
Create a safe and simple locking mutex to block other threads from

execution.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::ValueAlert Class Reference

Detailed Description

This class object implements access to the alert register and the attentionable alert mask within the node.

The motor's firmware and hardware contains many mechanisms to protect the motor and the application from problems that may occur. When an operation-blocking condition occurs, it is known as an *Alert*, or sometimes called a *Shutdown*. The shutdown is logged in the non-volatile memory with a time-stamp for review with the ClearView application. These alerts are stored locally within the node in a ninety-six bit **alertReg** struct.

The Attentionable alert mask is the mask which determines what alert bits will show in the User Alert field of the **mnStatusReg**.

The following is an example

See also

sFnd::IStatus::Alerts

sFnd::IAttnNode::AlertMask

Attentions/Interupts for Event Driven Applications **Coding Example**

```
//This example code will check to see if a node
    has any current alerts and
//print out the state string of any alerts found

//This creates an Empty String for us to put our
    Alert State String into
char alertList[256];

// get a reference to our ValueAlert Object
ValueAlert &myNodesAlert = myNode.Status.Alerts;

// Refresh the Real-Time Status, and Alert Register
theNode.Status.RT.Refresh();
myNodesAlert.Refresh();


// Check the Real-Time Status Register's
    "AlertPresent" bit
```

```
// The bit is set true if there are alerts in the
     alert register
if (theNode.Status.RT.Value().cpm.AlertPresent)
{
    printf("Node has alerts!\n");
}
else
{
    printf("Node has no alerts!\n");
}

// get an alertReg reference, and check directly
     for alerts
if (myNodesAlert.Value().isInAlert())
{
    // Put a text description of all alert bits
     set into our empty string, then print the string
    myNodesAlerts.Value().StateStr(alertList, 256);
    printf("Node has alerts! Alerts: %s", alertList);

// Check for specific alerts by accessing the
     cpmAlertFlds within the alertReg Struct
    if (myNodesAlert.Value().cpm.TrackingShutdown)
    {
        printf("Tracking Shutdown\n");
    }
}
```

_cpmAlertFlds for a complete list of Possible Alerts

Inherits **sFnd::ValueBase**.

Public Member Functions

void	**Refresh** () Query the node for a new copy of the underlying alert register. More...
alertReg &	**operator=** (**alertReg** &newValue) Assignment operator to update value with **alertReg** union value. More...

	operator alertReg () Copy operator to return a copy of the current alert type register. More...
alertReg	**Value** () Return a copy of the last acquired Alert Type Register. More...
void	**Value** (const **alertReg** &newValue) Set the *Alert Register* type register with the new value. More...
alertReg	**Last** () Return a copy of the prior acquired value. More...

Public Member Functions inherited from sFnd::ValueBase

bool	**Exists** () Determine the parameter's existence. More...
void	**AutoRefresh** (bool newState) Adjust the auto-refresh setting. More...
bool	**AutoRefresh** () Return the auto-refresh state. More...
bool	**IsVolatile** () Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

alertReg sFnd::ValueAlert::Last ( ) inline

Return a copy of the prior acquired value.

sFnd::ValueAlert::operator alertReg ( )

Copy operator to return a copy of the current alert type register.

alertReg& sFnd::ValueAlert::operator= (alertReg & newValue)

Assignment operator to update value with **alertReg** union value. This function should only be used when setting up the Attentionable Alert mask for attentions.

/see **sFnd::IAttnNode::AlertMask**

Coding Example

```
//Create a Alert register mask to set the
    Attentionable Alert Mask to.
//Here we are setting up attentions for a tracking
    error shutdown, a RMS Shutdown,
// and a soft limit shutdown
alertReg mymask;
mymask.cpm.TrackingShutdown = 1;
mymask.cpm.RMSOverloadShutdown = 1;
mymask.cpm.SoftLimitsExceeded = 1;

//Set the Node's Attentionable Status Mask to our mask.
mynode.Adv.Attn.AlertMask = mymask;
```

void sFnd::ValueAlert::Refresh ( ) virtual

This function will update the stored value to reflect current state of the node's Alert Register.

Implements **sFnd::ValueBase**.

Examples:
Example-StatusAlerts/Example-StatusAlerts.cpp.

alertReg sFnd::ValueAlert::Value ( )

Copy operator to return a copy of the current alert type register.

Examples:
Example-StatusAlerts/Example-StatusAlerts.cpp.

void sFnd::ValueAlert::Value (const alertReg & newValue)

Parameters
[in] **newValue** The value to update.

Update the internal register value within the Node with newValue. This function should only be used when setting up the Attentionable Alert mask for attentions.

/see **sFnd::IAttnNode::AlertMask**

Coding Example

```
//Create a Alert register mask to set the
    Attentionable Alert Mask to.
//Here we are setting up attentions for a tracking
    error shutdown, a RMS Shutdown,
// and a soft limit shutdown
alertReg mymask;
mymask.cpm.TrackingShutdown = 1;
mymask.cpm.RMSOverloadShutdown = 1;
mymask.cpm.SoftLimitsExceeded = 1;

//Set the Node's Attentionable Status Mask to our mask.
mynode.Adv.Attn.AlertMask = mymask;
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

sFoundation 2.0

List of all members | Public Member Functions

sFnd::ValueBase Class Reference abstract

Detailed Description

This class defines the base parameter access object with functions to determine the existence, volatility, and refresh parameters from a node.

This object provides your application with:

- A test for parameter existence for your node
- A test for volatile type parameters
- A Refresh control for volatile parameters such as position

Inherits **sFnd::IObjWithNode**.

Inherited by **sFnd::ValueAlert**, **sFnd::ValueDouble**, **sFnd::ValuePowerReg**, **sFnd::ValueSigned**, **sFnd::ValueStatus**, **sFnd::ValueString**, and **sFnd::ValueUnsigned**.

Public Member Functions

virtual void	**Refresh** ()=0 Update our local copy from the node. More...
bool	**Exists** () Determine the parameter's existence. More...
void	**AutoRefresh** (bool newState) Adjust the auto-refresh setting. More...
bool	**AutoRefresh** () Return the auto-refresh state. More...
bool	**IsVolatile** () Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

void sFnd::ValueBase::AutoRefresh (bool newState) inline

Quite often a application needs to get a snapshot of a parameter for use during a loop through the application logic. If you don't care about "sample-time" consistency, you can save calling the *Refresh* function before each access by setting this to true.

The default is false for this setting. This will cause one refresh on the first access.

Note

For performance reasons, don't set AutoRefresh on static type parameters, such as Acceleration Limits. Setting this true will cause network traffic on each inquiry to the value.

Parameters

[in]	**newState**	Set true to force a refresh on each value access. This should only be used for dynamic parameters such as position.

bool sFnd::ValueBase::AutoRefresh ( ) inline

This feature controls when queries from a node occur. If true is returned, each inquiry via the Value or type conversion functions will result in network traffic to query the value from the node.

If false is returned, only the first inquiry will transfer data from the node.

Returns

Current auto-refresh state.

bool sFnd::ValueBase::Exists ( ) inline

Determine the parameter's existence.

This is a helpful function when a mixture of node types may be in a system an some models may not support certain parameters.

Returns
True if the underlying parameter exists for this node.

bool sFnd::ValueBase::IsVolatile ( ) inline

Determine if this parameter is a dynamic parameter, such as status.

Returns
True if the value of this parameter can change after refresh.

virtual void sFnd::ValueBase::Refresh ( ) pure virtual

Use this function to update the local cached copy of the node's parameter. For most static type parameters, there is little need to call this member function as the first access will perform this operation automatically.

For dynamic parameters, such as position, status and velocity, use this function to obtain a fresh copy from the node.

See also
AutoRefresh(bool) for changing the behavior to refresh on each access.

Implemented in **sFnd::ValuePowerReg**, **sFnd::ValueAlert**, **sFnd::ValueStatus**, **sFnd::ValueString**, **sFnd::ValueUnsigned**, **sFnd::ValueSigned**, and **sFnd::ValueDouble**.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::ValueDouble Class Reference

Detailed Description

For most applications the following code sequence samples illustrate the commonly used functions.

Coding Example

```
// Setting a value
myNode.someFeature.someParameter = 12345;

// Getting a value
double myVal = myNode.someFeature.someParameter;

// Using a parameter in a calculation
double myCalVal = myNode.someFeature.someParameter.Value() + 1000;

// Making a copy of a value. This copy will run independently of the
// node's reference. For example, myCopy.Refresh() will only
// update myCopy's internal value.
ValueDouble myCopy = myNode.someFeature.someParameter;

// Making a short-cut reference. This sometimes can reduce the typing
// requirements for using a parameter. In this example, myRef for
// all intents and purposes is the same as myNode.aFeature.aParameter.
ValueDouble &myRef = myNode.someFeature.someParameter;
myRef.Refresh();
// is the same as
myNode.someFeature.someParameter.Refresh();
```

See also

Value Parameter Overview for more information on ValueXXXX Classes

Inherits **sFnd::ValueBase**.

Public Member Functions

void	**Refresh** () Update our local copy from the node. More...
double	**operator=** (double newValue) Set new double value using the assignment operator. More...
	operator double () Get the current value and ensure it is represented as a double. More...
	operator int32_t () Get the current value and ensure it is represented as a signed integer. More...
	operator uint32_t ()

Get the current value and ensure it is an unsigned integer. More...

void **Value** (int32_t newValue, bool makeNonVolatile=false)
Change our run-time parameter and optionally update the non-volatile default power-up value. More...

void **Value** (double newValue, bool makeNonVolatile=false)
Change our run-time parameter and optionally update the non-volatile default power-up value. More...

double **Value** (bool getNonVolatile=false)
Return the parameter's current value as a double. More...

Public Member Functions inherited from **sFnd::ValueBase**

bool **Exists** ()
Determine the parameter's existence. More...

void **AutoRefresh** (bool newState)
Adjust the auto-refresh setting. More...

bool **AutoRefresh** ()
Return the auto-refresh state. More...

bool **IsVolatile** ()
Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from **sFnd::IObjWithNode**

INode & **Node** ()
Reference to our node object. More...

Public Member Functions inherited from **sFnd::IObj**

virtual bool **Supported** ()
Query if this feature is supported in this node. More...

Member Function Documentation

sFnd::ValueDouble::operator double ( )

Get the current value and ensure it is represented as a double.

Coding Example

```
// Implicit conversion
double myDouble = myNode.someFeature.someParameter;

// Explicit conversion
printf("%f", double(myNode.someFeature.someParameter));
```

sFnd::ValueDouble::operator int32_t ( )

Get the current value and ensure it is represented as a signed integer.

Note
If the parameter is larger then +/- 31-bits the value will be truncated.

Coding Example

```
// Implicit conversion
int32_t myInt = myNode.someFeature.someParameter;

// Explicit conversion
printf("%d", int32_t(myNode.someFeature.someParameter));
```

sFnd::ValueDouble::operator uint32_t ( )

Get the current value and ensure it is an unsigned integer.

Note
If the parameter is larger then 32-bits the value will be truncated.

Coding Example

```
// Implicit conversion
uint32_t myInt = myNode.someFeature.someParameter;

// Explicit conversion
printf("%u", uint32_t(myNode.someFeature.someParameter));
```

double sFnd::ValueDouble::operator= (double newValue)

Set new double value using the assignment operator.

Parameters

[in] **newValue** New value to send to the node.

Coding Example

```
myNode.someFeature.someParameter = 1000;
```

See also

sFnd::ValueDouble::Value For explicit function way of setting a parameter as well as it's power-on default value.

void sFnd::ValueDouble::Refresh ( ) virtual

Use this function to update the local cached copy of the node's parameter. For most static type parameters, there is little need to call this member function as the first access will perform this operation automatically.

For dynamic parameters, such as position, status and velocity, use this function to obtain a fresh copy from the node.

See also

AutoRefresh(bool) for changing the behavior to refresh on each access.

Implements **sFnd::ValueBase**.

Examples:

Example-Homing/Example-Homing.cpp, **Example-MultiThreaded/Axis.h**, and **Example-SingleThreaded/Axis.h**.

void sFnd::ValueDouble::Value (int32_t newValue, bool makeNonVolatile = false)

Change our run-time parameter and optionally update the non-volatile default power-up value.

Parameters

[in] **newValue** New value to send to the node.

[in] **makeNonVolatile** [Optionally update the non-volatile power on default as well]

Coding Example

```
// Set the run-time value only
myNode.someFeature.someParameter.Value(1000);
// Set the run-time value and non-volatile power-on default
myNode.someFeature.someParameter.Value(1000, true);
```

Note

Care should be taken to not excessively write the non-volatile power-up default parameter as the memory has a write count limitation.

See also

sFnd::ValueDouble::operator= For a simpler syntax for setting a parameter.

Examples:

Example-Homing/Example-Homing.cpp, **Example-Motion/Example-Motion.cpp**, and **Example-MultiThreaded/Axis.h**.

void sFnd::ValueDouble::Value (double newValue, bool makeNonVolatile = `false`)

Change our run-time parameter and optionally update the non-volatile default power-up value.

Parameters

[in]	**newValue**	New value to send to the node.
[in]	**makeNonVolatile**	[Optionally update the non-volatile power on default as well]

Coding Example

```
// Set the run-time value only
myNode.someFeature.someParameter.Value(1000.5);
// Set the run-time value and non-volatile power-on default
myNode.someFeature.someParameter.Value(1000.5, true);
```

Note

Care should be taken to not excessively write the non-volatile power-up default parameter as the memory has a write count limitation.

See also

sFnd::ValueDouble::operator= For simplified syntax to write the run-time value.

double sFnd::ValueDouble::Value (bool getNonVolatile = `false`)

Return the parameter's current value as a double.

Parameters

[in]	**getNonVolatile**	If true, the power-on default non-volatile value is returned.

Coding Example

```
// Get and print the run-time value
cout << myNode.someFeature.someParameter.Value() << endl;
// Get and print the power-on default non-volatile value
cout << myNode.someFeature.someParameter.Value(true) << endl;
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::ValuePowerReg Class Reference

Detailed Description

This class object implements access to the power state register.

The motor's firmware monitors the power connections continuously. This register shows the currently detected connections and any anomalies detected during operations. These include loose phase wiring and which terminals the power is attached to.

The following is an example

```
//This example code will check and return the
    current power conditions and
//print out the state string of any issues found.

//This creates an Empty String for us to put our
    state string into
char pwrInfo[256];

// get a reference to our ValuePowerReg Object
ValuePowerReg &myPower = myNode.Status.Power;

// Refresh the Real-Time Status, and Alert Register
theNode.Status.Power.Refresh();
myPower.Refresh();


// Check that there has been no disruptions in
    the AC supply.
// The bit is set true if there are alerts in
    the alert register
if (theNode.Status.Power.Value().fld.InBusLoss)
{
printf("Node's power bus has been removed!\n");
}
else
```

```
{
printf("Node has no bus supply problems!\n");
}

// Put a text description of all alert bits set
    into our empty string, then print the string
myNodesAlerts.Value().StateStr(pwrInfo, 256);
printf("Node power state: %s", pwrInfo);
```

See also

_mnPowerFlds for a complete list of Possible Alerts

Inherits **sFnd::ValueBase**.

Public Member Functions

void	**Refresh** () Query the node for a new copy of the underlying alert register. More...
	operator mnPowerReg () Copy operator to return a copy of the current power type register. More...
mnPowerReg	**Value** () Return a copy of the last acquired power Type Register. More...
mnPowerReg	**Last** () Return a copy of the prior acquired value. More...

Public Member Functions inherited from sFnd::ValueBase

bool	**Exists** () Determine the parameter's existence. More...
void	**AutoRefresh** (bool newState)

	Adjust the auto-refresh setting. More...
bool	**AutoRefresh** () Return the auto-refresh state. More...
bool	**IsVolatile** () Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

mnPowerReg sFnd::ValuePowerReg::Last ( ) inline

Return a copy of the prior acquired value.

sFnd::ValuePowerReg::operator mnPowerReg ( )

Copy operator to return a copy of the current power type register.

void sFnd::ValuePowerReg::Refresh ( ) virtual

This function will update the stored value to reflect current state of the node's Alert Register.

Implements **sFnd::ValueBase**.

mnPowerReg sFnd::ValuePowerReg::Value ( )

Copy operator to return a copy of the current power type register.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::ValueSigned Class Reference

Detailed Description

This class object implements access to integer parameters within the node.

For most applications the following code sequence samples illustrate the commonly used functions.

Coding Example

```
// Setting a value
myNode.someFeature.someParameter = 12345;

// Getting a value
int32_t myVal = myNode.someFeature.someParameter;

// Using a parameter in a calculation
int32_t myCalVal = myNode.someFeature.someParameter.Value() + 1000;

// Making a copy of a value. This copy will run independently of the
// node's reference. For example, myCopy.Refresh() will only
// update myCopy's internal value.
ValueSigned myCopy = myNode.someFeature.someParameter;

// Making a short-cut reference. This sometimes can reduce the typing
// requirements for using a parameter. In this example, aRef for
// all intents and purposes is the same as myNode.aFeature.aParameter.
ValueSigned &myRef = myNode.someFeature.someParameter;
myRef.Refresh();
// is the same as
myNode.someFeature.someParameter.Refresh();
```

See also

Value Parameter Overview for more information on ValueXXXX classes.

Inherits **sFnd::ValueBase**.

Public Member Functions

void	**Refresh** () Query the node and obtain a new copy of the parameter. More...
int32_t	**operator=** (int32_t newValue) Set new signed value using the assignment operator. More...
	operator double () Get the current value and ensure it is represented as a double. More...
	operator int32_t () Get the current value and ensure it is represented as a signed integer. More...

	operator uint32_t () Get the current value and ensure it is an unsigned integer. More...
void	**Value** (int32_t newValue, bool makeNonVolatile=false) Change our integer based parameter and optionally update the non-volatile default value. More...
int32_t	**Value** (bool getNonVolatile=false) Get the current parameter value as an integer. More...

Public Member Functions inherited from sFnd::ValueBase

bool	**Exists** () Determine the parameter's existence. More...
void	**AutoRefresh** (bool newState) Adjust the auto-refresh setting. More...
bool	**AutoRefresh** () Return the auto-refresh state. More...
bool	**IsVolatile** () Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

sFnd::ValueSigned::operator double ( )

Get the current parameter as double.

Coding Example

```
// Implicit conversion
double someDouble = myNode.someFeature.someParameter;
or
// Explicit conversion
printf("%f", double(myNode.someFeature.someParameter));
```

sFnd::ValueSigned::operator int32_t ( )

Get the current parameter as a signed integer.

Note
If the parameter is larger then +/- 31-bits the value will be truncated.

Coding Example

```
// Implicit conversion
int32_t someInt = myNode.someFeature.someParameter;
or
// Explicit conversion
printf("%d", int32_t(myNode.someFeature.someParameter));
```

sFnd::ValueSigned::operator uint32_t ( )

Get the current parameter as an unsigned integer.

Note
If the parameter is larger then 32-bits the value will be truncated.

Coding Example

```
// Implicit conversion
uint32_t someInt = myNode.someFeature.someParameter;
or
// Explicit conversion
printf("%u", uint32_t(myNode.someFeature.someParameter));
```

int32_t sFnd::ValueSigned::operator= (int32_t newValue)

Set new signed value using the assignment operator.

This is the most common way to change an integer based parameter.

Coding Example

```
myNode.someFeature.someParameter = 1000;
```

See also

sFnd::ValueSigned::Value For an explicit way of setting a parameter as well as it's power-on default value.

void sFnd::ValueSigned::Refresh ( ) virtual

Query the node and obtain a new copy of the parameter.

Coding Example

```
//Refresh a volatile parameter before reading
mynode.someFeature.someParameter.Refresh();
```

Implements **sFnd::ValueBase**.

void sFnd::ValueSigned::Value (int32_t newValue, bool makeNonVolatile = false)

Parameters

[in]	**newValue**	New value to send to the node.
[in]	**makeNonVolatile**	[Optionally update the non-volatile power-on default as well]

Coding Example

```
// Set Value and non-volatile power-on default
myNode.someFeature.someParameter.Value(-1000, true);
// Set Value only
myNode.someFeature.someParameter.Value(-1000);
```

Note

Care should be taken to not excessively write the non-volatile parameter as the memory has a write count limitation.

See also

sFnd::ValueSigned::operator= For simplified syntax to write the run-time value.

int32_t sFnd::ValueSigned::Value (bool getNonVolatile = false)

Parameters

[in] **getNonVolatile** If set, the value returned will be the power-on non-volatile value.

Coding Example

```
// Get and print the run-time value
cout << myNode.Limits.SoftLimit0.Value() << endl;
// Get and print the power-on default non-volatile value
cout << myNode.Limits.SoftLimit0.Value(true) << endl;
```

Note

Care should be taken to not excessively write the non-volatile parameter as the memory has a write count limitation.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::ValueStatus Class Reference

Detailed Description

This class object implements access to the various types of status register/masks within the node.

Different Types of Node status registers are useful for determining whether a particular status field is currently asserted, has been asserted since the last read, or has been cleared since the last read. The different types of Status Registers are:

- **Real-Time** Which represents the current state of the Status Register.
- **Rising** Which tracks rising edges of status fields. Fields in this register are set when the **Real-Time** fields transition from 0 to 1.
- **Falling** Which tracks falling edges of status fields. Fields in this register are set when the **Real-Time** fields transition from 1 to 0.
- **Accumulating** Which represents the Or accumulation of the **Real-Time** register.

These registers are internally clear-on-read within the Node. This object provides a caching mechanism allowing the application to control when the stored value of this register is cleared/refreshed. This allows multiple threads which are competing for different Status event to coexist without interfering with each other.

This object also provides access to Status masks used to generate attentions.

See also

sFnd::IStatus::RT

sFnd::IStatus::Rise

sFnd::IStatus::Fall

sFnd::IStatus::Accum

cpmStatusRegFlds for a complete list of ClearPath-SC Status Fields

Attentions/Interupts for Event Driven Applications

Inherits **sFnd::ValueBase**.

Public Member Functions

void	**Refresh** () Update our Status Register from the Node. More...
bool	**TestAndClear** (const **mnStatusReg** &mask, **mnStatusReg** &result) Read, Test and Clear Accumulated *Status Register* state. More...
bool	**TestAndClear** (const **mnStatusReg** &mask) Read and simple test and clear for single fields. More...
void	**Clear** () Clear the accumulated state. More...
mnStatusReg &	**operator=** (**mnStatusReg** &newValue) Assignment operator to update value with **mnStatusReg** union value. More...
	operator mnStatusReg () Copy operator to return a copy of the current status type register. More...
mnStatusReg	**Value** () Return a copy of the last acquired Status Type Register. More...
void	**Value** (const **mnStatusReg** &newValue) Set the *Status Register* type register with the new value. More...
mnStatusReg	**Last** () Return a copy of the last acquired value. More...

Public Member Functions inherited from sFnd::ValueBase

bool	**Exists** () Determine the parameter's existence. More...
void	**AutoRefresh** (bool newState) Adjust the auto-refresh setting. More...
bool	**AutoRefresh** () Return the auto-refresh state. More...
bool	**IsVolatile** () Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

void sFnd::ValueStatus::Clear ( )

This function is used to clear out accumulated status information from the internal copy. This function is not needed for the Real-Time register.

Coding Example

```
//Clear, and refresh The Accumulating register
mynode.Status.Accum.Clear();
mynode.Status.Accum.Refresh();
```

mnStatusReg sFnd::ValueStatus::Last ( )

Return a copy of the last acquired value.

Returns
Copy of the last updated value.

sFnd::ValueStatus::operator mnStatusReg ( )

Copy operator to return a copy of the current status type register.

mnStatusReg& sFnd::ValueStatus::operator= (mnStatusReg & newValue)

Assignment operator to update value with **mnStatusReg** union value. This function should only be used when setting up Status masks for attentions.

See also

sFnd::IAttnNode::Mask

sFnd::IAttnNode::StatusMask

Coding Example

```
//Create a Status register mask to set the
    Attentionable Status Mask to.
//Here we are setting up attentions for Move
    done, A from start, and B from end
// Only the lower 32 bits are attentionable.
mnStatusReg mymask;
mymask.cpm.MoveDone = 1;
mymask.cpm.BFromEnd = 1;
mymask.cpm.AFromStart = 1;

//Set the Node's Attentionable Status Mask to
    our mask.
mynode.Adv.Attn.Mask = mymask;
```

void sFnd::ValueStatus::Refresh ( ) virtual

This function will atomically read and accumulate Rising, Falling, and Accumulating registers to prevent the loss of state information when accessed by multiple threads. This means refreshing these registers will not clear any currently asserted bits. Use one of the **TestAndClear** functions to combine with testing to prevent possible information loss from clearing the entire register at once.

For the Real-Time register this function simply replaces the value of the stored register to the current status register within the Node.

Coding Example

```
//Refresh The Accumulating register
mynode.Status.Accum.Clear();        //Clear the
    accumulating register so we only capture
    accumulated events since the last read
```

```
mynode.Status.Accum.Refresh();      //Refresh
     the register

//Refresh the Register (No clear is needed
     since the Real Time register simply
     overwrites the REgister to be the current value
mynode.Status.RT.Refresh();
```

Implements **sFnd::ValueBase**.

Examples:
Example-StatusAlerts/Example-StatusAlerts.cpp.

bool sFnd::ValueStatus::TestAndClear (const mnStatusReg & mask, mnStatusReg & result)

Parameters

[in] **mask** fields to test for.
[out] **result** fields that were cleared.

Returns

true if one or more were true.

This function is a thread safe way of updating and atomically testing and clearing fields from the accumulating *Status Registers* ()Rise, Fall, Accum). The *result* is updated with all the fields detected by the mask. This is useful when testing multiple fields. This function is not needed with the Real-Time register.

Coding Example

```
//Create a Status register mask to test the
     Accumulating register
//Test for move done, A from start, and B from end.
mnStatusReg mymask;
```

```
mymask.cpm.MoveDone = 1;
mymask.cpm.BFromEnd = 1;
mymask.cpm.AFromStart = 1;

//Test and clear these three fields
mnStatusReg myResult;
if(mynode.Status.Accum(mymask,myResult))
{
    if(myResult.cpm.MoveDone)
    { Move Done occured}
    if(myResult.cpm.BFromEnd)
    { B from end occured}
    if(myResult.cpm.AFromStart)
    { A from start occured}
}
//At this point the three fields are now clear
    in our stored copy of the Accum register
    Refresh to find more events
```

bool sFnd::ValueStatus::TestAndClear (const mnStatusReg & mask)

Parameters

[in] **mask** *Status Register* union with the desired field to test and clear.

Returns

True if the desired field was set.

This function is a thread safe way of updating and atomically testing and clearing a field from the accumulating *Status Registers* ()Rise, Fall, Accum). This function is not needed with the Real-Time register.

Note

You should only use this function to test a single field. You cannot determine which field was set if multiple fields are set in the mask.

See also

sFnd::ValueStatus::TestAndClear(const mnStatusReg &, mnStatusReg &) for multiple field test and clear.

Coding Example

```
//Create a Status register mask to test the
    Accumulating register
//Test for move done
mnStatusReg mymask;
mymask.cpm.MoveDone = 1;

//Test and clear the move done field
if(mynode.Status.Accum(mymask))
{Move Done occured}

//At this point the three fields are now clear
    in our stored copy of the Accum register
    Refresh to find more events
```

mnStatusReg sFnd::ValueStatus::Value ( )

Return a copy of the last acquired Status Type Register.

Examples:

Example-GPIO/Example-GPIO.cpp, and Example-StatusAlerts/Example-StatusAlerts.cpp.

void sFnd::ValueStatus::Value (const mnStatusReg & newValue)

Update the internal register value within the Node with newValue. This function should only be used when setting up Status masks for attentions.

See also

sFnd::IAttnNode::Mask

sFnd::IAttnNode::StatusMask

Coding Example

```
//Create a Status register mask to set the
    Attentionable Status Mask to.
//Here we are setting up attentions for Move
    done, A from start, and B from end
// Only the lower 32 bits are attentionable.
mnStatusReg mymask;
mymask.cpm.MoveDone = 1;
mymask.cpm.BFromEnd = 1;
mymask.cpm.AFromStart = 1;

//Set the Node's Attentionable Status Mask to
    our mask.
mynode.Adv.Attn.Mask.Value(mymask);
```

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

List of all members | Public Member Functions

sFnd::ValueString Class Reference abstract

Detailed Description

This class object implements access to various string oriented parameters within the node.

For most applications the following code sequence samples illustrate the commonly used functions.

Coding Example

```
// Setting a value
myNode.someFeature.someParameter = 12345;

// Getting a value
String myVal = myNode.someFeature.someParameter;

// Using a parameter in a calculation
String myCalVal = myNode.someFeature.someParameter.Value() + 1000;

// Making a copy of a value. This copy will run independently of the
// node's reference. For example, myCopy.Refresh() will only
// update myCopy's internal value.
ValueString myCopy = myNode.someFeature.someParameter;

// Making a short-cut reference. This sometimes can reduce the typing
// requirements for using a parameter. In this example, aRef for
// all intents and purposes is the same as myNode.aFeature.aParameter.
ValueString &myRef = myNode.someFeature.someParameter;
myRef.Refresh();
// is the same as
myNode.someFeature.someParameter.Refresh();
```

See also

Value Parameter Overview for more information on ValueXXXX classes.

Inherits **sFnd::ValueBase**.

Public Member Functions

virtual void	**Refresh** ()=0 Query the node and obtain a new copy of the parameter. More...
virtual void	**Value** (const char *newValue)=0 Update the string parameter. More...
virtual const char *	**Value** ()=0 Get the current string parameter. More...
const char *	**operator=** (const char *newValue) Update the string value using the assignment operator. More...

	operator const char * () Get a Pointer to the Current String Value. More...

Public Member Functions inherited from sFnd::ValueBase

bool	**Exists** () Determine the parameter's existence. More...
void	**AutoRefresh** (bool newState) Adjust the auto-refresh setting. More...
bool	**AutoRefresh** () Return the auto-refresh state. More...
bool	**IsVolatile** () Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode &	**Node** () Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool	**Supported** () Query if this feature is supported in this node. More...

Member Function Documentation

sFnd::ValueString::operator const char * ( )

Get a pointer to the read-only current string value.

Coding Example

```
const char *myParam = myNode.Info.UserID;
```

const char* sFnd::ValueString::operator= (const char * newValue)

Parameters

[in] **newValue** Incoming string pointer.

This is the most common way to change a string based parameter.

Coding Example

```
myNode.Info.UserID = "My Node";
```

virtual void sFnd::ValueString::Refresh ( ) pure virtual

Query the node and obtain a new copy of the parameter.

Coding Example

```
//Refresh a volatile parameter before reading
mynode.someFeature.someParameter.Refresh();
```

Implements **sFnd::ValueBase**.

virtual void sFnd::ValueString::Value (const char * newValue) pure virtual

Parameters

[in] **newValue** New C-String to use for update.

Update the string parameter with *newValue*.

Coding Example

```
myNode.Info.UserID.Value("My Node");
```

Examples:

Example-Homing/Example-Homing.cpp, Example-Motion/Example-Motion.cpp, and Example-MultiThreaded/Axis.h.

virtual const char* sFnd::ValueString::Value ( ) pure virtual

Returns
The current string parameter.

Return a pointer to a read-only buffer that contains the current value.

Coding Example

```
cout << myNode.Info.UserID.Value() << endl;
```

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

- C:/Git/inc/inc-pub/pubSysCls.h

List of all members | Public Member Functions

sFnd::ValueUnsigned Class Reference

Detailed Description

This class object implements access to unsigned integer parameters within the node.

For most applications the following code sequence samples illustrate the commonly used functions.

Coding Example

```
// Setting a value
myNode.someFeature.someParameter = 12345;

// Getting a value
uint32_t myVal = myNode.someFeature.someParameter;

// Using a parameter in a calculation
uint32_t myCalVal = myNode.someFeature.someParameter.Value() + 1000;

// Making a copy of a value. This copy will run independently of the
// node's reference. For example, myCopy.Refresh() will only
// update myCopy's internal value.
ValueUnsigned myCopy = myNode.someFeature.someParameter;

// Making a short-cut reference. This sometimes can reduce the typing
// requirements for using a parameter. In this example, aRef for
// all intents and purposes is the same as myNode.aFeature.aParameter.
ValueUnsigned &myRef = myNode.someFeature.someParameter;
myRef.Refresh();
// is the same as
myNode.someFeature.someParameter.Refresh();
```

See also

Value Parameter Overview for more information on ValueXXXX classes.

Inherits **sFnd::ValueBase**.

Public Member Functions

void	**Refresh** () Query the node and obtain a new copy of the parameter. More...
uint32_t	**operator=** (uint32_t newValue) Set a new unsigned value using the assignment operator. More...
	operator double () Get the current parameter and ensure it is represented as a double. More...
	operator uint32_t () Get the current value and ensure it is an integer. More...

void **Value** (uint32_t newValue, bool makeNonVolatile=false)
Change our integer based parameter and optionally update the non-volatile default value. More...

uint32_t **Value** (bool getNonVolatile=false)
Get the current parameter value as an integer. More...

Public Member Functions inherited from sFnd::ValueBase

bool **Exists** ()
Determine the parameter's existence. More...

void **AutoRefresh** (bool newState)
Adjust the auto-refresh setting. More...

bool **AutoRefresh** ()
Return the auto-refresh state. More...

bool **IsVolatile** ()
Determine if this parameter is a dynamic parameter. More...

Public Member Functions inherited from sFnd::IObjWithNode

INode & **Node** ()
Reference to our node object. More...

Public Member Functions inherited from sFnd::IObj

virtual bool **Supported** ()
Query if this feature is supported in this node. More...

Member Function Documentation

sFnd::ValueUnsigned::operator double ( )

Get the current parameter and format it as a double.

Coding Example

```
// Implicit conversion
double someDouble = myNode.someFeature.someParameter;
or
// Explicit conversion
printf("%f", double(myNode.someFeature.someParameter));
```

sFnd::ValueUnsigned::operator uint32_t ( )

Get the current value and ensure it is an integer.

Note
If the parameter is larger then 31-bits the value will be truncated or undefined.

Coding Example

```
// Implicit conversion
uint32_t someInt = myNode.someFeature.someParameter;
or
// Explicit conversion
printf("%u", int32_t(myNode.someFeature.someParameter));
```

uint32_t sFnd::ValueUnsigned::operator= (uint32_t newValue)

This is the most common way to change an integer based parameter.

Coding Example

```
myNode.someFeature.someParameter = 1000;
```

See also
sFnd::ValueSigned::Value For an explicit way of setting a parameter as well as it's power-on default value.

void sFnd::ValueUnsigned::Refresh ( ) virtual

Query the node and obtain a new copy of the parameter.

Coding Example

```
//Refresh a volatile parameter before reading
mynode.someFeature.someParameter.Refresh();
```

Implements **sFnd::ValueBase**.

void sFnd::ValueUnsigned::Value (uint32_t newValue, bool makeNonVolatile = `false`)

Change our integer based parameter and optionally update the non-volatile default value.

Parameters

- [in] **newValue** New value to send to the node.
- [in] **makeNonVolatile** [Optionally update the non-volatile power-on default as well]

Coding Example

```
// Set Value and non-volatile power-on default
myNode.someFeature.someParameter.Value(2000, true);
// Set Value only
myNode.someFeature.someParameter.Value(2000);
```

Note

Care should be taken to not excessively write the non-volatile parameter as the memory has a write count limitation.

See also

sFnd::ValueSigned::operator= For simplified syntax to write the run-time value.

Examples:

Example-Homing/Example-Homing.cpp, **Example-Motion/Example-Motion.cpp**, and **Example-MultiThreaded/Axis.h**.

uint32_t sFnd::ValueUnsigned::Value (bool getNonVolatile = `false`)

Get the current parameter value as an integer.

Parameters

- [in] **getNonVolatile** If set, the value returned will be the power-on non-volatile value.

Coding Example

```
// Get and print the run-time value
cout << myNode.Limits.SoftLimit0.Value() << endl;
// Get and print the power-on default non-volatile value
cout << myNode.Limits.SoftLimit0.Value(true) << endl;
```

Note

Care should be taken to not excessively write the non-volatile parameter as the memory has a write count limitation.

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

- C:/Git/inc/inc-pub/**pubSysCls.h**

_alertReg Union Reference

List of all members | Public Member Functions | Public Attributes

Detailed Description

The alert register defines the shutdown state of the node. When a node enters a shutdown or potentially non-operable condition, the node will set bits in this register to tell the application what went wrong. Alerts range from serious to minor.

- Hardware/Software failures during power on self testing and internal consistency checks that occur during operation. These include items such as as:
  - Sensor failures
  - Component failures
  - Sensors drifting out of their nominal ranges
  - Non-volatile configuration errors
- Self preservation shutdowns where the node shuts down to protect itself from being damaged.
  - Over motor bus voltage
  - Shorted circuits
  - Excessive power delivery to motor
  - Jams
- Application errors such as:
  - Parameters out-of-range for requested operations such as commanding a node past it's soft limits

Most shutdowns can be cleared via the **sFnd::IStatus::AlertsClear** function to allow continued node operations, however fatal errors cannot be cleared by this mechanism and require a power-cycle or node restart to clear them.

Note

See the specific bit for potential recovery actions to allow node operations to continue. If the Status Register's **HwFailure** bit is asserted, this is a hardware failure and recovery is probably not possible.

See also

_cpmAlertFlds For the complete list of ClearPath-SC alert Fields.

sFnd::IStatus::Alerts for access to this register.

Public Member Functions

bool	**isClear** () Test if all bits are non-zero. More...
bool	**isInAlert** () Test if any bits are non-zero. More...
char *	**StateStr** (char *buffer, size_t size) Update a string with the set fields in the Alert Register. More...

Public Attributes

Uint32	**bits** [3] "Parallel" view of register as three unsigned 32-bit integers.
cpmAlertFlds	**cpm** Access to ClearPath-SC specific alert fields.

Member Function Documentation

bool _alertReg::isClear ( ) inline

Returns true if all fields in the register are zero.

Coding Example

```
// get an alertReg reference, and check
     directly for alerts
if (myNode.Status.Alerts.Value().isClear())
{
printf("Node has no alerts!");
}
```

bool _alertReg::isInAlert ( ) inline

Returns true if any fields in the register is asserted.

Coding Example

```
// get an alertReg reference, and check
     directly for alerts
if (myNode.Status.Alerts.Value().isInAlert())
{
printf("Node has alerts!");
}
```

char* _alertReg::StateStr (char * buffer, size_t size) inline

Parameters

[in,out]	**buffer**	Point to string buffer to update.
[in]	**size**	The size of the buffer in characters including the null character.

Returns

the parameter *buffer*.

Coding Example

```
char myBuffer[500];
cout <<
    myNode.Status.Alert.Value.StateStr(myBuffer,
    sizeof(myBuffer));
```

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

- C:/Git/inc/inc-pub/**pubCoreRegs.h**

List of all members | Public Member Functions | Public Attributes

_cpmAlertFlds Struct Reference

Detailed Description

This page provides the detailed list of the Clearpath SC alerts. These fields describe the reason a shutdown has occurred, and potential solutions.

See also

sFnd::ValueAlert For information on accessing alert registers within the Node

Clearpath SC Alerts

Alert Field	Bit #		Alert Field	Bit #
n/a	32		MtrPhaseOverload	64
n/a	33		n/a	65
n/a	34		MtrBadSetupParams	66
MoveGenRange	35		HardStopBrokeFree	67
n/a	36		n/a	68
n/a	37		TrackingShutdown	69
JrkLimVelRequestBad	38		n/a	70
n/a	39		RMSOverloadShutdown	71
PhaseSensorFailed	40		n/a	72
LimSwitchActivated	41		n/a	73
SoftLimitExceeded	42		n/a	74
n/a	43		n/a	75
n/a	44		NoCommSweepFailed	76
n/a	45		NoCommSweepReversed	77
n/a	46		NoCommFailed	78
n/a	47		n/a	79
n/a	48		TempAmbientHigh	80
n/a	49		n/a	81
n/a	50		BusOverCurrent	82
n/a	51		BusOverVoltage	83
n/a	52		BusVoltageLow	84
n/a	53		BusRMSOverload	85
n/a	54		n/a	86
n/a	55		n/a	87
n/a	56		n/a	88
n/a	57		NoCommBusLow	89
n/a	58		n/a	90
n/a	59		n/a	91
MtrEncGlitch	60		n/a	92
n/a	61		n/a	93
n/a	62		n/a	94
n/a	63		n/a	95

Public Member Functions

MN_EXPORT char *MN_DECL	**StateStr** (char *buffer, size_t size) const Create a new line delimited string of the alerts set. More...

Public Attributes

commonAlertFlds	**Common** Common shutdown events. Least significant bits 0-31.
BFld	**JrkLimVelRequestBad**: 1 Velocity limit too high for jerk limit. More...
BFld	**PhaseSensorFailed**: 1 Phase sensor bad. More...
BFld	**LimSwitchActivated**: 1 Limit switch activated. More...
BFld	**SoftLimitExceeded**: 1 Soft limit prevents move. More...
BFld	**ACPowerLost**: 1 AC Power Lost. More...
BFld	**ACPhaseLost**: 1 AC Phase Lost. More...
BFld	**MtrTempLow**: 1 Low motor temperature. More...
BFld	**MtrVectorBad**: 1 Vector Problem. More...

BFld	**MtrEncGlitch**: 1 Motor encoder glitch. More...
BFld	**MtrPhaseOverload**: 1 Motor phase overload. More...
BFld	**MtrBadSetupParams**: 1 Bad motor parameters. More...
BFld	**HardStopBrokeFree**: 1 Hardstop broke free. More...
BFld	**TrackingShutdown**: 1 Tracking shutdown. More...
BFld	**RMSOverloadShutdown**: 1 RMS torque overload shutdown. More...
BFld	**NoCommSweepFailed**: 1 Insufficient sensorless sweep distance. More...
BFld	**NoCommSweepReversed**: 1 Sensorless motion in wrong direction. More...
BFld	**NoCommFailed**: 1 Sensorless startup failed. More...
BFld	**DriveTempHigh**: 1 High Drive temperature. More...
BFld	**StatorTempHigh**: 1 Stator temperature. More...
BFld	**BusOverCurrent**: 1

	Motor's bus power supply overloaded. More...
BFld	**BusOverVoltage**: 1 Motor's bus power supply voltage too high. More...
BFld	**BusVoltageLow**: 1 Motor's bus power supply voltage too low. More...
BFld	**BusRMSOverload**: 1 Motor's bus power supply overloaded. More...
BFld	**MtrEncIndexMisplaced**: 1 MtrEncIndexMisplaced. More...
BFld	**StepsDuringPosnRecovery**: 1 StepsDuringPosnRecovery. More...

Member Function Documentation

MN_EXPORT char* MN_DECL _cpmAlertFlds::StateStr (char * **buffer**, size_t **size**) const

Create a new line delimited string of the alerts set

Parameters

[in,out]	**buffer**	Pointer to a string buffer to replace with the alert fields that are non-zero.
[in]	**size**	Count of characters in *buffer*.

Returns

The original pointer the buffer *buffer*. This allows this function to be the argument of printing statement or other formatting function.

Member Data Documentation

BFld _cpmAlertFlds::ACPhaseLost

Motor was running with AC power and one or more of the AC phases have been removed.

Remedies:

- Check AC power wiring and power quality to attached modules.

BFld _cpmAlertFlds::ACPowerLost

Motor was running with AC power and AC power has been removed.

Remedies:

- Check AC power wiring and power quality to attached modules.

BFld _cpmAlertFlds::BusOverCurrent

The motor's bus current has exceeded maximum thresholds.

Remedies:

- Increase the bus voltage.
- If condition still persists under non-stressful no-load conditions, replace the damaged motor.
- Check the mechanics for abnormally high friction/binding. Grease if necessary.

BFld _cpmAlertFlds::BusOverVoltage

The motor's bus voltage has exceeded maximum thresholds. This is recoverable when the bus voltage return to the operating range of the motor.

Remedies:

- Check that the power supply has been correctly configured for your AC voltage and is producing the expected voltage level.
- Check and/or install regenerated bus voltage management system.

BFld _cpmAlertFlds::BusRMSOverload

The motor's bus current has exceeded maximum continuous thresholds.

Remedies:

- Check that the power supply has been correctly configured for your AC voltage and is producing the expected voltage level.
- Check the mechanics for abnormally high friction/binding.
- Verify the configuration file in the motor is appropriate.
- Reduce motion duty cycle, acceleration, or load.

BFld _cpmAlertFlds::BusVoltageLow

The motor has detected a loss or brown-out of the motor's bus voltage. This condition will automatically clear upon restoration of the bus voltage without host intervention.

Remedies:

- Check that the power supply has been correctly configured for your AC voltage and is producing the expected voltage level.
- Check the mechanics for abnormally high friction/binding. Grease if necessary.

BFId _cpmAlertFIds::DriveTempHigh

The motor has detected an excessive operating temperature. This condition is recoverable when the temperature has been sufficiently reduced.

Remedies:

- Provide more ventilation/fan cooling to motor.

BFId _cpmAlertFIds::HardStopBrokeFree

The hardstop moved when pressure was applied by the motor. Once the motor has qualified for the hardstop condition, the mechanics must stay reasonably still while motor torque is applied at the foldback level. If the coupling or hardstop should break free, the axis will move in the direction of the hardstop and this alert will be issued

Remedies:

- Checking homing parameters in ClearView
- Ensure that the hardstop for this axis is solid and unmovable.
- For High friction applications, or machines subject to binding issues, ensure that the hardstop entry torque defined through ClearView is high enough that the motor will not qualify for the hardstop condition except when the axes is against the hardstop on the machine.

BFId _cpmAlertFIds::JrkLimVelRequestBad

The move was canceled due to the **Velocity Limit** of the attempted move exceeding the encoder's or RAS/G-Stop's maximum allowed speed.

Remedies:

- Reduce the **Velocity Limit** of the move.
- Decrease the RAS/G-Stop to one with less speed limiting(a lower time).

BFld _cpmAlertFlds::LimSwitchActivated

Attempting to move in a direction in which the limit switch is activated. Motion in the direction of the limit is prevented. The current move, and any queued moves, are canceled.

Remedies:

- Move away from the limit switch or clear the limit switch activation.

BFld _cpmAlertFlds::MtrBadSetupParams

A motor-specific parameter has an illegal value.

Remedies:

- Restore motor to factory defaults and reload a valid configuration file.
- If restoring to factory defaults doesn't clear the problem, replace motor.

BFld _cpmAlertFlds::MtrEncGlitch

A glitch was detected on the motor encoder signals.

Remedies:

- Replace motor.

BFId _cpmAlertFIds::MtrEncIndexMisplaced

The motor encoder index was seen when it was not expected.

Appears in: AlertReg, WarnReg

Remedies:

- Replace defective motor

BFId _cpmAlertFIds::MtrPhaseOverload

A motor phase has overloaded.

Remedies:

- Restore motor to factory defaults and reload a valid configuration file.
- Check the mechanics for abnormally high friction/binding. Grease if necessary.
- If problem persists under no-load conditions, replace motor.

BFId _cpmAlertFIds::MtrTempLow

The motor has detected an excessive operating temperature. This condition is recoverable when the temperature has been sufficiently reduced.

Remedies:

- Move the motor to a warmer environment.

BFId _cpmAlertFIds::MtrVectorBad

The motor's commutation vector has drifted. This will lead to poor performance. This is normally a warning indication.

Appears in: WarnReg

Remedies:

- Replace defective motor.

BFId _cpmAlertFIds::NoCommFailed

This shutdown occurs when the sensorless startup failed after too many retries.

Remedies:

- Verify that the configuration file is correct for the motor; reload the configuration file.
- Check that the power supply has been correctly configured for your AC voltage, and is producing the expected voltage level.
- Check the mechanics for abnormally high friction/binding.
- Check for excessive external forces acting on the axis.

BFId _cpmAlertFIds::NoCommSweepFailed

This shutdown occurs when a motor's sensorless start fails to move the appropriate distance.

Remedies:

- Check the mechanics for abnormally high friction/binding. Grease if necessary.
- Check for excessive external forces acting on the axis.

- Verify that the configuration file is correct for the motor; reload the configuration file.
- Check that the power supply has been correctly configured for your AC voltage, and is producing the expected voltage level.
- Check for the misapplication of any torque limiters such as the Global Torque Limit that might be limiting torque in the motor.

BFId _cpmAlertFIds::NoCommSweepReversed

This shutdown occurs when a motor's sensorless start detects the motor moving in the wrong direction.

Remedies:

- Verify that the configuration file is correct for the motor; reload the configuration file.
- Check for excessive external forces acting on the axis.
- Check that the power supply has been correctly configured for your AC voltage, and is producing the expected voltage level.
- Check the mechanics for abnormally high friction/binding.

BFId _cpmAlertFIds::PhaseSensorFailed

Phase sensor failed power-on self test.

Remedies:

- Replace motor.

BFId _cpmAlertFIds::RMSOverloadShutdown

The RMS torque of the motor has exceeded the critical limit. This is a resettable shutdown condition.

Remedies:

- Check the mechanics for abnormally high friction/binding.
- Reduce velocity, acceleration, and/or duty cycle.
- Check that the configuration file is appropriate for this motor.
- If the motor is actively cooled with a fan raise the continuous current limit of the motor appropriately
- Use a larger motor.

BFId _cpmAlertFIds::SoftLimitExceeded

A move attempted to exceed the bounds set by the software limits. The current move, and any queued moves, are canceled.

Remedies:

- Decrease the distance of the commanded move such that it ends within the bounds of the current soft limit settings.
- Move away from the limit
- Adjust the soft limit settings.

BFId _cpmAlertFIds::StatorTempHigh

The motor has detected an excessive motor stator operating temperature. This condition is recoverable when the temperature has been sufficiently reduced.

Remedies:

- Provide more ventilation/fan cooling to motor.

BFId _cpmAlertFIds::StepsDuringPosnRecovery

Step input commands were seen during the position recovery operation.

Appears in: AlertReg

Remedies:

- Check that power supply is producing the expected voltage level.
- Do not send steps until the motor is ready

BFld _cpmAlertFlds::TrackingShutdown

The position tracking error has exceeded the user specified critical tracking limit. This is a resettable shutdown condition.

Remedies:

- Check the mechanics for abnormally high friction/binding. Grease if necessary.
- Reduce velocity, acceleration, and/or duty cycle.
- Check that the configuration file is appropriate for this motor.
- Increase tracking error limit to a higher value through the ClearView application.

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

- C:/Git/inc/inc-pub/**pubCpmRegs.h**

_cpmStatusRegFlds Struct Reference

Detailed Description

This page provides the detailed list of fields in the ClearPath SC status register. The lower 32-bits of this register are attentionable.

See also

sFnd::ValueStatus for information on accessing different status registers within the Node.

Attentions/Interupts for Event Driven Applications for information regarding attentions.

_cpmStatusRegFlds							
Status Field	Bit #		Status Field	Bit #		Status Field	Bit #
n/a	0		MoveCanceled	16		InMotion	32
UserAlert	1		MoveDone	17			33
NotReady	2		OutOfRange	18		InDisableStop	34
MoveBufAvail	3		BFromEnd	19		n/a	35
Ready	4		AbovePosn	20		n/a	36
n/a	5		AtTargetVel	21		VectorSearch	37
AlertPresent	6		InA	22		MoveCmdComplete	38
n/a	7		InB	23		InHardStop	39
InPosLimit	8		InvInA	24		ShutdownState	40
InNegLimit	9		InvInB	25			41
MotionBlocked	10		n/a	26		HwFailure	42
WasHomed	11		n/a	27		TriggerArmed	43
Homing	12		AFromStart	28		StepsActive	44
GoingDisabled	13		MoveCmdNeg	29		n/a	45
StatusEvent	14		Disabled	30		SoftwareInputs	46
Enabled	15		TimerExpired	31		n/a	47

Status Register Overview

Public Attributes

BFld **UserAlert**: 1
Motion Activity Indicator. More...

BFld **NotReady**: 1
Node not ready for operation. More...

BFld **MoveBufAvail**: 1
Move buffer is available. More...

BFld **Ready**: 1
Node ready for operation. More...

BFld **PowerEvent**: 1
Power Event Occurred. More...

BFld **AlertPresent**: 1
Alert Present. More...

BFld **InPosLimit**: 1
Positive limit asserted. More...

BFld **InNegLimit**: 1
Negative limit asserted. More...

BFld **MotionBlocked**: 1
Motion Blocked. More...

BFld **WasHomed**: 1
Was Homed. More...

BFld **Homing**: 1
In Homing Sequence. More...

BFld **GoingDisabled**: 1
Going To Disable Soon. More...

BFld **StatusEvent**: 1
Status Rise Event detected. More...

BFld **Enabled**: 1
Drive Enabled. More...

BFld **MoveCanceled**: 1
Motion Canceled. More...

BFld **MoveDone**: 1
Move completed and settled. More...

BFld **OutOfRange**: 1
Tracking out of range. More...

BFld **BFromEnd**: 1
B Counts from End of move. More...

BFld **AbovePosn**: 1
Greater than position location. More...

BFld **AtTargetVel**: 1
At Target Velocity. More...

BFld **InA**: 1
Input-A state. More...

BFld **InB**: 1
Input-B state. More...

BFld **InvInA**: 1
Inverted Input-A state. More...

BFld **InvInB**: 1
Inverted Input-B state. More...

BFld **AFromStart**: 1
A From Start of move. More...

BFld **MoveCmdNeg**: 1
Commanded move is negative. More...

BFld **Disabled**: 1
Drive is disabled. More...

BFld **InMotion**: 2
Motion Indicator. More...

BFld **InDisableStop**: 1
The Output Register disable bit is set due to a NodeStop. More...

BFld **VectorSearch**: 1
Vector searching. More...

BFld **MoveCmdComplete**: 1
Motion command has completed. More...

BFld **InHardStop**: 1
In hard-stop mode. More...

BFld **ShutdownState**: 2
Shutdown state. More...

BFld **HwFailure**: 1
Hardware failure detected. More...

BFld **TriggerArmed**: 1
Move awaiting trigger event. More...

BFld **StepsActive**: 1
This field is set true when the step+direction inputs are active. More...

Member Data Documentation

BFld _cpmStatusRegFlds::AbovePosn

This field asserts when the measured position is greater than the Absolute Position location specified in the torque limiting dialog window of ClearView.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::AFromStart

This field asserts when the move is **A From Start** of the move.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::AlertPresent

This field is set when the *Alert Register* is non-zero.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::AtTargetVel

This field asserts when velocity commands have reached their target velocity.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::BFromEnd

This field asserts when the move is **B From End** of the move.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::Disabled

This field asserts when the drive is disabled.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::Enabled

This field is set when the drive is enabled.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::GoingDisabled

Indicates that the drive is about to go disabled.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::Homing

Indicates that a homing sequence is being executed.

See also

Homing Page for more information on homing

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::HwFailure

The unit's shutdown is caused by a hardware failure. This unit should be replaced.

BFld _cpmStatusRegFlds::InA

This field asserts when the Node's input A asserts.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::InB

This field asserts when the Node's input B asserts.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::InDisableStop

The Node Stop feature has forced the enable output to the disabled state. **Clear Nodestops** to return the Node to it's previous enable state.

See also

CPM NodeStops and E-Stops to Cancel Motion

BFld _cpmStatusRegFlds::InHardStop

This field asserts when the ClearPath-SC has entered the *Hard Stopped* mode during homing and the node is applying clamping torque.

See also

HomingPage

BFld _cpmStatusRegFlds::InMotion

This field represents the load's velocity status. The table describes the states:

- 0: Near zero speed
- 1: Moving in a positive direction
- 2: Moving in a negative direction
- 3: Moved in both directions since the last read (for Accumulating Registers)

BFld _cpmStatusRegFlds::InNegLimit

When This field is asserted, the Node is unable to execute moves in the negative direction. The input configured through Clearview to limit motion in the negative direction is currently limiting motion.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::InPosLimit

When This field is asserted, the Node is unable to execute moves in the positive direction. The input configured through Clearview to limit motion in the positive direction is currently limiting motion.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::InvInA

This field asserts when the Input-A is de-asserted.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::InvInB

This field asserts when the Input-B is de-asserted.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::MotionBlocked

This field is asserted when motion is blocked by an E-Stop Nodestop Type.

See also

CPMnodeStopPage

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::MoveBufAvail

This field indicates the Node can receive additional move commands which will be stored in the Node's move buffer. If this bit is deasserted, any move command sent to the Node will be ignored.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::MoveCanceled

This condition will occur when a queued or executing move is canceled en-route by a NodeStop.

See also

CPMnodeStopPage

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::MoveCmdComplete

This field is asserted after the motion command has been completed. The move is considered totally complete when the **Move Done** has been asserted.

BFld _cpmStatusRegFlds::MoveCmdNeg

This field asserts when the current command resulted in the position changing in a negative direction. This field changes at the start of a move and persists until the next move. A zero step move is considered positive.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::MoveDone

This field asserts when a Move is complete.

See also

Move Done Feature

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::NotReady

If this field is true, a Node is not ready to execute motion. You must make sure to **clear shutdowns**, ensure the node is **Enabled**, and **clear any NodeStops.**

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::OutOfRange

This field indicates the Node is currently not tracking within the In-Range window. This is especially common during moves.

See also

Move Done Feature for more information regarding setting up the In-Range window in ClearView

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::PowerEvent

This field is asserted when a power event occurs. The power status register can be used to determine the specifics.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::Ready

This field is asserted when a Node is ready to receive motion commands.

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::ShutdownState

These fields describes the shutdown condition of the node:

- 0: Idle: no shutdowns are present
- 1: Delay: A shutdown is imminent
- 2: Ramp: Node is ramping down to zero speed
- 3: Stopped: Node is currently shutdown

BFld _cpmStatusRegFlds::StatusEvent

This field is set when the bit-wise AND of the **Status Mask** and the Status Register is non-zero.

See also

sFnd::IAttnNode::StatusMask

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::StepsActive

This field is set true when the step and direction inputs are active.

BFld _cpmStatusRegFlds::TriggerArmed

The node has a triggered move awaiting release.

See also

Triggered Moves

BFld _cpmStatusRegFlds::UserAlert

This is a bit-mapped field definition that describes the current motion detected on the motor. The VEL_BOTH state will occur with accumulating status when the motor has moved in both directions since the last status query. User alert is present.

This field is set when the bit-wise AND of the **User Alert register** and the Alert Register is non-zero.

See also

sFnd::IAttnNode::AlertMask

Remarks

This field can generate an attention request to the host for autonomous handling.

BFld _cpmStatusRegFlds::VectorSearch

This field asserted when the vector has not been finalized. Typically, this happens on the first enable after power-up. Once the vector is finalized normal operation can commence.

BFld _cpmStatusRegFlds::WasHomed

Indicates that a homing sequence has been executed.

Remarks

This field can generate an attention request to the host for autonomous handling.

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

- C:/Git/inc/inc-pub/**pubCpmRegs.h**

_mnStatusReg Union Reference

List of all members | Public Member Functions | Public Attributes

Detailed Description

This Union defines the Universal Status Register for all types of Nodes.

The Status register is used to reflect the state of the Node. Status events occur when bits/fields of this register are set/cleared.

The lower 32 bits of this register can be used to generate an asynchronous attention packet to signal the host an event occurred.

See also

sFnd::ValueStatus Class for accessing Status registers within the Node.

cpmStatusRegFlds For the Complete list of ClearPath-SC alert Fields.

Attentions/Interupts for Event Driven Applications for more information regarding Attentions.

Public Member Functions

char *	**StateStr** (char *buffer, size_t size) Update a string with the set fields in the Status Register. More...

Public Attributes

Uint16	**bits** [3] Integer view as three 16-bit integers.
Uint32	**attnBits** Long integer view that represents the attentional bits.

cpmStatusRegFlds	**cpm** Access to the ClearPath-SC status fields.

Member Function Documentation

char* _mnStatusReg::StateStr (char * buffer, size_t size) inline

Parameters

[in,out]	**buffer**	Point to string buffer to update.
[in]	**size**	The size of the buffer in characters including the null character.

Returns

the parameter *buffer*.

Coding Example

```
char myBuffer[500];
cout <<
    myNode.Status.RT.Value.StateStr(myBuffer,
    sizeof(myBuffer));
```

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

- C:/Git/inc/inc-pub/**pubCoreRegs.h**

List of all members | Public Attributes

_ShutdownInfo Struct Reference

Detailed Description

This is the main struct for the *Auto-shutdown* feature. The struct consists of three parts:

- Whether or not the node should respond to a Group Shutdown.
- The Node Stop type for this node.
- The **Status Register** mask for this Node which will trigger a Group shutdown.

Coding Example

```
// The following lines of code show an example of
    how to create a Shutdown info object.

// First we create the status mask object consisting
    of which status events on the node will trigger
    a group shutdown.
// In this case we are using "AlertPresent",
    "Disabled", and "Input A"

mnStatusReg myShutdownStatusMask;
    //Create Status mask
myShutdownStatusMask.cpm.AlertPresent = 1;
    //Set desired fields
myShutdownStatusMask.cpm.Disabled = 1;
myShutdownStatusMask.cpm.InA = 1;

// Next we create the ShutdownInfo object we will
    use to configure the Node with
ShutdownInfo myShutdownInfo;
    //Create ShutdownInfo Object
myShutdownInfo.enabled = true;
    // This sets the node to respond to Group Shutdowns
myShutdownInfo.theStopType = STOP_TYPE_ABRUPT;
    //Use an Abrupt Nodestop type for this node in
    response to a group shutdown
myShutdownInfo.statusMask = myShutdownStatusMask;
    // Configure the node to issue a group shutdown
```

```
    automatically based on our status mask

//Finally we use configure the Port's first Node
    (Node 0) with our shutdown object
myport.GrpShutdown.ShutdownWhen(0,myShutdownInfo);
```

See also

CPM NodeStops and E-Stops to Cancel Motion

Public Attributes

nodebool	**enabled**
mgNodeStopReg	**theStopType**
mnStatusReg	**statusMask**

Member Data Documentation

nodebool _ShutdownInfo::enabled

Enables checking at this node. If this is set to false, the node will not shutdown during a group shutdown.

mnStatusReg _ShutdownInfo::statusMask

Items to trigger a Group Shutdown stop when the Node's status register rises on the unmasked bits of the *statusMask*. Set the bits to non-zero to enable the detection of this event.

mgNodeStopReg _ShutdownInfo::theStopType

The kind of stop to assert when this feature fires off.

See also

CPM NodeStops and E-Stops to Cancel Motion for a complete list of Node Stop types.

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

- C:/Git/inc/inc-pub/**pubNetAPI.h**

commonAlertFlds Struct Reference

Detailed Description

This union defines the common areas of the node's Alert Register. All types of nodes will use these definitions.

See also

alertReg Universal Alert and Warning Register Type.

commonAlertFlds				
Alert Field	Bit #		Alert Field	Bit #
FwSelfTest	0		HwRAM	16
FwNetBufferOverrun	1		HwADC	17
FwError0	2		HwADCsat	18
FwError1	3		0	19
FwStackOverrun	4		0	20
FwWatchdogRestarted	5		0	21
FwInvalidConfiguration	6		0	22
0	7		0	23
0	8		NetVoltageLow	24
0	9		NetWatchdog	25
0	10		0	26
HwPowerProblem	11		0	27
HwClockProblem	12		EStopped	28
HwEEPROMdead	13		ConfigOutOfDate	29
HwFlashCorrupt	14		RunTimeErr	30
HwFlashChanged	15		0	31

Public Attributes

unsigned **FwSelfTest**: 1
Node power-on self test has failed. More...

unsigned	**FwNetBufferOverrun**: 1 Network buffer overrun. More...
unsigned	**FwError0**: 1 Internal firmware error 0. More...
unsigned	**FwError1**: 1 Internal firmware error 0. More...
unsigned	**FwStackOverrun**: 1 Internal firmware error due to stack issues. More...
unsigned	**FwWatchdogRestarted**: 1 The DSP watchdog timer has restarted the node. More...
unsigned	**FwInvalidConfiguration**: 1 Invalid setting detected. More...
unsigned	**HwPowerProblem**: 1 Power supply problem. More...
unsigned	**HwClockProblem**: 1 DSP clock problem. More...
unsigned	**HwEEPROMdead**: 1 The node's non-volatile memory is broken. More...
unsigned	**HwFlashCorrupt**: 1 Flash content drifting. More...
unsigned	**HwFlashChanged**: 1 Flash changed due to update. More...
unsigned	**HwRAM**: 1 The power-on self test has determined the RAM is bad.

	More...
unsigned	**HwADC**: 1 Analog to digital converter failed self-test. More...
unsigned	**HwADCsat**: 1 Analog to digital converter Saturated. More...
unsigned	**NetVoltageLow**: 1 Link/Node power supply low. More...
unsigned	**NetWatchdog**: 1 Network activity watchdog node stop. More...
unsigned	**EStopped**: 1 Node E-Stopped. More...
unsigned	**ConfigOutOfDate**: 1 Configuration file out of date for firmware. More...
unsigned	**RunTimeErr**: 1 Internal firmware error. More...

Member Data Documentation

unsigned commonAlertFlds::ConfigOutOfDate

The node's internal configuration file is out of date for the loaded firmware.

Remedies:

- Clear to factory defaults and load a valid configuration file.

unsigned commonAlertFlds::EStopped

The node has entered an E-Stop state and will no longer accept motion commands until this condition is cleared. This code is not cleared with the **sFnd::IStatus::AlertsClear** function to prevent unintended motion.

Remedies:

- Issue a **sFnd::IMotion::NodeStopClear** command to clear the E-Stop node stopped state.

unsigned commonAlertFlds::FwError0

Internal firmware error 0. Remedies:

- Update firmware.

unsigned commonAlertFlds::FwError1

Internal firmware error 0.

Remedies:

- Update firmware.

unsigned commonAlertFlds::FwInvalidConfiguration

An invalid setting is detected that prevents the drive from running.

Remedies:

- Clear to factory defaults and load a valid configuration file.
- Replace motor.

unsigned commonAlertFlds::FwNetBufferOverrun

Network buffer overrun.

Remedies:

- Check command pacing.

unsigned commonAlertFlds::FwSelfTest

Node power-on self test has failed.

Remedies:

- Replace motor.

unsigned commonAlertFlds::FwStackOverrun

Internal firmware error due to stack issues.

Remedies:

- Update firmware.

unsigned commonAlertFlds::FwWatchdogRestarted

The DSP watchdog timer has restarted the node.

Remedies:

- Check link power supply for brown-outs.
- Update firmware.

unsigned commonAlertFlds::HwADC

The power-on self test has determined the Analog to Digital converter or its related analog hardware is bad.

Remedies:

- Replace motor.

unsigned commonAlertFlds::HwADCsat

Analog to digital converter in saturation mode.

Remedies:

- Check for new firmware.
- Replace motor.

unsigned commonAlertFlds::HwClockProblem

The DSP clock did not initialize properly. Most likely you will not be able to communicate with this node. This condition is stored in internal log for factory use.

Remedies:

- Replace motor.

unsigned commonAlertFlds::HwEEPROMdead

The node's non-volatile memory is broken.

Remedies:

- Replace motor.

unsigned commonAlertFlds::HwFlashChanged

This bit is asserted after the flash memory is updated with new firmware or has drifted from the known image. Normally the Teknic supplied flash updater will reset this when it clears the configuration out of the node as the firmware is updated.

Remedies:

- If the firmware has not been recently updated, replace motor.
- Reload the node's configuration file using the ClearView application.

unsigned commonAlertFlds::HwFlashCorrupt

The run-time test of the firmware has determined the flash memory is drifting.

Remedies:

- Replace motor.

unsigned commonAlertFlds::HwPowerProblem

Onboard power supply out of regulation or missing. This condition is stored in internal log for factory use.

Remedies:

- Replace motor.

unsigned commonAlertFlds::HwRAM

The power-on self test has determined the RAM is bad.

Remedies:

- Replace motor.

unsigned commonAlertFlds::NetVoltageLow

The link and node's internal power supply is low.

Remedies:

- Replace motor.

unsigned commonAlertFlds::NetWatchdog

The node has entered a node stop state due to a lack of communications from the host.

Remedies:

- Ensure the application requests some real-time parameter or run a command within the watchdog time.
- Adjust or disable the network activity watchdog parameter(sFnd::ISetupEx::NetWatchdogMsec).

unsigned commonAlertFlds::RunTimeErr

Internal firmware error.

Remedies:

- Update node firmware to newest release.
- Replace motor.

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

- inc-doc/CPSCAlertsWarnings.doc

---

sFoundation 2.0

Class Index

C | I | S | T | U | V | _

I

IAttnNode (sFnd)
IAttnPort (sFnd)
IBrakeControl (sFnd)
IGrpShutdown (sFnd)
IHoming (sFnd)
IInfo (sFnd)
IInfoEx (sFnd)
ILimits (sFnd)
IMotion (sFnd)
IMotionAdv (sFnd)
INode (sFnd)
INodeAdv (sFnd)
IObj (sFnd)
IObjWithNode (sFnd)
IObjWithPort (sFnd)
IPort (sFnd)
IPortAdv (sFnd)
ISetup (sFnd)
ISetupEx (sFnd)
IStatus (sFnd)
IStatusAdv (sFnd)

S

SysManager (sFnd)

T

SysManager::ThreadLock (sFnd)

U

INode::UseMutex (sFnd)

V

ValueAlert (sFnd)
ValueBase (s
ValueDouble
ValuePowerI
ValueSigned
ValueStatus
ValueString
ValueUnsign

_

_alertReg
_cpmAlertFl

C | I | S | T | U | V | _

sFoundation 2.0

Class Hierarchy

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

[detail level 1 2 3 4]

_alertReg	Universal Alert and Warning Register Type
_cpmAlertFlds	ClearPath-SC Alert and Warning Register Field Definitions
_cpmStatusRegFlds	ClearPath-SC Node Status Register Fields
sFnd::_mnErr	Error code and reason structure
_mnStatusReg	Universal Status Register Type
_ShutdownInfo	Information to setup group shutdowns at a node
commonAlertFlds	Alert bits common to all nodes
sFnd::INode	Virtual Node Interface
▼ sFnd::IObj	Feature Availability Interface
▼ sFnd::IObjWithNode	Feature Availability And Root Node Access Interface

sFnd::IAttnNode	*Node Attention* Feature Interface
sFnd::IHoming	Homing Feature Interface
sFnd::IInfo	Node Information Interface
sFnd::IInfoEx	Expert Information Interface
sFnd::ILimits	Torque and Position Limits Feature Interface
sFnd::IMotion	Motion Control Feature Interface
sFnd::IMotionAdv	Advanced Motion Control Interface
sFnd::INodeAdv	Advanced Node Features Interface
sFnd::ISetup	Node Setup Features Interface
sFnd::ISetupEx	Expert Setup Interface
sFnd::IStatus	Node Status Features Interface
sFnd::IStatusAdv	Advanced Status Features Interface
▼ sFnd::ValueBase	Node Parameter Container Base Class
sFnd::ValueAlert	Alert Type Register Node Parameter Access Object
sFnd::ValueDouble	This class object implements access to floating point parameters from a node. This class object implements access to floating point

	parameters within the node
sFnd::ValuePowerReg	Power Status Register Node Parameter Access Object
sFnd::ValueSigned	Signed 32-bit Integer Parameter Object
sFnd::ValueStatus	Status Type Register Node Parameter Access Object
sFnd::ValueString	String Parameter Object
sFnd::ValueUnsigned	Unsigned 32-bit Integer Parameter Object
▼ sFnd::IObjWithPort	Feature Availability And Port Access Interface
sFnd::IAttnPort	Serial Port Attention Feature Interface
sFnd::IBrakeControl	Brake Control Feature Interface
sFnd::IGrpShutdown	Group Shutdown Feature Interface
sFnd::IPortAdv	Advanced Port Features Interface
sFnd::IPort	Serial Port Interface
sFnd::SysManager	SFoundation Top Level System Object
sFnd::SysManager::ThreadLock	A safe thread locking mechanism
sFnd::INode::UseMutex	A Simple and Safe Mutex mechanism

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- _ -

- _accUnits : **sFnd::INode**
- _nodeTypes : **sFnd::IInfo**
- _trqUnits : **sFnd::INode**
- _velUnits : **sFnd::INode**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- a -

- AbovePosn : **_cpmStatusRegFlds**
- AccessLevelIsFull() : **sFnd::ISetup**
- AccLimit : **sFnd::IMotion**
- Accum : **sFnd::IStatus**
- AccUnit() : **sFnd::INode**
- ACPhaseLost : **_cpmAlertFlds**
- ACPowerLost : **_cpmAlertFlds**
- Addr() : **sFnd::IInfoEx**
- AddToPosition() : **sFnd::IMotion**
- Adv : **sFnd::IMotion** , **sFnd::INode** , **sFnd::IPort** , **sFnd::IStatus**
- AFromStart : **_cpmStatusRegFlds**
- AfterStartDistance : **sFnd::IMotionAdv**
- AlertMask : **sFnd::IAttnNode**
- AlertPresent : **_cpmStatusRegFlds**
- Alerts : **sFnd::IStatus**
- AlertsClear() : **sFnd::IStatus**
- AMPS : **sFnd::INode**
- AtTargetVel : **_cpmStatusRegFlds**
- Attn : **sFnd::INodeAdv** , **sFnd::IPortAdv**
- attnBits : **_mnStatusReg**
- AttnHandler() : **sFnd::IAttnPort**

- AutoRefresh() : **sFnd::ValueBase**

---

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- b -

- BeforeEndDistance : **sFnd::IMotionAdv**
- BFromEnd : **_cpmStatusRegFlds**
- bits : **_alertReg** , **_mnStatusReg**
- BrakeControl : **sFnd::IPort**
- BrakeEngaged() : **sFnd::IBrakeControl**
- BrakeSetting() : **sFnd::IBrakeControl**
- BusOverCurrent : **_cpmAlertFlds**
- BusOverVoltage : **_cpmAlertFlds**
- BusRMSOverload : **_cpmAlertFlds**
- BusVoltageLow : **_cpmAlertFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- c -

- CapturedHiResPosn : **sFnd::IStatusAdv**
- CapturedPos : **sFnd::IStatusAdv**
- Clear() : **sFnd::ValueStatus**
- ClearAttn() : **sFnd::IAttnNode**
- CLEARPATH_SC : **sFnd::IInfo**
- CLEARPATH_SC_ADV : **sFnd::IInfo**
- ComHubPort() : **sFnd::SysManager**
- CommandTraceSave() : **sFnd::IPort**
- Common : **_cpmAlertFlds**
- ConfigLoad() : **sFnd::ISetup**
- ConfigOutOfDate : **commonAlertFlds**
- ConfigSave() : **sFnd::ISetup**
- COUNTS_PER_SEC : **sFnd::INode**
- COUNTS_PER_SEC2 : **sFnd::INode**
- cpm : **_alertReg** , **_mnStatusReg**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- d -

- DecelLimit : **sFnd::IMotionAdv**
- Delay() : **sFnd::SysManager**
- DelayToDisableMsecs : **sFnd::ISetup**
- Disabled : **_cpmStatusRegFlds**
- DriveTempHigh : **_cpmAlertFlds**
- DwellMs : **sFnd::IMotion**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- e -

- Enable() : **sFnd::IAttnPort**
- Enabled : **_cpmStatusRegFlds**
- enabled : **_ShutdownInfo**
- Enabled() : **sFnd::IAttnPort**
- EnableReq() : **sFnd::INode**
- ErrorCode : **sFnd::_mnErr**
- ErrorMsg : **sFnd::_mnErr**
- EStopped : **commonAlertFlds**
- Ex : **sFnd::IInfo** , **sFnd::ISetup**
- Exists() : **sFnd::ValueBase**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- f -

- Fall : **sFnd::IStatus**
- FindComHubPorts() : **sFnd::SysManager**
- FirmwareVersion : **sFnd::IInfo**
- FwError0 : **commonAlertFlds**
- FwError1 : **commonAlertFlds**
- FwInvalidConfiguration : **commonAlertFlds**
- FwNetBufferOverrun : **commonAlertFlds**
- FwSelfTest : **commonAlertFlds**
- FwStackOverrun : **commonAlertFlds**
- FwWatchdogRestarted : **commonAlertFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- g -

- GoingDisabled : **_cpmStatusRegFlds**
- GroupNodeStop() : **sFnd::IMotion**
- GrpShutdown : **sFnd::IPort**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- h -

- HadTorqueSaturation() : **sFnd::IStatus**
- HardStopBrokeFree : **_cpmAlertFlds**
- HasAttnHandler() : **sFnd::IAttnPort**
- HeadDistance : **sFnd::IMotionAdv**
- HeadTailVelLimit : **sFnd::IMotionAdv**
- Homing : **_cpmStatusRegFlds** , **sFnd::IMotion**
- HomingValid() : **sFnd::IHoming**
- HwADC : **commonAlertFlds**
- HwADCsat : **commonAlertFlds**
- HwClockProblem : **commonAlertFlds**
- HwEEPROMdead : **commonAlertFlds**
- HwFailure : **_cpmStatusRegFlds**
- HwFlashChanged : **commonAlertFlds**
- HwFlashCorrupt : **commonAlertFlds**
- HwPowerProblem : **commonAlertFlds**
- HwRAM : **commonAlertFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- i -

- InA : **_cpmStatusRegFlds**
- InB : **_cpmStatusRegFlds**
- InDisableStop : **_cpmStatusRegFlds**
- Info : **sFnd::INode**
- InHardStop : **_cpmStatusRegFlds**
- Initiate() : **sFnd::IHoming**
- InMotion : **_cpmStatusRegFlds**
- InNegLimit : **_cpmStatusRegFlds**
- InPosLimit : **_cpmStatusRegFlds**
- InvInA : **_cpmStatusRegFlds**
- InvInB : **_cpmStatusRegFlds**
- isClear() : **_alertReg**
- IsHoming() : **sFnd::IHoming**
- isInAlert() : **_alertReg**
- IsReady() : **sFnd::IMotion** , **sFnd::IStatus**
- IsVolatile() : **sFnd::ValueBase**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- j -

- JrkLimVelRequestBad : **_cpmAlertFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- l -

- Last() : **sFnd::ValueAlert** , **sFnd::ValuePowerReg** , **sFnd::ValueStatus**
- Limits : **sFnd::INode**
- LimSwitchActivated : **_cpmAlertFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- m -

- Mask : **sFnd::IAttnNode**
- Model : **sFnd::IInfo**
- Motion : **sFnd::INode**
- MotionBlocked : **_cpmStatusRegFlds**
- MoveBufAvail : **_cpmStatusRegFlds**
- MoveCanceled : **_cpmStatusRegFlds**
- MoveCmdComplete : **_cpmStatusRegFlds**
- MoveCmdNeg : **_cpmStatusRegFlds**
- MoveDone : **_cpmStatusRegFlds**
- MoveIsDone() : **sFnd::IMotion**
- MovePosnAsymDurationMsec() : **sFnd::IMotionAdv**
- MovePosnAsymStart() : **sFnd::IMotionAdv**
- MovePosnDurationMsec() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MovePosnHeadTailDurationMsec() : **sFnd::IMotionAdv**
- MovePosnHeadTailStart() : **sFnd::IMotionAdv**
- MovePosnStart() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MoveVelDurationMsec() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MoveVelStart() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MoveWentDone() : **sFnd::IMotion**
- MtrBadSetupParams : **_cpmAlertFlds**
- MtrEncGlitch : **_cpmAlertFlds**

- MtrEncIndexMisplaced : **_cpmAlertFlds**
- MtrPhaseOverload : **_cpmAlertFlds**
- MtrTempLow : **_cpmAlertFlds**
- MtrVectorBad : **_cpmAlertFlds**

---

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- n -

- NetVoltageLow : **commonAlertFlds**
- NetWatchdog : **commonAlertFlds**
- NetWatchdogMsec : **sFnd::ISetupEx**
- NoCommFailed : **_cpmAlertFlds**
- NoCommSweepFailed : **_cpmAlertFlds**
- NoCommSweepReversed : **_cpmAlertFlds**
- Node() : **sFnd::IObjWithNode**
- NodeCount() : **sFnd::IPort**
- NodeGet() : **sFnd::SysManager**
- NodeIndex() : **sFnd::IInfoEx**
- Nodes() : **sFnd::IPort**
- NodeStop() : **sFnd::IMotion** , **sFnd::IPort**
- NodeStopClear() : **sFnd::IMotion**
- NodeType() : **sFnd::IInfo**
- nodeTypes : **sFnd::IInfo**
- NotReady : **_cpmStatusRegFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- o -

- OpenState() : **sFnd::IPort**
- operator alertReg() : **sFnd::ValueAlert**
- operator const char *() : **sFnd::ValueString**
- operator double() : **sFnd::ValueDouble** , **sFnd::ValueSigned** , **sFnd::ValueUnsigned**
- operator int32_t() : **sFnd::ValueDouble** , **sFnd::ValueSigned**
- operator mnPowerReg() : **sFnd::ValuePowerReg**
- operator mnStatusReg() : **sFnd::ValueStatus**
- operator uint32_t() : **sFnd::ValueDouble** , **sFnd::ValueSigned** , **sFnd::ValueUnsigned**
- operator=() : **sFnd::ValueAlert** , **sFnd::ValueDouble** , **sFnd::ValueSigned** , **sFnd::ValueStatus** , **sFnd::ValueString** , **sFnd::ValueUnsigned**
- OutOfRange : **_cpmStatusRegFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- p -

- Parameter() : **sFnd::IInfoEx**
- PCT_MAX : **sFnd::INode**
- PhaseSensorFailed : **_cpmAlertFlds**
- Port : **sFnd::INode**
- Ports() : **sFnd::SysManager**
- PortsClose() : **sFnd::SysManager**
- PortsOpen() : **sFnd::SysManager**
- PositioningResolution : **sFnd::IInfo**
- PosnCommanded : **sFnd::IMotion**
- PosnMeasured : **sFnd::IMotion**
- PosnTracking : **sFnd::IMotion**
- PosnTrackingLimit : **sFnd::ILimits**
- Power : **sFnd::IStatus**
- PowerEvent : **_cpmStatusRegFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- r -

- Ready : **_cpmStatusRegFlds**
- Refresh() : **sFnd::ValueAlert** , **sFnd::ValueBase** , **sFnd::ValueDouble** , **sFnd::ValuePowerReg** , **sFnd::ValueSigned** , **sFnd::ValueStatus** , **sFnd::ValueString** , **sFnd::ValueUnsigned**
- RestartCold() : **sFnd::IPort** , **sFnd::SysManager**
- RestartWarm() : **sFnd::SysManager**
- Rise : **sFnd::IStatus**
- RMSlevel : **sFnd::IStatus**
- RMSOverloadShutdown : **_cpmAlertFlds**
- RPM : **sFnd::INode**
- RPM_PER_SEC : **sFnd::INode**
- RT : **sFnd::IStatus**
- RunTimeErr : **commonAlertFlds**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- s -

- SerialNumber : **sFnd::IInfo**
- Setup : **sFnd::INode**
- ShutdownInitiate() : **sFnd::IGrpShutdown**
- ShutdownState : **_cpmStatusRegFlds**
- ShutdownWhen() : **sFnd::IGrpShutdown**
- ShutdownWhenGet() : **sFnd::IGrpShutdown**
- SignalComplete() : **sFnd::IHoming**
- SignalInvalid() : **sFnd::IHoming**
- SoftLimitExceeded : **_cpmAlertFlds**
- StateStr() : **_alertReg** , **_cpmAlertFlds** , **_mnStatusReg**
- StatorTempHigh : **_cpmAlertFlds**
- Status : **sFnd::INode**
- StatusEvent : **_cpmStatusRegFlds**
- statusMask : **_ShutdownInfo**
- StatusMask : **sFnd::IAttnNode**
- StepsActive : **_cpmStatusRegFlds**
- StepsDuringPosnRecovery : **_cpmAlertFlds**
- Supported() : **sFnd::IObj** , **sFnd::IPortAdv**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- t -

- TailDistance : **sFnd::IMotionAdv**
- TestAndClear() : **sFnd::ValueStatus**
- TheAddr : **sFnd::_mnErr**
- theStopType : **_ShutdownInfo**
- ThreadLock() : **sFnd::SysManager::ThreadLock**
- TimeStampMsec() : **sFnd::SysManager**
- TimeStampMsecStr() : **sFnd::SysManager**
- TrackingShutdown : **_cpmAlertFlds**
- TriggerArmed : **_cpmStatusRegFlds**
- TriggerGroup() : **sFnd::IMotionAdv**
- TriggerMove() : **sFnd::IMotionAdv**
- TriggerMovesInGroup() : **sFnd::IPortAdv**
- TriggerMovesInMyGroup() : **sFnd::IMotionAdv**
- TrqCommanded : **sFnd::IMotion**
- TrqGlobal : **sFnd::ILimits**
- TrqMeasured : **sFnd::IMotion**
- TrqUnit() : **sFnd::INode**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- u -

- UNKNOWN : **sFnd::IInfo**
- UseMutex() : **sFnd::INode::UseMutex**
- UserAlert : **_cpmStatusRegFlds**
- UserData() : **sFnd::IInfo**
- UserID : **sFnd::IInfo**
- UserRAM : **sFnd::IInfo**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- v -

- Value() : **sFnd::ValueAlert** , **sFnd::ValueDouble** , **sFnd::ValuePowerReg** , **sFnd::ValueSigned** , **sFnd::ValueStatus** , **sFnd::ValueString** , **sFnd::ValueUnsigned**
- VectorSearch : **_cpmStatusRegFlds**
- VelCommanded : **sFnd::IMotion**
- VelLimit : **sFnd::IMotion**
- VelMeasured : **sFnd::IMotion**
- VelocityAtTarget() : **sFnd::IMotion**
- VelocityReachedTarget() : **sFnd::IMotion**
- VelUnit() : **sFnd::INode**

sFoundation 2.0

Here is a list of all documented class members with links to the class documentation for each member:

- w -

- WaitForAttn() : **sFnd::IAttnNode**
- WaitForOnline() : **sFnd::IPort**
- WasHomed : **_cpmStatusRegFlds** , **sFnd::IHoming**
- WentNotReady() : **sFnd::IMotion**

sFoundation 2.0

- a -

- AccessLevelIsFull() : **sFnd::ISetup**
- AccUnit() : **sFnd::INode**
- Addr() : **sFnd::IInfoEx**
- AddToPosition() : **sFnd::IMotion**
- AlertsClear() : **sFnd::IStatus**
- AttnHandler() : **sFnd::IAttnPort**
- AutoRefresh() : **sFnd::ValueBase**

- b -

- BrakeEngaged() : **sFnd::IBrakeControl**
- BrakeSetting() : **sFnd::IBrakeControl**

- c -

- Clear() : **sFnd::ValueStatus**
- ClearAttn() : **sFnd::IAttnNode**
- ComHubPort() : **sFnd::SysManager**
- CommandTraceSave() : **sFnd::IPort**
- ConfigLoad() : **sFnd::ISetup**
- ConfigSave() : **sFnd::ISetup**

- d -

- Delay() : **sFnd::SysManager**

- e -

- Enable() : **sFnd::IAttnPort**
- Enabled() : **sFnd::IAttnPort**
- EnableReq() : **sFnd::INode**
- Exists() : **sFnd::ValueBase**

- f -

- FindComHubPorts() : **sFnd::SysManager**

- g -

- GroupNodeStop() : **sFnd::IMotion**

- h -

- HadTorqueSaturation() : **sFnd::IStatus**
- HasAttnHandler() : **sFnd::IAttnPort**
- HomingValid() : **sFnd::IHoming**

- i -

- Initiate() : **sFnd::IHoming**
- isClear() : **_alertReg**
- IsHoming() : **sFnd::IHoming**
- isInAlert() : **_alertReg**
- IsReady() : **sFnd::IMotion** , **sFnd::IStatus**
- IsVolatile() : **sFnd::ValueBase**

- l -

- Last() : **sFnd::ValueAlert** , **sFnd::ValuePowerReg** , **sFnd::ValueStatus**

- m -

- MoveIsDone() : **sFnd::IMotion**
- MovePosnAsymDurationMsec() : **sFnd::IMotionAdv**
- MovePosnAsymStart() : **sFnd::IMotionAdv**
- MovePosnDurationMsec() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MovePosnHeadTailDurationMsec() : **sFnd::IMotionAdv**
- MovePosnHeadTailStart() : **sFnd::IMotionAdv**
- MovePosnStart() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MoveVelDurationMsec() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MoveVelStart() : **sFnd::IMotion** , **sFnd::IMotionAdv**
- MoveWentDone() : **sFnd::IMotion**

- n -

- Node() : **sFnd::IObjWithNode**
- NodeCount() : **sFnd::IPort**
- NodeGet() : **sFnd::SysManager**
- NodeIndex() : **sFnd::IInfoEx**
- Nodes() : **sFnd::IPort**
- NodeStop() : **sFnd::IMotion** , **sFnd::IPort**
- NodeStopClear() : **sFnd::IMotion**
- NodeType() : **sFnd::IInfo**

- o -

- OpenState() : **sFnd::IPort**
- operator alertReg() : **sFnd::ValueAlert**
- operator const char *() : **sFnd::ValueString**
- operator double() : **sFnd::ValueDouble** , **sFnd::ValueSigned** , **sFnd::ValueUnsigned**
- operator int32_t() : **sFnd::ValueDouble** , **sFnd::ValueSigned**
- operator mnPowerReg() : **sFnd::ValuePowerReg**
- operator mnStatusReg() : **sFnd::ValueStatus**

- operator uint32_t() : **sFnd::ValueDouble** , **sFnd::ValueSigned** , **sFnd::ValueUnsigned**
- operator=() : **sFnd::ValueAlert** , **sFnd::ValueDouble** , **sFnd::ValueSigned** , **sFnd::ValueStatus** , **sFnd::ValueString** , **sFnd::ValueUnsigned**

- p -

- Parameter() : **sFnd::IInfoEx**
- Ports() : **sFnd::SysManager**
- PortsClose() : **sFnd::SysManager**
- PortsOpen() : **sFnd::SysManager**

- r -

- Refresh() : **sFnd::ValueAlert** , **sFnd::ValueBase** , **sFnd::ValueDouble** , **sFnd::ValuePowerReg** , **sFnd::ValueSigned** , **sFnd::ValueStatus** , **sFnd::ValueString** , **sFnd::ValueUnsigned**
- RestartCold() : **sFnd::IPort** , **sFnd::SysManager**
- RestartWarm() : **sFnd::SysManager**

- s -

- ShutdownInitiate() : **sFnd::IGrpShutdown**
- ShutdownWhen() : **sFnd::IGrpShutdown**
- ShutdownWhenGet() : **sFnd::IGrpShutdown**
- SignalComplete() : **sFnd::IHoming**
- SignalInvalid() : **sFnd::IHoming**
- StateStr() : **_alertReg** , **_cpmAlertFlds** , **_mnStatusReg**
- Supported() : **sFnd::IObj** , **sFnd::IPortAdv**

- t -

- TestAndClear() : **sFnd::ValueStatus**
- ThreadLock() : **sFnd::SysManager::ThreadLock**
- TimeStampMsec() : **sFnd::SysManager**
- TimeStampMsecStr() : **sFnd::SysManager**

- TriggerGroup() : **sFnd::IMotionAdv**
- TriggerMove() : **sFnd::IMotionAdv**
- TriggerMovesInGroup() : **sFnd::IPortAdv**
- TriggerMovesInMyGroup() : **sFnd::IMotionAdv**
- TrqUnit() : **sFnd::INode**

- u -

- UseMutex() : **sFnd::INode::UseMutex**
- UserData() : **sFnd::IInfo**

- v -

- Value() : **sFnd::ValueAlert** , **sFnd::ValueDouble** , **sFnd::ValuePowerReg** , **sFnd::ValueSigned** , **sFnd::ValueStatus** , **sFnd::ValueString** , **sFnd::ValueUnsigned**
- VelocityAtTarget() : **sFnd::IMotion**
- VelocityReachedTarget() : **sFnd::IMotion**
- VelUnit() : **sFnd::INode**

- w -

- WaitForAttn() : **sFnd::IAttnNode**
- WaitForOnline() : **sFnd::IPort**
- WasHomed() : **sFnd::IHoming**
- WentNotReady() : **sFnd::IMotion**

sFoundation 2.0

- a -

- AbovePosn : **_cpmStatusRegFlds**
- AccLimit : **sFnd::IMotion**
- Accum : **sFnd::IStatus**
- ACPhaseLost : **_cpmAlertFlds**
- ACPowerLost : **_cpmAlertFlds**
- Adv : **sFnd::IMotion** , **sFnd::INode** , **sFnd::IPort** , **sFnd::IStatus**
- AFromStart : **_cpmStatusRegFlds**
- AfterStartDistance : **sFnd::IMotionAdv**
- AlertMask : **sFnd::IAttnNode**
- AlertPresent : **_cpmStatusRegFlds**
- Alerts : **sFnd::IStatus**
- AtTargetVel : **_cpmStatusRegFlds**
- Attn : **sFnd::INodeAdv** , **sFnd::IPortAdv**
- attnBits : **_mnStatusReg**

- b -

- BeforeEndDistance : **sFnd::IMotionAdv**
- BFromEnd : **_cpmStatusRegFlds**
- bits : **_alertReg** , **_mnStatusReg**
- BrakeControl : **sFnd::IPort**
- BusOverCurrent : **_cpmAlertFlds**

- BusOverVoltage : **_cpmAlertFlds**
- BusRMSOverload : **_cpmAlertFlds**
- BusVoltageLow : **_cpmAlertFlds**

- c -

- CapturedHiResPosn : **sFnd::IStatusAdv**
- CapturedPos : **sFnd::IStatusAdv**
- Common : **_cpmAlertFlds**
- ConfigOutOfDate : **commonAlertFlds**
- cpm : **_alertReg** , **_mnStatusReg**

- d -

- DecelLimit : **sFnd::IMotionAdv**
- DelayToDisableMsecs : **sFnd::ISetup**
- Disabled : **_cpmStatusRegFlds**
- DriveTempHigh : **_cpmAlertFlds**
- DwellMs : **sFnd::IMotion**

- e -

- Enabled : **_cpmStatusRegFlds**
- enabled : **_ShutdownInfo**
- ErrorCode : **sFnd::_mnErr**
- ErrorMsg : **sFnd::_mnErr**
- EStopped : **commonAlertFlds**
- Ex : **sFnd::IInfo** , **sFnd::ISetup**

- f -

- Fall : **sFnd::IStatus**
- FirmwareVersion : **sFnd::IInfo**
- FwError0 : **commonAlertFlds**
- FwError1 : **commonAlertFlds**
- FwInvalidConfiguration : **commonAlertFlds**
- FwNetBufferOverrun : **commonAlertFlds**
- FwSelfTest : **commonAlertFlds**

- FwStackOverrun : **commonAlertFlds**
- FwWatchdogRestarted : **commonAlertFlds**

- g -

- GoingDisabled : **_cpmStatusRegFlds**
- GrpShutdown : **sFnd::IPort**

- h -

- HardStopBrokeFree : **_cpmAlertFlds**
- HeadDistance : **sFnd::IMotionAdv**
- HeadTailVelLimit : **sFnd::IMotionAdv**
- Homing : **_cpmStatusRegFlds** , **sFnd::IMotion**
- HwADC : **commonAlertFlds**
- HwADCsat : **commonAlertFlds**
- HwClockProblem : **commonAlertFlds**
- HwEEPROMdead : **commonAlertFlds**
- HwFailure : **_cpmStatusRegFlds**
- HwFlashChanged : **commonAlertFlds**
- HwFlashCorrupt : **commonAlertFlds**
- HwPowerProblem : **commonAlertFlds**
- HwRAM : **commonAlertFlds**

- i -

- InA : **_cpmStatusRegFlds**
- InB : **_cpmStatusRegFlds**
- InDisableStop : **_cpmStatusRegFlds**
- Info : **sFnd::INode**
- InHardStop : **_cpmStatusRegFlds**
- InMotion : **_cpmStatusRegFlds**
- InNegLimit : **_cpmStatusRegFlds**
- InPosLimit : **_cpmStatusRegFlds**
- InvInA : **_cpmStatusRegFlds**
- InvInB : **_cpmStatusRegFlds**

- j -

- JrkLimVelRequestBad : **_cpmAlertFlds**

- l -

- Limits : **sFnd::INode**
- LimSwitchActivated : **_cpmAlertFlds**

- m -

- Mask : **sFnd::IAttnNode**
- Model : **sFnd::IInfo**
- Motion : **sFnd::INode**
- MotionBlocked : **_cpmStatusRegFlds**
- MoveBufAvail : **_cpmStatusRegFlds**
- MoveCanceled : **_cpmStatusRegFlds**
- MoveCmdComplete : **_cpmStatusRegFlds**
- MoveCmdNeg : **_cpmStatusRegFlds**
- MoveDone : **_cpmStatusRegFlds**
- MtrBadSetupParams : **_cpmAlertFlds**
- MtrEncGlitch : **_cpmAlertFlds**
- MtrEncIndexMisplaced : **_cpmAlertFlds**
- MtrPhaseOverload : **_cpmAlertFlds**
- MtrTempLow : **_cpmAlertFlds**
- MtrVectorBad : **_cpmAlertFlds**

- n -

- NetVoltageLow : **commonAlertFlds**
- NetWatchdog : **commonAlertFlds**
- NetWatchdogMsec : **sFnd::ISetupEx**
- NoCommFailed : **_cpmAlertFlds**
- NoCommSweepFailed : **_cpmAlertFlds**
- NoCommSweepReversed : **_cpmAlertFlds**
- NotReady : **_cpmStatusRegFlds**

- o -

- OutOfRange : **_cpmStatusRegFlds**

- p -

- PhaseSensorFailed : **_cpmAlertFlds**
- Port : **sFnd::INode**
- PositioningResolution : **sFnd::IInfo**
- PosnCommanded : **sFnd::IMotion**
- PosnMeasured : **sFnd::IMotion**
- PosnTracking : **sFnd::IMotion**
- PosnTrackingLimit : **sFnd::ILimits**
- Power : **sFnd::IStatus**
- PowerEvent : **_cpmStatusRegFlds**

- r -

- Ready : **_cpmStatusRegFlds**
- Rise : **sFnd::IStatus**
- RMSlevel : **sFnd::IStatus**
- RMSOverloadShutdown : **_cpmAlertFlds**
- RT : **sFnd::IStatus**
- RunTimeErr : **commonAlertFlds**

- s -

- SerialNumber : **sFnd::IInfo**
- Setup : **sFnd::INode**
- ShutdownState : **_cpmStatusRegFlds**
- SoftLimitExceeded : **_cpmAlertFlds**
- StatorTempHigh : **_cpmAlertFlds**
- Status : **sFnd::INode**
- StatusEvent : **_cpmStatusRegFlds**
- statusMask : **_ShutdownInfo**
- StatusMask : **sFnd::IAttnNode**
- StepsActive : **_cpmStatusRegFlds**
- StepsDuringPosnRecovery : **_cpmAlertFlds**

- t -

- TailDistance : **sFnd::IMotionAdv**

- TheAddr : **sFnd::_mnErr**
- theStopType : **_ShutdownInfo**
- TrackingShutdown : **_cpmAlertFlds**
- TriggerArmed : **_cpmStatusRegFlds**
- TrqCommanded : **sFnd::IMotion**
- TrqGlobal : **sFnd::ILimits**
- TrqMeasured : **sFnd::IMotion**

- u -

- UserAlert : **_cpmStatusRegFlds**
- UserID : **sFnd::IInfo**
- UserRAM : **sFnd::IInfo**

- v -

- VectorSearch : **_cpmStatusRegFlds**
- VelCommanded : **sFnd::IMotion**
- VelLimit : **sFnd::IMotion**
- VelMeasured : **sFnd::IMotion**

- w -

- WasHomed : **_cpmStatusRegFlds**

sFoundation 2.0

- nodeTypes : **sFnd::llInfo**

- _accUnits : **sFnd::INode**
- _nodeTypes : **sFnd::IInfo**
- _trqUnits : **sFnd::INode**
- _velUnits : **sFnd::INode**

sFoundation 2.0

- AMPS : **sFnd::INode**
- CLEARPATH_SC : **sFnd::IInfo**
- CLEARPATH_SC_ADV : **sFnd::IInfo**
- COUNTS_PER_SEC : **sFnd::INode**
- COUNTS_PER_SEC2 : **sFnd::INode**
- PCT_MAX : **sFnd::INode**
- RPM : **sFnd::INode**
- RPM_PER_SEC : **sFnd::INode**
- UNKNOWN : **sFnd::IInfo**

sFoundation 2.0

Example-MultiThreaded/Axis.cpp

Return to Example-MultiThreaded

Related Files:

- Main Function File: Example-MultiThreaded.cpp
- Axis Class Header File: Axis.h
- Supervisor Class Header File: Axis.h
- Supervisor Class Implementaion: Axis.cpp

```
//
// NAME
//      Axis
//
// DESCRIPTION:
//      This class holds the state machine and handles communication
//      directly with the ClearPath-SC node
//
//
     *
//*************************************************************************
     *******

//*************************************************************************
     *******
// NAME
     *
//  Axis.cpp headers
//
#include <stdio.h>
#include "Axis.h"
#include "Supervisor.h"
#if defined(_WIN32)||defined(_WIN64)
#include <windows.h>
#endif
//
     *
//*************************************************************************
     *******

//*************************************************************************
     *******
// NAME
     *
//  Axis.cpp defines
//
```

```
//
     *
//*********************************************************************
     *******
#if defined(_MSC_VER) && _MSC_VER < 1900
#define snprintf _snprintf
#endif

//*********************************************************************
     *******
//  NAME
     *
//      Axis::Axis
//
//  DESCRIPTION:
//      Constructor for the axis state
//
Axis::Axis(INode *node) :
    m_node(node),
    m_moveCount(0),
    m_quitting(false) {

    // Save the config file before starting
    // This would typically be performed right after tuning each axis.
    // Chosing a file name that contains the address and serial number
    // can ensure that the system does not get miswired.
    // Using the UserID as a filename key can allow swapping motors with
    // minimal setup required. This requires that no invalid characters
    // are present in the UserID.
    char tempPath[MAX_CONFIG_PATH_LENGTH];
    #if defined(_WIN32)||defined(_WIN64)
        GetTempPathA(MAX_CONFIG_PATH_LENGTH, tempPath);
    #else
        strncpy(tempPath, "/tmp", sizeof(tempPath));
    #endif
    snprintf(m_configFile, sizeof(m_configFile), "%s/config-%02d-%d.mtr",
     tempPath,
        m_node->Info.Ex.Addr(), m_node->Info.SerialNumber.Value());
    // sprintf(m_configFile, "%s\config-%s.mtr", tempPath, m_node->Info.User

    if (m_node->Setup.AccessLevelIsFull())
        m_node->Setup.ConfigSave(m_configFile);

};
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::~Axis
```

```
//
//  DESCRIPTION:
//      Destructor for the axis state
//
Axis::~Axis(){
    // Print out the move statistics one last time
    PrintStats();

    // If we don't have full access, there's nothing to do here
    if (!m_node->Setup.AccessLevelIsFull()){
        return;
    }

    // Disable the node and restore the config file
    attnReg attnReq;
    attnReq.cpm.Disabled = 1;
    m_node->Adv.Attn.ClearAttn(attnReq);
    m_node->EnableReq(false);

    m_node->Status.RT.Refresh();
    if (m_node->Status.RT.Value().cpm.Enabled){
        attnReg theAttn = m_node->Adv.Attn.WaitForAttn(attnReq, 3000);
        if (theAttn.cpm.Enabled){
            printf("Error: Timed out waiting for disable\n");
            return;
        }
    }
    // Restore the original config file
    // This would typically be performed when setting up the machine.
    m_node->Setup.ConfigLoad(m_configFile);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::Enable
//
//  DESCRIPTION:
//      Clear alerts and get the node ready to roll
//
void Axis::Enable(){
    if (m_node != NULL){
        // Clear alerts and node stops
        m_node->Status.AlertsClear();
        m_node->Motion.NodeStop(STOP_TYPE_ABRUPT);
        m_node->Motion.NodeStopClear();

        // Wait for node to be ready to take commands
        attnReg attnReq;
```

```
        attnReq.cpm.Ready = 1;
        m_node->Adv.Attn.ClearAttn(attnReq);

        m_node->EnableReq(true);

        // If the node is not currently ready, wait for it to get there
        if (!m_node->Status.IsReady()){
            attnReg theAttn = m_node->Adv.Attn.WaitForAttn(attnReq, 3000);
            if (!theAttn.cpm.Ready){
                mnErr eInfo;
                eInfo.ErrorCode = MN_ERR_TIMEOUT;
                eInfo.TheAddr = m_node->Info.Ex.Addr();
                snprintf(eInfo.ErrorMsg, sizeof(eInfo.ErrorMsg),
                    "Error: Timed out waiting for enable\n");
                throw eInfo;
            }
        }
        attnReq.cpm.Ready = 0;
        attnReq.cpm.WasHomed = 1;

        m_node->Adv.Attn.ClearAttn(attnReq);
        //If node is set up to home, start homing
        if (m_node->Motion.Homing.HomingValid()) {
            m_node->Motion.Homing.Initiate();

                attnReg theAttn = m_node->Adv.Attn.WaitForAttn(attnReq, 3000
                if (!theAttn.cpm.WasHomed) {
                    mnErr eInfo;
                    eInfo.ErrorCode = MN_ERR_TIMEOUT;
                    eInfo.TheAddr = m_node->Info.Ex.Addr();
                    snprintf(eInfo.ErrorMsg, sizeof(eInfo.ErrorMsg),
                        "Error: Timed out waiting for home\n");
                    throw eInfo;
                }
        }

    }
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//   NAME
     *
//       Axis::InitMotionParams
//
//   DESCRIPTION:
//       Initialize accLim, velLim, etc
//
void Axis::InitMotionParams(){
    // Set the user units to RPM and RPM/s
```

```
    m_node->VelUnit(INode::RPM);
    m_node->AccUnit(INode::RPM_PER_SEC);
    m_node->Motion.VelLimit = VEL_LIM_RPM;
    m_node->Motion.AccLimit = ACC_LIM_RPM_PER_SEC;
    m_node->Motion.JrkLimit = RAS_CODE;
    m_node->Limits.PosnTrackingLimit.Value(TRACKING_LIM);
    m_node->Limits.TrqGlobal.Value(TRQ_PCT);
    // Set up the move itself
    ::int32_t moveDuration;
#define USE_HEAD_TAIL_MOVES 1
#if USE_HEAD_TAIL_MOVES
    m_move.type = MG_MOVE_STYLE_HEADTAIL_TRIG;
    // Create a reference to advanced motion object to save typing
    IMotionAdv &motion = m_node->Motion.Adv;
    motion.HeadDistance = m_move.value/4;
    motion.TailDistance = m_move.value/4;
    motion.HeadTailVelLimit = m_node->Motion.VelLimit.Value() / 8;
    _mgPosnStyle theStyle(m_move.type);
    moveDuration = (::int32_t)motion.MovePosnHeadTailDurationMsec(m_move.val
                            theStyle.fld.relative == 0,
                            motion.HeadDistance.Value() > 0,
                            motion.TailDistance.Value() > 0);
#else
    m_move.type = MG_MOVE_STYLE_NORM_TRIG;
    _mgPosnStyle theStyle(m_move.type);
    moveDuration = (::int32_t)m_node->Motion.Adv.MovePosnDurationMsec(m_move
                            theStyle.fld.relative == 0);
#endif
    // Timeout for move: expected move duration + some time for synch
    between the nodes
    m_moveTimeoutMs = moveDuration + 20;
    //printf("  [%2d]: moveCounts=%d moveTimeout=%d\n", m_node-
    >Info.Ex.Addr, m_move.value, m_moveTimeoutMs);
}
//
    *
//*********************************************************************
    *******

//*********************************************************************
    *******
//  NAME
    *
//      Axis::Move
//
//  DESCRIPTION:
//      Issue the move
//
void Axis::Move(){
    // Clear out any lingering move done attentions
    attnReg attnMask;
    attnMask.cpm.MoveDone = 1;
    attnMask.cpm.Disabled = 1;
    attnMask.cpm.NotReady = 1;
```

```
    m_node->Adv.Attn.ClearAttn(attnMask);

    // Uncomment the following line to enable breakpoint debugging. The
     lock
    // allows the application to break without having timeout errors be thro
    // WARNING: While stepping through, leaving the scope this lock is
    // declared in (ie this function) will disengage the lock,
    // and may result in timeouts.
    //SysManager::ThreadLock lock;

    _mgPosnStyle theStyle(m_move.type);
#if USE_HEAD_TAIL_MOVES
    if (m_moveCount % 2 == 0){
        m_node->Motion.Adv.MovePosnStart(0, true);
    } else {
        m_node->Motion.Adv.MovePosnHeadTailStart(m_move.value,
            theStyle.fld.relative == 0, theStyle.fld.wait == 1,
            m_node->Motion.Adv.HeadDistance.Value() > 0,
            m_node->Motion.Adv.TailDistance.Value() > 0,
            theStyle.fld.dwell == 1);
    }
#else
    m_node->Motion.Adv.MovePosnStart(m_move.value,
        theStyle.fld.relative == 0, theStyle.fld.wait == 1,
        theStyle.fld.dwell == 1);
#endif
    //printf("  Start move axis %d: %f\n", m_node->Info.Ex.Addr(), infcCoreT
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::WaitForMove
//
//  DESCRIPTION:
//      Wait for attention that move has completed
//
bool Axis::WaitForMove(::int32_t timeoutMs){
    if (m_node == NULL){
        return false;
    }
    attnReg attnMask;
    attnReg attnSeen;
    attnMask.cpm.MoveDone = 1;
    attnMask.cpm.Disabled = 1;
    attnMask.cpm.NotReady = 1;
    attnSeen = m_node->Adv.Attn.WaitForAttn(attnMask, timeoutMs, false);
    //printf("  End move axis %d: %f\n", m_node->Info.Ex.Addr(), infcCoreTim
    if (attnSeen.cpm.Disabled){
```

```
        printf("  ERROR: [%d] disabled during move\n", m_node->Info.Ex.Addr(
        return false;
    }
    if (attnSeen.cpm.NotReady){
        printf("  ERROR: [%d] went NotReady during move\n", m_node->Info.Ex.
        return false;
    }
    return attnSeen.cpm.MoveDone;
};
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::AxisMain
//
//  DESCRIPTION:
//      The main work of the axis class - initialize the node and run
//      the state machine
//
void Axis::AxisMain(Supervisor *theSuper){
    bool moveDone;

    m_super = theSuper;

    // If we don't have nodes, exit
    if (m_node == NULL) {
        printf("Thread: No node, exiting\n");
        return;
    }

    try {
        attnReg attnMask;
        attnMask.cpm.MoveDone = 1;
        attnMask.cpm.Disabled = 1;
        attnMask.cpm.Enabled = 1;
        attnMask.cpm.Ready = 1;
        attnMask.cpm.WasHomed = 1;
        m_node->Adv.Attn.Mask = attnMask;

        // Get the node ready to go
        Enable();
        // Initialize the motion values
        InitMotionParams();
    }
    catch (mnErr &err) {
        fprintf(stderr, "Thread Setup link error: %s\n", err.ErrorMsg);
        m_super->Quit();
    }
    catch(...) {
```

```
        fprintf(stderr, "Thread setup failed generically.\n");
        m_super->Quit();
    }

    if (!m_quitting){
        // Perform homing here??

        // Zero out the measured position
        m_node->Motion.PosnMeasured.Refresh();
        double posn = m_node->Motion.PosnMeasured.Value();
        m_node->Motion.AddToPosition(-posn);
        m_node->Motion.PosnMeasured.Refresh();
        PrintStats(true);
    }

    m_state = STATE_IDLE;

    m_super->SignalReady(m_node->Info.Ex.Addr());

    // Start the machine cycling
    while (!m_quitting)  {
        try {

            // Save last state for debugging purposes
            m_lastState = m_state;

            switch (m_state) {
                case STATE_IDLE:
                    // Update idle state
                    m_super->SignalIdle(m_node->Info.Ex.Addr());
                    // Quitting?
                    if (m_quitting)
                        continue;
                    m_state = STATE_WAIT_FOR_KICK;
                    break;
                case STATE_WAIT_FOR_KICK:
                    m_super->WaitForKick();
                    // Out of here?
                    if (m_quitting)
                        continue;
                    // Supervisor restarted us
                    m_state = STATE_SEND_MOVE;
                    break;
                case STATE_SEND_MOVE:
                    // Initiate the motion
                    Move();
                    m_state = STATE_SIGNAL_MOVE_SENT;
                    break;
                case STATE_SIGNAL_MOVE_SENT:
                    m_super->SignalMoveSent(m_node->Info.Ex.Addr());
                    m_state = STATE_WAIT_FOR_MOVE_DONE;
                    break;
                case STATE_WAIT_FOR_MOVE_DONE:
                    moveDone = WaitForMove(m_moveTimeoutMs);
```

```
                    m_super->SignalMoveDone(m_node->Info.Ex.Addr());
                    if (!moveDone){
                        printf("ERROR: [%2d] timed out waiting for move
done\n", m_node->Info.Ex.Addr());
                        m_super->Quit();
                        return;
                    }
                    else{
                        m_moveCount++;
                        m_state = STATE_IDLE;
                    }
                    if (m_quitting)
                        continue;
                    break;
                default:
                    fprintf(stderr, "ThreadFunction: Illegal state");
                    return;
            }
        } // Try block
        catch (mnErr &err) {
            fprintf(stderr, "Thread link error: %s\n", err.ErrorMsg);
            m_super->Quit();
            // Send any signals that the supervisor might be waiting for
            m_super->SignalMoveSent(m_node->Info.Ex.Addr());
            m_super->SignalMoveDone(m_node->Info.Ex.Addr());
            ResetToIdle();
            continue;
        }
        catch(...) {
            fprintf(stderr, "Thread failed generically.\n");
            m_super->Quit();
            // Send any signals that the supervisor might be waiting for
            m_super->SignalMoveSent(m_node->Info.Ex.Addr());
            m_super->SignalMoveDone(m_node->Info.Ex.Addr());
            ResetToIdle();
        }
    } // while (!m_quitting)

    m_state = STATE_EXITED;
    ResetToIdle();
    m_super->SignalIdle(m_node->Info.Ex.Addr());
    printf("[%s] Thread Exited...\n", m_node->Info.UserID.Value());
    return;
};
//
     *
//*************************************************************************
     *******
```

sFoundation 2.0

Example-MultiThreaded/Example-MultiThreaded.cpp

Return to Example-MultiThreaded

Related Files:

- Axis Class Header File: Axis.h
- Axis Class Implementaion: Axis.cpp
- Supervisor Class Header File: Axis.h
- Supervisor Class Implementaion: Axis.cpp

```
//*********************************************************************
     *******

#include <stdio.h>
#include <ctime>
#include <chrono>
#include <string>
#include <iostream>
#include "Axis.h"
#include "Supervisor.h"

// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}

/*********************************************************************
     *****
*  Function Name: AttentionDetected
*    Description:   This is the port-level attention handler function.
*                   This handler simply prints out the attention informatior
*                   to the console window.
*     Parameters:
*          Input:   detected        - contains the attention information
*         Return:       none
*********************************************************************
     ****/
void MN_DECL AttentionDetected(const mnAttnReqReg &detected)
{
    // Make a local, non-const copy for printing purposes
    mnAttnReqReg myAttns = detected;
    // Create a buffer to hold the attentionReg information
    char attnStringBuf[512];
    // Load the buffer with the string representation of the attention infor
    myAttns.AttentionReg.StateStr(attnStringBuf, 512);
```

```
    // Print it out to the console
    printf("  --> ATTENTION: port %d, node=%d, attn=%s\n",
        detected.MultiAddr >> 4, detected.MultiAddr, attnStringBuf);
}


#if _MSC_VER
#pragma warning(disable:4996)
#endif
// A nice way of printing out the system time
string CurrentTimeStr() {
    time_t now = time(NULL);
    return string(ctime(&now));
}
#define CURRENT_TIME_STR CurrentTimeStr().c_str()
#if _MSC_VER
#pragma warning(default:4996)
#endif

int main(int argc, char* argv[])
{
    msgUser("Multithreaded Example starting. Press Enter to continue.");

    size_t portCount = 0;
    std::vector<std::string> comHubPorts;

    //Create the SysManager object. This object will coordinate actions
     among various ports
    // and within nodes. In this example we use this object to setup and
     open our port.
    SysManager myMgr;                           //Create System Manager myMg

    //This will try to open the port. If there is an error/exception
     during the port opening,
    //the code will jump to the catch loop where detailed information
     regarding the error will be displayed;
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount < comHubPorts.size() && portCount <
     NET_CONTROLLER_MAX; portCount++) {

            myMgr.ComHubPort(portCount, comHubPorts[portCount].c_str());
     //define the first SC Hub port (port 0) to be associated
                                                // with COM portnum (as seen
     in device manager)
        }

        if (portCount > 0) {
            //printf("\n I will now open port \t%i \n \n", portnum);
```

```
        myMgr.PortsOpen(portCount);             //Open the port

        for (size_t i = 0; i < portCount; i++) {
            IPort &myPort = myMgr.Ports(i);

            printf(" Port[%d]: state=%d, nodes=%d\n",
                myPort.NetNumber(), myPort.OpenState(), myPort.NodeCount
        }
    }
    else {
        printf("Unable to locate SC hub port\n");

        msgUser("Press any key to continue."); //pause so the user
can see the error message; waits for user to press a key

        return -1;  //This terminates the main program
    }

    // Create a list of axes - one per node
    vector<Axis*> listOfAxes;
    // Assume that the nodes are of the right type and that this app
has full control
    bool nodeTypesGood = true, accessLvlsGood = true;

    for (size_t iPort = 0; iPort < portCount; iPort++){
        // Get a reference to the port, to make accessing it easier
        IPort &myPort = myMgr.Ports(iPort);

        // Enable the attentions for this port
        myPort.Adv.Attn.Enable(true);
        // The attentions will be handled by the individual nodes,
but register
        // a handler at the port level, just for illustrative purposes.
        myPort.Adv.Attn.AttnHandler(AttentionDetected);

        for (unsigned iNode = 0; iNode < myPort.NodeCount(); iNode++){

            // Get a reference to the node, to make accessing it easier
            INode &theNode = myPort.Nodes(iNode);
            theNode.Motion.AccLimit = 100000;

            // Make sure we are talking to a ClearPath SC
            if (theNode.Info.NodeType() != IInfo::CLEARPATH_SC_ADV) {
                printf("---> ERROR: Uh-oh! Node %d is not a
ClearPath-SC Advanced Motor\n", iNode);
                nodeTypesGood = false;
            }

            if (nodeTypesGood) {
                // Create an axis for this node
                listOfAxes.push_back(new Axis(&theNode));

                if (!theNode.Setup.AccessLevelIsFull()) {
```

```
                        printf("---> ERROR: Oh snap! Access level is not
    good for node %u\n", iNode);
                        accessLvlsGood = false;
                    }
                    else {
                        // Set the move distance based on where it is in
    the network
                        listOfAxes.at(iNode)->SetMoveRevs((iNode + 1) * 2);
                        // Set the trigger group indicator
                        theNode.Motion.Adv.TriggerGroup(1);
                    }
                }

            }

#if 0
            // Set the last node in the ring to a longer move
            listOfAxes.at(listOfAxes.size()-1)->SetMoveRevs(listOfAxes.size(
#endif
        }

        // If we have full access to the nodes and they are all
    ClearPath-SC advanced nodes,
        // then continue with the example
        if (nodeTypesGood && accessLvlsGood){

            // Create the supervisor thread, giving it access to the list
    of axes
            Supervisor theSuper(listOfAxes, myMgr);
            theSuper.CreateThread();

            printf("\nMachine starting: %s\n", CURRENT_TIME_STR);

            // Everything is running - wait for the user to press a key
    to end it
            msgUser("Press any key to stop the machine."); //pause so the
    user can see the error message; waits for user to press a key

            printf("Machine stopping: %s\n", CURRENT_TIME_STR);

            // Tell the supervisor to stop
            theSuper.Quit();
            theSuper.Terminate();

            printf("\nFinalStats:\tnumMoves\t\tPosition\n");
        }
        else{
            // If something is wrong with the nodes, tell the user about
    it and quit
            if (!nodeTypesGood){
                printf("\n\tFAILURE: Please attach only ClearPath-SC
    Advanced nodes.\n\n");
            }
            else if (!accessLvlsGood){
```

```
                printf("\n\tFAILURE: Please get full access on all your node
            }
        }

        // Delete the list of axes that were created
        for (size_t iAxis = 0; iAxis < listOfAxes.size(); iAxis++){
            delete listOfAxes.at(iAxis);
        }

        // Close down the ports
        myMgr.PortsClose();

    }
    catch (mnErr& theErr) {
        fprintf(stderr, "Caught error: addr=%d, err=0x%0x\nmsg=%s\n",
                theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
        printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
                theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
        msgUser("Press any key to continue."); //pause so the user can
     see the error message; waits for user to press a key
        return(2);
    }
    catch (...) {
        fprintf(stderr, "Error generic caught\n");
        printf("Generic error caught\n");
        msgUser("Press any key to continue."); //pause so the user can
     see the error message; waits for user to press a key
        return(3);
    }

    // Good-bye
    msgUser("Press any key to continue."); //pause so the user can see
     the error message; waits for user to press a key
    return 0;
}
//
     *
//*********************************************************************
     *******
```

sFoundation 2.0

Example-MultiThreaded/Supervisor.cpp

Return to Example-MultiThreaded

Related Files:

- Main Function File: Example-MultiThreaded.cpp
- Axis Class Header File: Axis.h
- Axis Class Implementaion: Axis.cpp
- Supervisor Class Header File: Axis.h

```
//
// NAME
//      Supervisor
//
// DESCRIPTION:
//      This machine initializes all the nodes and runs the state machine
//      that synchronizes the moves among the nodes; the synchronization
//      is simply waiting for all nodes to send their moves to their nodes,
//      triggering them all at the same time, then waiting for them all to
//      finish before issuing any more moves
//
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
// NAME
     *
//  Supervisor.cpp headers
//
#include "Supervisor.h"
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Supervisor::Supervisor
//
//  DESCRIPTION:
//      Constructor for the supervisor state
//
```

```
Supervisor::Supervisor(vector<Axis*> theAxes, SysManager &myMgr)
    : m_sysMgr(myMgr)
{
    m_axes = theAxes;
    m_state = SUPER_IDLE;
    m_quitting = false;
    // create list of flags to keep track of the nodes for
    // various conditions
    for (Uint16 iAxis = 0; iAxis < m_axes.size(); iAxis++){
        m_axisIdle.push_back(false);
        m_axisReady.push_back(false);
        m_axisSentMove.push_back(false);
        m_axisDone.push_back(false);
    }
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Supervisor::SignaleIdle
//
//  DESCRIPTION:
//      Nodes signal to the supervisor when they are idle
//
void Supervisor::SignalIdle(Uint16 nodeNum){
    SignalPerNode(m_axisIdle, m_nodesIdle, nodeNum);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Supervisor::SignalReady
//
//  DESCRIPTION:
//      Nodes signal to the supervisor when they are ready to go
//
void Supervisor::SignalReady(Uint16 nodeNum){
    SignalPerNode(m_axisReady, m_nodesReady, nodeNum);
}
//
     *
//*********************************************************************
     *******
```

```
//*********************************************************************
     *******
//  NAME
     *
//      Supervisor::WaitForKick
//
//  DESCRIPTION:
//      Nodes will wait to be kicked
//
void Supervisor::WaitForKick(){
    WaitForCondition(m_nodesKicked);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Supervisor::SignalMoveSent
//
//  DESCRIPTION:
//      Nodes signal that they have sent their moves
//
void Supervisor::SignalMoveSent(Uint16 nodeNum){
    SignalPerNode(m_axisSentMove, m_nodesMoved, nodeNum);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Supervisor::SignalMoveDone
//
//  DESCRIPTION:
//      Nodes signal to the supervisor when their move is done
//
void Supervisor::SignalMoveDone(Uint16 nodeNum){
    SignalPerNode(m_axisDone, m_nodesDone, nodeNum);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
```

```
//  NAME
    *
//      Supervisor::SuperMain
//
//  DESCRIPTION:
//      The Supervisor's state machine thread; tells each Axis to create its
//      own thread and initialize itself, then runs the state machine to
//      tell the Axis objects when to send their moves, triggers the moves
//      after they've all been sent, and waits for all the nodes to
    finish
//      their move before starting the process again.
//
void Supervisor::SuperMain(){
    // Get all conditions to a known state
    ResetCondition(m_axisIdle, m_nodesIdle);
    ResetCondition(m_axisSentMove, m_nodesMoved);
    ResetCondition(m_axisDone, m_nodesDone);

    try{
        // Create threads for all the axes
        for (Uint16 iAxis = 0; iAxis < m_axes.size(); iAxis++){
            //printf(" axis[%d]: userID=%s\n", iAxis, theAxes.at(iAxis)-
    >MyNode()->Info.UserID.Value());
            m_axes.at(iAxis)->CreateThread(this);
        }
    }
    catch(...){
        printf("Supervisor failed initializing threads\n");
        Quit();
    }

    // Get the state machine going
    while (!m_quitting){
        try {
            // For debugging
            m_lastState = m_state;
            switch (m_state){
                case SUPER_IDLE:
                    // Reset each node to idle
                    for (Uint16 iAxis = 0; iAxis < m_axes.size(); iAxis++)
                        m_axes.at(iAxis)->ResetToIdle();
                    m_state = SUPER_WAIT_FOR_IDLE;
                    break;
                case SUPER_WAIT_FOR_IDLE:
                    if (m_quitting)
                        continue;
                    // Wait for the nodes to go idle
                    WaitForCondition(m_nodesIdle);
                    if (m_quitting)
                        continue;
                    ResetCondition(m_axisIdle, m_nodesIdle);
                    // Everyone is waiting; kick all the nodes
                    SetCondition(m_nodesKicked);
                    m_state = SUPER_WAIT_FOR_MOVES_SENT;
```

```
                    break;
                case SUPER_WAIT_FOR_MOVES_SENT:
                    // Wait for them all to say their moves have been sent
                    WaitForCondition(m_nodesMoved);
                    if (m_quitting)
                        continue;
                    ResetCondition(m_nodesKicked);
                    ResetCondition(m_axisSentMove, m_nodesMoved);
                    m_axes.at(0)->MyNode()->Port.Adv.TriggerMovesInGroup(1);
                    m_state = SUPER_WAIT_FOR_MOVES_DONE;
                    break;
                case SUPER_WAIT_FOR_MOVES_DONE:
                    WaitForCondition(m_nodesDone);
                    PrintStats();
                    // Take a break
                    m_sysMgr.Delay(200);
                    if (m_quitting)
                        continue;
                    ResetCondition(m_axisDone, m_nodesDone);
                    m_state = SUPER_WAIT_FOR_IDLE;
                    break;
                default:
                    printf("Super: Undefined state %d\n", m_state);
                    return;
            }
        } // Try block
        catch (mnErr &err) {
            fprintf(stderr, "Supervisor thread failed: %s\n", err.ErrorMsg);
            Quit();
        }
        catch(...) {
            fprintf(stderr, "Supervisor thread failed.\n");
            Quit();
        }
        m_sysMgr.Delay(0);
    }

    SetCondition(m_nodesKicked);
    WaitForCondition(m_nodesIdle);

    m_state = SUPER_EXITED;
    // Wait for threads to end
    for (Uint16 iAxis = 0; iAxis < m_axes.size(); iAxis++){
        m_axes.at(iAxis)->Join();
    }
    return;
}
//
     *
//*********************************************************************
     *******
```

Example-SingleThreaded/Axis.cpp

Return to Example-SingleThreaded

Related Files:

- Main Function File: Example-SingleThreaded.cpp
- Axis Class Header File: Axis.h

```
//
// NAME
//      Axis
//
// DESCRIPTION:
//      This class holds the state machine and handles communication
//      directly with the ClearPath-SC node
//
     *
//************************************************************************
     *******

//************************************************************************
     *******
// NAME
     *
//  Axis.cpp headers
//
#include <stdio.h>
#include "Axis.h"
//
     *
//************************************************************************
     *******

//************************************************************************
     *******
// NAME
     *
//  Axis.cpp defines
//
//
     *
//************************************************************************
     *******
#if defined(_MSC_VER) && _MSC_VER < 1900
#define snprintf _snprintf
#endif
```

```
//*********************************************************************
     *******
//   NAME
     *
//       Axis::Axis
//
//   DESCRIPTION:
//       Constructor for the axis state
//
Axis::Axis(SysManager& SysMgr, INode *node) :
    m_node(node),
    m_moveCount(0),
    m_positionWrapCount(0),
    m_quitting(false),
    m_sysMgr (SysMgr) {

    // OPTIONAL - Save the config file before starting
    // This would typically be performed right after tuning each axis.
    // Chosing a file name that contains the address and serial number
    // can ensure that the system does not get miswired.
    // Using the UserID as a filename key can allow swapping motors with
    // minimal setup required. This requires that no invalid characters
    // are present in the UserID.
#if defined(_WIN32) || defined(_WIN64)
    char tempPath[MAX_CONFIG_PATH_LENGTH];
    GetTempPathA(MAX_CONFIG_PATH_LENGTH, tempPath);
    snprintf(m_configFile, sizeof(m_configFile), "%s\\config-%02d-
     %d.mtr", tempPath,
        m_node->Info.Ex.Addr(), m_node->Info.SerialNumber.Value());
#else
    snprintf(m_configFile, sizeof(m_configFile), "/tmp/config-%02d-%d.mtr",
            m_node->Info.Ex.Addr(), m_node->Info.SerialNumber.Value());
#endif

    if (m_node->Setup.AccessLevelIsFull())
        m_node->Setup.ConfigSave(m_configFile);
};
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//   NAME
     *
//       Axis::~Axis
//
//   DESCRIPTION:
//       Destructor for the axis state; disable the node and restore the save
//       config file
//
Axis::~Axis() {
    // Print out the move statistics one last time
```

```
    PrintStats();

    // If we don't have full access, there's nothing to do here
    if (!m_node->Setup.AccessLevelIsFull()) {
        return;
    }

    // Disable the node and wait for it to disable
    m_node->EnableReq(false);

    // Poll the status register to confirm the node's disable
    time_t timeout;
    timeout = time(NULL) + 3;
    m_node->Status.RT.Refresh();
    while (m_node->Status.RT.Value().cpm.Enabled) {
        if (time(NULL) > timeout) {
            printf("Error: Timed out waiting for disable\n");
            return;
        }
        m_node->Status.RT.Refresh();
    };

    // Restore the original config file
    m_node->Setup.ConfigLoad(m_configFile);
}
//
     *
//*************************************************************************
     *******

//*************************************************************************
     *******
//   NAME
     *
//       Axis::Enable
//
//   DESCRIPTION:
//       Clear alerts and get the node ready to roll
//
void Axis::Enable() {
    if (m_node != NULL) {
        // Clear alerts and node stops
        m_node->Status.AlertsClear();
        m_node->Motion.NodeStop(STOP_TYPE_ABRUPT);
        m_node->Motion.NodeStop(STOP_TYPE_CLR_ALL);

        // Enable the node
        m_node->EnableReq(true);

        // If the node is not currently ready, wait for it to get there
        time_t timeout;
        timeout = time(NULL) + 3;
        // Basic mode - Poll until disabled
        while (!m_node->Status.IsReady()) {
```

```
            if (time(NULL) > timeout) {
                printf("Error: Timed out waiting for enable\n");
                return;
            }
        };

        //If node is set up to home, start homing
        if (m_node->Motion.Homing.HomingValid()) {
            m_node->Motion.Homing.Initiate();
            printf("Homing...");
            time_t timeout;
            timeout = time(NULL) + 10000;
            // Basic mode - Poll until disabled
            while (!m_node->Motion.Homing.WasHomed()) {
                if (time(NULL) > timeout) {
                    printf("Error: Timed out waiting for homing\n");
                    return;
                }
            };
            printf("Completed\n");
        }
    }
}
//
    *
//*********************************************************************
    *******

//*********************************************************************
    *******
//  NAME
    *
//      Axis::InitMotionParams
//
//  DESCRIPTION:
//      Initialize accLim, velLim, etc
//
void Axis::InitMotionParams() {
    // Set the user units to RPM and RPM/s
    m_node->VelUnit(INode::RPM);
    m_node->AccUnit(INode::RPM_PER_SEC);

    // Set the motion's limits
    m_node->Motion.VelLimit = VEL_LIM_RPM;
    m_node->Motion.AccLimit = ACC_LIM_RPM_PER_SEC;
    m_node->Motion.JrkLimit = RAS_CODE;
    m_node->Limits.PosnTrackingLimit.Value(TRACKING_LIM);
    m_node->Limits.TrqGlobal.Value(TRQ_PCT);

    // Set the dwell used by the motor when applicable
    m_node->Motion.DwellMs = 5;
}
//
    *
```

```
//*********************************************************************
     *******

//*********************************************************************
     *******
//   NAME
     *
//       Axis::ChangeMotionParams
//
//   DESCRIPTION:
//       Modify velLimit, AccLimit, etc.
//
void Axis::ChangeMotionParams(double newVelLimit,
    double newAccelLimit,
    uint32_t newJrkLimit,
    uint32_t newTrackingLimit,
    int32_t newTrqGlobal) {
    // Set the motion's limits
    m_node->Motion.VelLimit = newVelLimit;
    m_node->Motion.AccLimit = newAccelLimit;
    m_node->Motion.JrkLimit = newJrkLimit;
    m_node->Limits.PosnTrackingLimit.Value(newTrackingLimit);
    m_node->Limits.TrqGlobal.Value(newTrqGlobal);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//   NAME
     *
//       Axis::PosnMove
//
//   DESCRIPTION:
//       Issue a position move
//
void Axis::PosnMove(int32_t targetPosn, bool isAbsolute, bool addDwell) {
    // Save the move's calculated time for timeout handling
    m_moveTimeoutMs = (int32_t)m_node->Motion.MovePosnDurationMsec(targetPos
        isAbsolute);
    m_moveTimeoutMs += 20;

    // Check for number-space wrapping. With successive moves, the
     position may
    // pass the 32-bit number cap (+ or -). The check below looks for
     that and
    // updates our counter to be used with GetPosition
    if (!isAbsolute) {
        if (GetPosition(false) + targetPosn > INT32_MAX) {
            m_positionWrapCount++;
        }
        else if (GetPosition(false) + targetPosn < INT32_MIN) {
```

```
            m_positionWrapCount--;
        }
    }

    // Issue the move to the node
    m_node->Motion.MovePosnStart(targetPosn, isAbsolute, addDwell);
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::VelMove
//
//  DESCRIPTION:
//      Issue a velocity move
//
void Axis::VelMove(double targetVel) {
    // Issue the move to the node
    m_node->Motion.MoveVelStart(targetVel);

    // Move at the set velocity for half a second
    m_sysMgr.Delay(500);

    // Issue a node stop, forcing any movement (like a
    // velocity move) to stop. There are several
    // different types of node stops
    m_node->Motion.NodeStop(STOP_TYPE_ABRUPT);
    //m_node->Motion.NodeStop(STOP_TYPE_RAMP);

    // Configure the timeout for the node stop to settle
    m_moveTimeoutMs = 100;
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::WaitForMove
//
//  DESCRIPTION:
//      Wait for attention that move has completed
//
bool Axis::WaitForMove(::int32_t timeoutMs, bool secondCall) {
    if (m_node == NULL) {
        return false;
```

```
    }

    // Wait for the proper calculated move time
    time_t timeout;
    timeout = time(NULL) + timeoutMs;

    m_node->Status.RT.Refresh();
    // Basic mode - Poll until move done
    while (!m_node->Status.RT.Value().cpm.MoveDone) {
        m_node->Status.RT.Refresh();
        if (time(NULL) > timeout) {
            printf("Error: Timed out during move\n");
            return false;
        }

        // Catch specific errors that can occur during a move
        if (m_node->Status.RT.Value().cpm.Disabled) {
            printf("  ERROR: [%d] disabled during move\n",
                m_node->Info.Ex.Addr());
            return false;
        }
        if (m_node->Status.RT.Value().cpm.NotReady) {
            printf("  ERROR: [%d] went NotReady during move\n",
                m_node->Info.Ex.Addr());
            return false;
        }
    }

    return m_node->Status.RT.Value().cpm.MoveDone;
};
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//  NAME
     *
//      Axis::AxisMain
//
//  DESCRIPTION:
//      The main work of the axis class - initialize the node and run
//      the state machine
//
void Axis::AxisMain() {
    bool moveDone;

    // If we don't have nodes, exit
    if (m_node == NULL) {
        printf("No node, exiting\n");
        return;
    }
```

```
    try {
        // Get the node ready to go
        Enable();
        // Initialize the motion values
        InitMotionParams();
    }
    catch (mnErr &err) {
        fprintf(stderr, "Setup error: %s\n", err.ErrorMsg);
    }
    catch (...) {
        fprintf(stderr, "Setup failed generically.\n");
    }

    if (!m_quitting) {
        // If your node has appropriate hardware (hardstop/home switch)
        // and is configured in ClearView, uncomment the line below to
        // initiate homing
        //theNode.Motion.Homing.Initiate();

        // Set the hardware brake to allow motion
        m_node->Port.BrakeControl.BrakeSetting(0, BRAKE_ALLOW_MOTION);

        // Zero out the commanded position
        m_node->Motion.PosnCommanded.Refresh();
        double posn = m_node->Motion.PosnCommanded.Value();
        m_node->Motion.AddToPosition(-posn);
        m_node->Motion.PosnCommanded.Refresh();
        PrintStats(true);
    }

    m_state = STATE_IDLE;

    // Start the machine cycling
    while (!m_quitting) {
        try {
            // Save last state for debugging purposes
            m_lastState = m_state;

            // Check for alerts
            CheckForAlerts();

            switch (m_state) {
            case STATE_IDLE:
                // Update idle state
                // Quitting?
                if (m_quitting)
                    continue;
                m_state = STATE_SEND_MOVE;
                break;
            case STATE_SEND_MOVE:
                // Initiate the motion
                if (m_moveCount % 2 == 0) {
                    PosnMove(5000);
                }
```

```
                else {
                    //Set new motion parameters before the next move
                    ChangeMotionParams(1500, 7500, 4,
                        m_node->Info.PositioningResolution.Value() / 4, 95);

                    VelMove(50);
                }

                // Reset our motion parameters.  The parameters for the
                // next move could be entered instead to get a head start
                InitMotionParams();

                m_state = STATE_WAIT_FOR_MOVE_DONE;
                break;
            case STATE_WAIT_FOR_MOVE_DONE:
                //Check if the move is done yet
                moveDone = WaitForMove(m_moveTimeoutMs);
                if (!moveDone) {
                    printf("ERROR: [%2d] timed out waiting for move done\n",
                        m_node->Info.Ex.Addr());
                    return;
                }
                else {
                    m_moveCount++;
                    m_state = STATE_IDLE;
                }
                if (m_moveCount == 4) {
                    // Stop once we've properly demonstrated movement
                    Quit();
                }
                PrintStats();
                if (m_quitting)
                    continue;
                break;
            default:
                fprintf(stderr, "Illegal state");
                return;
            }
        } // Try block
        catch (mnErr &err) {
            fprintf(stderr, "Link error: %s\n", err.ErrorMsg);
            ResetToIdle();
            continue;
        }
        catch (...) {
            fprintf(stderr, "Node failed generically.\n");
            ResetToIdle();
        }
    }

    m_state = STATE_EXITED;
    ResetToIdle();
    printf("[%s] Quitting node...\n", m_node->Info.UserID.Value());
    return;
```

```
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//   NAME
     *
//       Axis::CheckForAlerts
//
//   DESCRIPTION:
//       Checks the status register if there are alerts present.
//       If there are, print some information about them and quit.
//
void Axis::CheckForAlerts()
{
    // Refresh our relevant registers
    m_node->Status.Alerts.Refresh();
    m_node->Status.RT.Refresh();

    //Check for alerts
    if (m_node->Status.RT.Value().cpm.AlertPresent) {
        char alertList[256];
        m_node->Status.Alerts.Value().StateStr(alertList, 256);
        printf("Node %d has alerts! Alert: %s",
            m_node->Info.Ex.Addr(), alertList);
        m_quitting = true;
    }
}
//
     *
//*********************************************************************
     *******

//*********************************************************************
     *******
//   NAME
     *
//       Axis::SetBrakeControl
//
//   DESCRIPTION:
//       Setup the hardware brake control on the node's port. Set's the indic
//       brake's mode to the passed-in value
//
void Axis::SetBrakeControl(size_t brakeNum, BrakeControls brakeMode)
{
    m_node->Port.BrakeControl.BrakeSetting(brakeNum, brakeMode);
}
//
     *
//*********************************************************************
     *******
```

```
//*********************************************************************
     *******
//   NAME
     *
//       Axis::GetPosition
//
//   DESCRIPTION:
//       Return the node's commanded position scaled properly to wraps of the
//       number space.
//
int64_t Axis::GetPosition(bool includeWraps)
{
    // Create a variable to return and refresh the position
    long long scaledPosn;
    m_node->Motion.PosnCommanded.Refresh();

    // If there have been no number wraps, just return the position
    scaledPosn = int64_t(m_node->Motion.PosnCommanded.Value());

    if (includeWraps) {
        scaledPosn += int64_t(m_positionWrapCount) << 32;
    }

    return scaledPosn;
}
//
     *
//*********************************************************************
     *******
```

sFoundation 2.0

Example-SingleThreaded/Example-SingleThreaded.cpp

Return to Example-SingleThreaded

Related Files:

- Axis Class Header File: Axis.h
- Axis Class Implementaion: Axis.cpp

```
//*****************************************************************
     *************

#include <stdio.h>
#include <ctime>
#include <string>
#include <iostream>
#include "Axis.h"

// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}

/*****************************************************************
     ***********
*  Function Name: main
*   Description:    The main function for this single threaded
     example.
*                   One command-line argument is expected, which is
     the COM
*                   port number for where the SC network is attached.
*       Return:     0 if successful; non-zero if there was a problem
*****************************************************************
     **********/
int main(int argc, char* argv[])
{
    msgUser("Single-threaded Example starting. Press Enter to continue

    size_t portCount = 0;
    size_t returnVal = 0;
    std::vector<std::string> comHubPorts;
```

```
    //Create the SysManager object. This object will coordinate
    actions among various ports
    // and within nodes. In this example we use this object to
    setup and open our port.
    SysManager myMgr;                           //Create System
    Manager myMgr

    //This will try to open the port. If there is an
    error/exception during the port opening,
    //the code will jump to the catch loop where detailed
    information regarding the error will be displayed;
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount < comHubPorts.size() &&
    portCount < NET_CONTROLLER_MAX; portCount++) {

            myMgr.ComHubPort(portCount,
    comHubPorts[portCount].c_str());    //define the first SC Hub
    port (port 0) to be associated
                                            // with COM portnum (as
    seen in device manager)
        }

        if (portCount > 0) {
            //printf("\n I will now open port \t%i \n \n", portnum);
            myMgr.PortsOpen(portCount);             //Open the port

            for (size_t i = 0; i < portCount; i++) {
                IPort &myPort = myMgr.Ports(i);

                printf(" Port[%d]: state=%d, nodes=%d\n",
                    myPort.NetNumber(), myPort.OpenState(), myPort.Nod
            }
        }
        else {
            printf("Unable to locate SC hub port\n");

            msgUser("Press any key to continue."); //pause so the
    user can see the error message; waits for user to press a key

            returnVal = -1;  //This terminates the main program
        }

        // Create a list of axes - there will be one Axis per node
        vector<Axis*> listOfAxes;
```

```
    // Assume that the nodes are of the right type and that
this app has full control
    bool nodeTypesGood = true, accessLvlsGood = true;

    for (size_t iPort = 0; iPort < portCount; iPort++) {
        // Get a reference to the port, to make accessing it easie
        IPort &myPort = myMgr.Ports(iPort);

        for (unsigned iNode = 0; iNode < myPort.NodeCount();
iNode++) {
            // Get a reference to the node, to make accessing
it easier
            INode &theNode = myPort.Nodes(iNode);

            // Make sure we are talking to a ClearPath SC
(advanced or basic model will work)
            if (theNode.Info.NodeType() != IInfo::CLEARPATH_SC_ADV
                && theNode.Info.NodeType() !=
IInfo::CLEARPATH_SC) {
                printf("---> ERROR: Uh-oh! Node %d is not a
ClearPath-SC Motor\n", iNode);
                nodeTypesGood = false;
            }

            if (nodeTypesGood) {
                // Create an axis for this node
                listOfAxes.push_back(new Axis(myMgr, &theNode));

                // Make sure we have full access
                if (!theNode.Setup.AccessLevelIsFull()) {
                    printf("---> ERROR: Oh snap! Access level
is not good for node %u\n", iNode);
                    accessLvlsGood = false;
                }
            }
        }
    }

    // If we have full access to the nodes and they are all
ClearPath-SC nodes,
    // then continue with the example
    if (nodeTypesGood && accessLvlsGood) {
        for (Uint16 iAxis = 0; iAxis < listOfAxes.size();
iAxis++) {
            // Tell each axis to do its thing
            listOfAxes.at(iAxis)->AxisMain();
        }
    }
    else {
```

```
            // If something is wrong with the nodes, tell the user
    about it
            if (!nodeTypesGood) {
                printf("\n\tFAILURE: Please attach only ClearPath-
    SC nodes.\n\n");
                returnVal = -5;
            }
            else if (!accessLvlsGood) {
                printf("\n\tFAILURE: Please get full access on all
    your nodes.\n\n");
                returnVal = -6;
            }
        }

        // Delete the list of axes that were created
        for (size_t iAxis = 0; iAxis < listOfAxes.size(); iAxis++) {
            delete listOfAxes.at(iAxis);
        }

        // Close down the ports
        myMgr.PortsClose();

    }
    catch (mnErr& theErr) {
        fprintf(stderr, "Caught error: addr=%d, err=0x%0x\nmsg=%s\n",
            theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
        printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
            theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
        returnVal = -3;
    }
    catch (...) {
        fprintf(stderr, "Error generic caught\n");
        printf("Generic error caught\n");
        returnVal = -4;
    }

    // Good-bye
    msgUser("Press any key to continue."); //pause so the user can
     see the error message; waits for user to press a key
    return returnVal;
}
//
     *
//*****************************************************************
     *************
```

sFoundation 2.0

sFnd::_mnErr Member List

This is the complete list of members for **sFnd::_mnErr**, including all inherited members.

ErrorCode	**sFnd::_mnErr**
ErrorMsg	**sFnd::_mnErr**
TheAddr	**sFnd::_mnErr**

sFoundation 2.0

Example-GPIO/Example-GPIO.cpp

Return to Example-GPIO

```
/*********************************************************************
     ******/
/*********************************************************************
     ******/

#include <stdio.h>
#include <string>
#include <iostream>
#include "pubSysCls.h"

using namespace sFnd;
using namespace std;

// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}

int main(int argc, char* argv[])
{
    msgUser("GPIO Example starting. Press Enter to continue.");
size_t portCount = 0;
    std::vector<std::string> comHubPorts;

    //Create the SysManager object. This object will coordinate actions
     among various ports
    // and within nodes. In this example we use this object to setup and
     open our port.
    SysManager myMgr;                           //Create System Manager myMg

    //This will try to open the port. If there is an error/exception
     during the port opening,
    //the code will jump to the catch loop where detailed information
     regarding the error will be displayed;
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount < comHubPorts.size() && portCount <
     NET_CONTROLLER_MAX; portCount++) {
```

```
        myMgr.ComHubPort(portCount, comHubPorts[portCount].c_str());
//define the first SC Hub port (port 0) to be associated
                                        // with COM portnum (as seen
in device manager)
    }

    if (portCount > 0) {
        //printf("\n I will now open port \t%i \n \n", portnum);
        myMgr.PortsOpen(portCount);             //Open the port

        for (size_t i = 0; i < portCount; i++) {
            IPort &myPort = myMgr.Ports(i);

            printf(" Port[%d]: state=%d, nodes=%d\n",
                myPort.NetNumber(), myPort.OpenState(), myPort.NodeCount
        }
    }
    else {
        printf("Unable to locate SC hub port\n");

        msgUser("Press any key to continue."); //pause so the user
can see the error message; waits for user to press a key

        return -1;  //This terminates the main program
    }

    //Once the code gets past this point, it can be assumed that the
Port has been opened without issue
    //Now we can get a reference to our port object which we will use
to access the node objects

    for (size_t iPort = 0; iPort < portCount; iPort++) {
        // Get a reference to the port, to make accessing it easier
        IPort &myPort = myMgr.Ports(iPort);



        // Print out some information about the port
        printf(" Port[%d]: state=%d, nodes=%d\n", myPort.NetNumber(),
myPort.OpenState(), myPort.NodeCount());

        // Set GPO to ON for the first brake control,
        myPort.BrakeControl.BrakeSetting(0, GPO_ON);
        // Can set GPO to OFF for the second brake control using the
method below
        //myPort.BrakeControl.BrakeSetting(1, GPO_OFF);

        for (size_t iBrake = 0; iBrake < 2; iBrake++) {
            // Print out brake mode for the first brake
            size_t controlMode = myPort.BrakeControl.BrakeSetting(iBrake
            if (controlMode == 3) {
                printf("\nPort[%d] Brake 0 control mode: GPO ON", myPort
            }
```

```
                else if (controlMode == 4) {
                    printf("\nPort[%d] Brake 0 control mode: GPO OFF", myPor
                }
                else {
                    printf("\nPort[%d] Brake 0 control mode: GPO not
set", myPort.NetNumber());
                }

                // keep the console output readable
                printf("\n");
            }

            printf("\n");

            for (unsigned iNode = 0; iNode < myPort.NodeCount(); iNode++)
{

                INode &theNode = myPort.Nodes(iNode);

                //Checking Inputs to each Node
                if (theNode.Status.RT.Value().cpm.InA)
                {
                    printf("Node %i Input A is asserted \n", iNode);
                }
                else
                {
                    printf("Node %i Input A is not asserted \n", iNode);
                }
                if (theNode.Status.RT.Value().cpm.InB)
                {
                    printf("Node %i Input B is asserted \n", iNode);
                }
                else
                {
                    printf("Node %i Input B is not asserted\n", iNode);
                }

            }

            printf("\nShutting down network\n");

            // Close down the ports
            myMgr.PortsClose();

        }
    }
    catch (mnErr& theErr) {
        fprintf(stderr, "Caught error: addr=%d, err=0x%0x\nmsg=%s\n",
                theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
        printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
                theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
        msgUser("Press any key to continue."); //pause so the user can
see the error message; waits for user to press a key
        // keep the console output readable
```

```
        printf("\n");
        return(2);
    }
    catch (...) {
        fprintf(stderr, "Error generic caught\n");
        printf("Generic error caught\n");
        msgUser("Press any key to continue."); //pause so the user can
    see the error message; waits for user to press a key
        // keep the console output readable
        printf("\n");
        return(3);
    }

    // Good-bye
    msgUser("Press any key to continue."); //pause so the user can see
    the error message; waits for user to press a key
    // keep the console output readable
    printf("\n");
    return 0;
}
//
    *
//*********************************************************************
    *******
```

sFoundation 2.0

Example-Homing/Example-Homing.cpp

Return to Example-Homing

```
//Required include files
#include <stdio.h>
#include <string>
#include <iostream>
#include "pubSysCls.h"

using namespace sFnd;

// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}

//**********************************************************************
     **********
//This program will load configuration files onto each node connected to
     the port, then executes
//sequential repeated moves on each axis.
//**********************************************************************
     **********

#define CHANGE_NUMBER_SPACE 2000    //The change to the numberspace after
     homing (cnts)
#define TIME_TILL_TIMEOUT   10000   //The timeout used for homing(ms)


int main(int argc, char* argv[])
{
    msgUser("Homing Example starting. Press Enter to continue.");

size_t portCount = 0;
    std::vector<std::string> comHubPorts;

    //Create the SysManager object. This object will coordinate actions
     among various ports
    // and within nodes. In this example we use this object to setup and
     open our port.
    SysManager myMgr;                           //Create System Manager myMg

    //This will try to open the port. If there is an error/exception
     during the port opening,
    //the code will jump to the catch loop where detailed information
     regarding the error will be displayed;
```

```
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount < comHubPorts.size() && portCount <
    NET_CONTROLLER_MAX; portCount++) {

            myMgr.ComHubPort(portCount, comHubPorts[portCount].c_str());
    //define the first SC Hub port (port 0) to be associated
                                                // with COM portnum (as seen
    in device manager)
        }

        if (portCount > 0) {
            //printf("\n I will now open port \t%i \n \n", portnum);
            myMgr.PortsOpen(portCount);             //Open the port

            for (size_t i = 0; i < portCount; i++) {
                IPort &myPort = myMgr.Ports(i);

                printf(" Port[%d]: state=%d, nodes=%d\n",
                    myPort.NetNumber(), myPort.OpenState(), myPort.NodeCount
            }
        }
        else {
            printf("Unable to locate SC hub port\n");

            msgUser("Press any key to continue."); //pause so the user
    can see the error message; waits for user to press a key

            return -1;  //This terminates the main program
        }


        //Once the code gets past this point, it can be assumed that the
    Port has been opened without issue
        //Now we can get a reference to our port object which we will use
    to access the node

        for (size_t iPort = 0; iPort < portCount; iPort++) {
            // Get a reference to the port, to make accessing it easier
            IPort &myPort = myMgr.Ports(iPort);

            for (unsigned iNode = 0; iNode < myPort.NodeCount(); iNode++)
    {

                //Here we identify the first Node, enable and home the
    node, then adjust the position reference

                        // Create a shortcut reference for the first node
                INode &theNode = myPort.Nodes(iNode);
```

```
            //theNode.EnableReq(false);             //Ensure Node is
disabled before starting

            printf("   Node[%d]: type=%d\n", iNode, theNode.Info.NodeTyp
            printf("            userID: %s\n", theNode.Info.UserID.Value
            printf("        FW version: %s\n", theNode.Info.FirmwareVers
            printf("          Serial #: %d\n",
theNode.Info.SerialNumber.Value());
            printf("             Model: %s\n", theNode.Info.Model.Value(

            //The following statements will attempt to enable the
node.  First,
            // any shutdowns or NodeStops are cleared, finally the
node in enabled
            theNode.Status.AlertsClear();                   //Clear
Alerts on node
            theNode.Motion.NodeStopClear(); //Clear Nodestops on Node
            theNode.EnableReq(true);                    //Enable node

            double timeout = myMgr.TimeStampMsec() +
TIME_TILL_TIMEOUT; //define a timeout in case the node is unable to enal

//This will loop checking on the Real time values of the node's
Ready status
            while (!theNode.Motion.IsReady()) {
                if (myMgr.TimeStampMsec() > timeout) {
                    printf("Error: Timed out waiting for Node %d to
enable\n", iNode);
                    msgUser("Press any key to continue."); //pause so
the user can see the error message; waits for user to press a key
                    return -2;
                }
            }
            //At this point the Node is enabled, and we will now
check to see if the Node has been homed
            //Check the Node to see if it has already been homed,
            if (theNode.Motion.Homing.HomingValid())
            {
                if (theNode.Motion.Homing.WasHomed())
                {
                    printf("Node %d has already been homed, current
position is: \t%8.0f \n", iNode, theNode.Motion.PosnMeasured.Value());
                    printf("Rehoming Node... \n");
                }
                else
                {
                    printf("Node [%d] has not been homed.  Homing
Node now...\n", iNode);
                }
                //Now we will home the Node
                    theNode.Motion.Homing.Initiate();
```

```
                    timeout = myMgr.TimeStampMsec() +
TIME_TILL_TIMEOUT;    //define a timeout in case the node is unable
to enable
                  // Basic mode - Poll until disabled
                    while (!theNode.Motion.Homing.WasHomed()) {
                        if (myMgr.TimeStampMsec() > timeout) {
                            printf("Node did not complete homing:
\n\t -Ensure Homing settings have been defined through ClearView.
\n\t -Check for alerts/Shutdowns \n\t -Ensure timeout is longer than
the longest possible homing move.\n");
                            msgUser("Press any key to continue.");
//pause so the user can see the error message; waits for user to
press a key
                            return -2;
                        }
                    }
                    theNode.Motion.PosnMeasured.Refresh();
//Refresh our current measured position
                    printf("Node completed homing, current position:
\t%8.0f \n", theNode.Motion.PosnMeasured.Value());
                    printf("Soft limits now active\n");

                    printf("Adjusting Numberspace by %d\n", CHANGE_NUMBE


theNode.Motion.AddToPosition(CHANGE_NUMBER_SPACE);          //Now
the node is no longer considered "homed, and soft limits are turned off

                    theNode.Motion.Homing.SignalComplete();
//reset the Node's "sense of home" soft limits (unchanged) are now
active again

                    theNode.Motion.PosnMeasured.Refresh();
//Refresh our current measured position
                    printf("Numberspace changed, current position:
\t%8.0f \n", theNode.Motion.PosnMeasured.Value());
                }
                else {
                    printf("Node[%d] has not had homing setup through
ClearView.  The node will not be homed.\n", iNode);
                }

                printf("Disabling Node\n");
                theNode.EnableReq(false);
            }
        }
    }
catch (mnErr& theErr)
{
    //This statement will print the address of the error, the error
code (defined by the mnErr class),
    //as well as the corresponding error message.
    printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
```

```
        msgUser("Press any key to continue."); //pause so the user can
    see the error message; waits for user to press a key

        return 0;  //This terminates the main program
    }


    // Close down the ports
    myMgr.PortsClose();

    msgUser("Press any key to continue."); //pause so the user can see
     the error message; waits for user to press a key
    return 0;           //End program
}
```

sFoundation 2.0

Example-Motion/Example-Motion.cpp

Return to Example-Motion

```
//Required include files
#include <stdio.h>
#include <string>
#include <iostream>
#include "pubSysCls.h"

using namespace sFnd;

// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}

//*********************************************************************
    ***************
//This program will load configuration files onto each node
    connected to the port, then executes
//sequential repeated moves on each axis.
//*********************************************************************
    ***************

#define ACC_LIM_RPM_PER_SEC 100000
#define VEL_LIM_RPM         700
#define MOVE_DISTANCE_CNTS  10000
#define NUM_MOVES           5
#define TIME_TILL_TIMEOUT   10000   //The timeout used for homing(ms)

int main(int argc, char* argv[])
{
    msgUser("Motion Example starting. Press Enter to continue.");

size_t portCount = 0;
    std::vector<std::string> comHubPorts;


    //Create the SysManager object. This object will coordinate
    actions among various ports
    // and within nodes. In this example we use this object to setup
    and open our port.
```

```
    SysManager myMgr;                           //Create System
    Manager myMgr

    //This will try to open the port. If there is an error/exception
    during the port opening,
    //the code will jump to the catch loop where detailed
    information regarding the error will be displayed;
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount < comHubPorts.size() &&
    portCount < NET_CONTROLLER_MAX; portCount++) {

            myMgr.ComHubPort(portCount,
    comHubPorts[portCount].c_str());    //define the first SC Hub
    port (port 0) to be associated
                                            // with COM portnum (as
    seen in device manager)
        }

        if (portCount < 0) {

            printf("Unable to locate SC hub port\n");

            msgUser("Press any key to continue."); //pause so the
    user can see the error message; waits for user to press a key

            return -1;  //This terminates the main program
        }
        //printf("\n I will now open port \t%i \n \n", portnum);
        myMgr.PortsOpen(portCount);             //Open the port

        for (size_t i = 0; i < portCount; i++) {
            IPort &myPort = myMgr.Ports(i);

            printf(" Port[%d]: state=%d, nodes=%d\n",
                myPort.NetNumber(), myPort.OpenState(), myPort.NodeCount



            //Once the code gets past this point, it can be assumed
    that the Port has been opened without issue
            //Now we can get a reference to our port object which we
    will use to access the node objects

            for (size_t iNode = 0; iNode < myPort.NodeCount();
    iNode++) {
```

```
            // Create a shortcut reference for a node
            INode &theNode = myPort.Nodes(iNode);

            theNode.EnableReq(false);               //Ensure
Node is disabled before loading config file

            myMgr.Delay(200);


            //theNode.Setup.ConfigLoad("Config File path");


            printf("   Node[%d]: type=%d\n", int(iNode),
theNode.Info.NodeType());
            printf("            userID: %s\n",
theNode.Info.UserID.Value());
            printf("        FW version: %s\n", theNode.Info.Firmware
            printf("          Serial #: %d\n",
theNode.Info.SerialNumber.Value());
            printf("             Model: %s\n", theNode.Info.Model.Va

            //The following statements will attempt to enable
the node.  First,
            // any shutdowns or NodeStops are cleared, finally
the node is enabled
            theNode.Status.AlertsClear();
//Clear Alerts on node
            theNode.Motion.NodeStopClear(); //Clear Nodestops on
Node
            theNode.EnableReq(true);                    //Enable
node
            //At this point the node is enabled
            printf("Node \t%zi enabled\n", iNode);
            double timeout = myMgr.TimeStampMsec() +
TIME_TILL_TIMEOUT; //define a timeout in case the node is unable
to enable

//This will loop checking on the Real time values of the node's
Ready status
            while (!theNode.Motion.IsReady()) {
                if (myMgr.TimeStampMsec() > timeout) {
                    printf("Error: Timed out waiting for Node %d
to enable\n", iNode);
                    msgUser("Press any key to continue.");
//pause so the user can see the error message; waits for user to
press a key
                    return -2;
                }
            }
            //At this point the Node is enabled, and we will now
check to see if the Node has been homed
```

```
            //Check the Node to see if it has already been
homed,
            if (theNode.Motion.Homing.HomingValid())
            {
                if (theNode.Motion.Homing.WasHomed())
                {
                    printf("Node %d has already been homed,
current position is: \t%8.0f \n", iNode,
theNode.Motion.PosnMeasured.Value());
                    printf("Rehoming Node... \n");
                }
                else
                {
                    printf("Node [%d] has not been homed.
Homing Node now...\n", iNode);
                }
                //Now we will home the Node
                theNode.Motion.Homing.Initiate();

                timeout = myMgr.TimeStampMsec() +
TIME_TILL_TIMEOUT;    //define a timeout in case the node is
unable to enable

// Basic mode - Poll until disabled
                while (!theNode.Motion.Homing.WasHomed()) {
                    if (myMgr.TimeStampMsec() > timeout) {
                        printf("Node did not complete homing:
\n\t -Ensure Homing settings have been defined through
ClearView. \n\t -Check for alerts/Shutdowns \n\t -Ensure timeout
is longer than the longest possible homing move.\n");
                        msgUser("Press any key to continue.");
//pause so the user can see the error message; waits for user to
press a key
                        return -2;
                    }
                }
                printf("Node completed homing\n");
            }
            else {
                printf("Node[%d] has not had homing setup
through ClearView.  The node will not be homed.\n", iNode);
            }

        }

        //At this point we will execute 10 rev moves
sequentially on each axis

        for (size_t i = 0; i < NUM_MOVES; i++)
        {
```

```
            for (size_t iNode = 0; iNode < myPort.NodeCount();
iNode++) {
                // Create a shortcut reference for a node
                INode &theNode = myPort.Nodes(iNode);

                theNode.Motion.MoveWentDone();
//Clear the rising edge Move done register

                theNode.AccUnit(INode::RPM_PER_SEC);
//Set the units for Acceleration to RPM/SEC
                theNode.VelUnit(INode::RPM);
//Set the units for Velocity to RPM
                theNode.Motion.AccLimit = ACC_LIM_RPM_PER_SEC;
//Set Acceleration Limit (RPM/Sec)
                theNode.Motion.VelLimit = VEL_LIM_RPM;
//Set Velocity Limit (RPM)

                printf("Moving Node \t%zi \n", iNode);

theNode.Motion.MovePosnStart(MOVE_DISTANCE_CNTS);
//Execute 10000 encoder count move
                printf("%f estiomated time.\n", theNode.Motion.MoveI
                double timeout = myMgr.TimeStampMsec() +
theNode.Motion.MovePosnDurationMsec(MOVE_DISTANCE_CNTS) + 100;
//define a timeout in case the node is unable to enable

                while (!theNode.Motion.MoveIsDone()) {
                    if (myMgr.TimeStampMsec() > timeout) {
                        printf("Error: Timed out waiting for
move to complete\n");
                        msgUser("Press any key to continue.");
//pause so the user can see the error message; waits for user to
press a key
                        return -2;
                    }
                }
                printf("Node \t%zi Move Done\n", iNode);
            } // for each node
        } // for each move



    //After moves have completed Disable node, and close ports
        printf("Disabling nodes, and closing port\n");
        //Disable Nodes

        for (size_t iNode = 0; iNode < myPort.NodeCount();
iNode++) {
            // Create a shortcut reference for a node
            myPort.Nodes(iNode).EnableReq(false);
        }
```

```
        }
    }
    catch (mnErr& theErr)
    {
        printf("Failed to disable Nodes n\n");
        //This statement will print the address of the error, the
    error code (defined by the mnErr class),
        //as well as the corresponding error message.
        printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
    theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);

        msgUser("Press any key to continue."); //pause so the user
    can see the error message; waits for user to press a key
        return 0;  //This terminates the main program
    }

    // Close down the ports
    myMgr.PortsClose();

    msgUser("Press any key to continue."); //pause so the user can
    see the error message; waits for user to press a key
    return 0;           //End program
}
```

sFoundation 2.0

Example-StatusAlerts/Example-StatusAlerts.cpp

Return to **Example-StatusAlerts**

```
//*********************************************************
     *********************

#include <stdio.h>
#include <string>
#include <iostream>
#include "pubSysCls.h"

using namespace sFnd;
using namespace std;


// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}


int main(int argc, char* argv[])
{
    msgUser("Status Alerts Example starting. Press Enter
     to continue.");
    size_t portCount = 0;
    std::vector<std::string> comHubPorts;

    //Create the SysManager object. This object will
     coordinate actions among various ports
    // and within nodes. In this example we use this
     object to setup and open our port.
    SysManager myMgr;                           //Create
     System Manager myMgr
```

```
    //This will try to open the port. If there is an
    error/exception during the port opening,
    //the code will jump to the catch loop where detailed
    information regarding the error will be displayed;
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount < comHubPorts.size()
    && portCount < NET_CONTROLLER_MAX; portCount++) {

            myMgr.ComHubPort(portCount,
    comHubPorts[portCount].c_str());    //define the first
    SC Hub port (port 0) to be associated
                                                // with COM
    portnum (as seen in device manager)
        }

        if (portCount > 0) {
            //printf("\n I will now open port \t%i \n \n",
    portnum);
            myMgr.PortsOpen(portCount);             //Open
    the port

            for (size_t i = 0; i < portCount; i++) {
                IPort &myPort = myMgr.Ports(i);

                printf(" Port[%d]: state=%d, nodes=%d\n",
                    myPort.NetNumber(),
    myPort.OpenState(), myPort.NodeCount());
            }
        }
        else {
            printf("Unable to locate SC hub port\n");

            msgUser("Press any key to continue."); //pause
    so the user can see the error message; waits for user
    to press a key

            return -1;  //This terminates the main program
```

```
    }


    for (size_t iPort = 0; iPort < portCount; iPort++)
{
        // Get a reference to the port, to make
accessing it easier
        IPort &myPort = myMgr.Ports(iPort);

        char alertList[256];

        printf("Checking for Alerts: \n");

        for (unsigned iNode = 0; iNode <
myPort.NodeCount(); iNode++) {
            // Get a reference to the node, to make
accessing it easier
            INode &theNode = myPort.Nodes(iNode);

            // make sure our registers are up to date
            theNode.Status.RT.Refresh();
            theNode.Status.Alerts.Refresh();

            printf("---------\n");
            printf(" Checking node %i for Alerts:\n", iNo

            // Check the status register's
"AlertPresent" bit
            // The bit is set true if there are alerts
in the alert register
            if
(!theNode.Status.RT.Value().cpm.AlertPresent) {
                printf("   Node has no alerts!\n");
            }
            //Check to see if the node experienced
torque saturation
            if (theNode.Status.HadTorqueSaturation()) {
                printf("      Node has experienced
torque saturation since last checking\n");
            }
            // get an alert register reference, check
the alert register directly for alerts
```

```
            if
(theNode.Status.Alerts.Value().isInAlert()) {
                // get a copy of the alert register
bits and a text description of all bits set

theNode.Status.Alerts.Value().StateStr(alertList, 256);
                printf("   Node has alerts!
Alerts:\n%s\n", alertList);

                // can access specific alerts using
the method below
                if
(theNode.Status.Alerts.Value().cpm.Common.EStopped) {
                    printf("      Node is e-stopped:
Clearing E-Stop\n");
                    theNode.Motion.NodeStopClear();
                }
                if
(theNode.Status.Alerts.Value().cpm.TrackingShutdown) {
                    printf("      Node exceeded
Tracking error limit\n");
                }



                // Check for more alerts and Clear Alerts
                theNode.Status.Alerts.Refresh();
                if
(theNode.Status.Alerts.Value().isInAlert()) {

theNode.Status.Alerts.Value().StateStr(alertList, 256);
                    printf("      Node has non-estop
alerts: %s\n", alertList);
                    printf("      Clearing non-serious
alerts\n");
                    theNode.Status.AlertsClear();

                    // Are there still alerts?
                    theNode.Status.Alerts.Refresh();
                    if
(theNode.Status.Alerts.Value().isInAlert()) {

theNode.Status.Alerts.Value().StateStr(alertList, 256);
```

```
                            printf("   Node has serious,
non-clearing alerts: %s\n", alertList);
                        }
                        else {
                            printf("   Node %d: all alerts
have been cleared\n", theNode.Info.Ex.Addr());
                        }
                    }
                    else {
                        printf("   Node %d: all alerts
have been cleared\n", theNode.Info.Ex.Addr());
                    }

                }
                printf("=========\n");
            }

            printf("\nShutting down network\n");

            // Close down the ports
            myMgr.PortsClose();
        }
    }
    catch (mnErr& theErr) {
        fprintf(stderr, "Caught error: addr=%d, err=0x%0x\nms
                theErr.TheAddr, theErr.ErrorCode, theErr.Erro
        printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
                theErr.TheAddr, theErr.ErrorCode, theErr.Erro
        msgUser("Press any key to continue.");
        // keep the console output readable
        printf("\n");
        return(2);
    }
    catch (...) {
        fprintf(stderr, "Error generic caught\n");
        printf("Generic error caught\n");
        msgUser("Press any key to continue.");
        // keep the console output readable
        printf("\n");
        return(3);
    }

    // Good-bye
```

```
    msgUser("Press any key to continue.");
    // keep the console output readable
    printf("\n");
    return 0;
}
//
     *
//*********************************************************
     **********************
```

sFoundation 2.0

HelloWorld/HelloWorld.cpp

Return to HelloWorld

```
//Required include files
#include <stdio.h>
#include <iostream>
#include "pubSysCls.h"

using namespace sFnd;

// Send message and wait for newline
void msgUser(const char *msg) {
    std::cout << msg;
    getchar();
}

/*****************************************************
     ************************
*   Description:    This program is the bare minimum
     to open a port and identify the number of nodes
*       on the port.
*
*
*       Return:     0 if the system was initialized prope
*                   -1 if there was a problem
     accessing the port.
*****************************************************
     ************************/

int main(int argc, char* argv[]){
    size_t portCount = 0;
    std::vector<std::string> comHubPorts;
```

```
    msgUser("Hello World Example starting. Press Enter
    to continue.");

    //Create the SysManager object. This object will
    coordinate actions among various ports
    // and within nodes. In this example we use this
    object to setup and open our port.
    SysManager myMgr;
    //Create System Manager myMgr

    printf("Hello World, I am SysManager\n");
    //Print to console

    //This will try to open the port. If there is an
    error/exception during the port opening,
    //the code will jump to the catch loop where
    detailed information regarding the error will be disp
    //otherwise the catch loop is skipped over
    try
    {

        SysManager::FindComHubPorts(comHubPorts);
        printf("Found %d SC Hubs\n", comHubPorts.size());

        for (portCount = 0; portCount <
    comHubPorts.size() && portCount <
    NET_CONTROLLER_MAX; portCount++) {

            myMgr.ComHubPort(portCount,
    comHubPorts[portCount].c_str());    //define the
    first SC Hub port (port 0) to be associated
                                            // with
    COM portnum (as seen in device manager)
        }

        if (portCount > 0) {
            //printf("\n I will now open port \t%i \n
    \n", portnum);
            myMgr.PortsOpen(portCount);
    //Open the port
```

```
            for (size_t i = 0; i < portCount; i++) {
                IPort &myPort = myMgr.Ports(i);

                printf(" Port[%d]: state=%d, nodes=%d\n",
                    myPort.NetNumber(),
myPort.OpenState(), myPort.NodeCount());
            }
            // Close down the ports
            myMgr.PortsClose();
        }
        else {
            printf("Unable to locate SC hub port\n");

            msgUser("Press any key to continue.");
//pause so the user can see the error message;
waits for user to press a key

            return -1;  //This terminates the main progra
        }
    }
    catch(mnErr& theErr)    //This catch statement
will intercept any error from the Class library
    {
        printf("Port Failed to open, Check to ensure
correct Port number and that ClearView is not
using the Port\n");
        //This statement will print the address of the
error, the error code (defined by the mnErr
class),
        //as well as the corresponding error message.
        printf("Caught error: addr=%d,
err=0x%08x\nmsg=%s\n", theErr.TheAddr,
theErr.ErrorCode, theErr.ErrorMsg);

        msgUser("Press any key to continue."); //pause
so the user can see the error message; waits for
user to press a key

        return -1;  //This terminates the main program
    }
```

```
    msgUser("Press any key to continue."); //pause so
     the user can see the error message; waits for user
     to press a key

    return 0;            //End program
}
//
```

sFoundation 2.0

Classes | Namespaces | Macros | Typedefs | Enumerations

pubNetAPI.h File Reference

Network Constants and types for interacting with nodes. More...

Go to the source code of this file.

Classes

struct	**sFnd::_mnErr** Error code and reason structure. More...
struct	**_ShutdownInfo** Information to setup group shutdowns at a node. More...

Namespaces

sFnd

Macros

#define	**MN_UNSET_ADDR** **multiaddr**(0xffffu) The numeric value for an unassigned multiaddr.

Typedefs

typedef Uint16	**multiaddr**
typedef Uint16	**nodeaddr**
typedef Uint16	**netaddr**
typedef enum **_openStates**	**openStates** This enumeration describes the port open state. More...
typedef enum **_portTypes**	**portTypes** The type of nodes attached to this port. More...
typedef enum **_BrakeControls**	**BrakeControls** State for the brake control feature.
typedef struct **sFnd::_mnErr**	**sFnd::mnErr** Error code and reason structure. More...
typedef struct **_ShutdownInfo**	**ShutdownInfo** Information to setup group shutdowns at a node.

Enumerations

enum **_openStates** {
UNKNOWN, **CLOSED**, **FLASHING**, **PORT_UNAVAILABLE**, **OPENED_SEARCHING**, **OPENED_ONLINE**
}
This enumeration describes the port open state. More...

enum **_portTypes** { **UNKNOWN_NODES**, **MERIDIAN_CONMOD**, **CPM_COMHUB**, **GENERIC_SERIAL** }

The type of nodes attached to this port. More...

enum **_BrakeControls** { **BRAKE_AUTOCONTROL**, **BRAKE_PREVENT_MOTION**, **BRAKE_ALLOW_MOTION**, **GPO_ON**, **GPO_OFF** }

State for the brake control feature. More...

Typedef Documentation

typedef Uint16 multiaddr

The internally assigned unique address type for a node.

The system initialization assigns this address for it's internal purposes. It should only be used for error messages and the arguments of legacy C-based API calls.

typedef Uint16 netaddr

The serial port index type.

This corresponds to the index supplied during system initialization.

typedef Uint16 nodeaddr

The internally assigned index type of a node on a serial port.

Node are numbered from zero, starting from the transmit port of the host.

typedef enum _openStates openStates

Operations to nodes may be performed when the port is in the OPENED_ONLINE state.

typedef enum _portTypes portTypes

This list defines the types of adapters attached directly to the serial port. For ClearPath-SC systems this is usually the CPM_COMHUB.

Enumeration Type Documentation

enum _BrakeControls

Enumerator	
BRAKE_AUTOCONTROL	Brake activates automatically depending on the state of the node. BrakeNum 0 will correspond to Brake 1 on the SC hub. BrakeNum 1 will correspond to Brake 2 on the SC hub
BRAKE_PREVENT_MOTION	Engage the brake.
BRAKE_ALLOW_MOTION	Disengage the brake
GPO_ON	Use the brake output as a GPO that is asserted.
GPO_OFF	Use the brake output as a GPO that is deasserted.

enum _openStates

Operations to nodes may be performed when the port is in the OPENED_ONLINE state.

Enumerator	
UNKNOWN	We do not know the current state
CLOSED	A previously opened port was shutdown.
FLASHING	The port is opened in FLASH mode.

PORT_UNAVAILABLE	The port requested does not exist or is currently owned by another application.
OPENED_SEARCHING	The port is opened and initial detection of nodes resulted in no detected nodes. Communications with nodes will result in failures.
OPENED_ONLINE	The port is opened and we have detected nodes. Interactions with nodes can occur normally.

enum _portTypes

This list defines the types of adapters attached directly to the serial port. For ClearPath-SC systems this is usually the CPM_COMHUB.

Enumerator	
UNKNOWN_NODES	Unspecified nodes
MERIDIAN_CONMOD	Meridian Family CON-MOD connection
CPM_COMHUB	ClearPath-SC Communications Hub connection
GENERIC_SERIAL	Node connected via generic serial connection

sFoundation 2.0

pubNetAPI.h

Go to the documentation of this file.

```
//*****************************************************************************
// $Archive: /ClearPath SC-1.0.123/User Driver/inc/pubNetAPI.h $
// $Revision: 30 $ $Date: 12/09/16 4:05p $
// $Workfile: pubNetAPI.h $
//
// DESCRIPTION:
// CREATION DATE:
//      2015-12-02 16:32:18
//
// COPYRIGHT NOTICE:
//      (C)Copyright 2015  Teknic, Inc.  All rights reserved.
//
//      This copyright notice must be reproduced in any copy, modification,
//      or portion thereof merged into another program. A copy of the
//      copyright notice must be included in the object library of a user
//      program.
//
*
//*****************************************************************************
#ifndef __PUBNETAPI_H__
#define __PUBNETAPI_H__
```

```
28
29
30 //*****************************************
   ***********************************
31 // NAME
   *
32 //  pubNetAPI.h headers
33 //
34     #include "tekTypes.h"
35
36 #ifndef __TI_COMPILER_VERSION__
37     #include "pubCoreRegs.h"
38     #include "mnErrors.h"
39     #include "pubIscRegs.h"
40 #endif
41     #include <string.h>
42 //
   *
43 //*****************************************
   ***********************************
44
45
46
47
48 //*****************************************
   ***********************************
49 // NAME
   *
50 //  pubNetAPI.h constants
51 //
52 //
53 // - - - - - - - - - - - - - - - - - - - - -
   - - - - - - - - - - - - - - - - - -
54 // NETWORK DEFINITIONS
55 // - - - - - - - - - - - - - - - - - - - - -
   - - - - - - - - - - - - - - - - - -
63 typedef Uint16 multiaddr;       // Combined
   net and node address
```

```
 69 typedef Uint16 nodeaddr;        // Node's address on its net
 75 typedef Uint16 netaddr;         // Net number (a.k.a. cNum)
 77 #define MN_UNSET_ADDR multiaddr(0xffffu) // The illegal address value
 78
 79 #define NET_CONTROLLER_MAX 3
 81 #define MN_API_MAX_NODES        16U
 83
 85 #define MN_USER_NV_SIZE     13U
 86
 95 typedef enum _openStates {
 99     UNKNOWN,
103     CLOSED,
107     FLASHING,
112     PORT_UNAVAILABLE,
118     OPENED_SEARCHING,
123     OPENED_ONLINE
124 } openStates;
125 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
126 //                          NODE/NET INFORMATION V2
127 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
128
129
130 #ifndef __TI_COMPILER_VERSION__
131 //*****************************************************************************
132 //
139 typedef enum _portTypes {
143     UNKNOWN_NODES,
147     MERIDIAN_CONMOD,
151     CPM_COMHUB,
155     GENERIC_SERIAL
```

```
156 } portTypes;
157 //
    *
158 //*****************************************
    ***********************************
159
161 //*****************************************
    ***********************************
162 //
    *
167 typedef struct _controllerSpec {
168     netRates PortRate;
169 #if defined(_WIN32)||defined(_WIN64)
170     Uint16 PortNumber;
171 #else
172     char PortName[MAX_PATH];
173 #endif
174 #ifdef __cplusplus
175     _controllerSpec() {
176         PortRate = MN_BAUD_108X;
177     }
178     #if defined(_WIN32)||defined(_WIN64)
179
184         _controllerSpec(Uint16 number,
    netRates rate = MN_BAUD_108X) {
185             PortNumber = number;
186             PortRate = rate;
187         }
188     #else
189
194         _controllerSpec(const char * name,
    netRates rate = MN_BAUD_108X);
195     #endif
196 #endif
197 } controllerSpec;
198 //
    *
```

```
199 //*****************************************
    ***********************************
202 //*****************************************
    ***********************************
203 //
    *
211 struct MN_EXPORT _portSpec {
212 #if defined(_WIN32)||defined(_WIN64)
213     netRates PortRate;
214     Uint16 PortNumber;
215     portTypes PortType;
216 #else
217     netRates PortRate;
218     char PortName[MAX_PATH];
219     portTypes PortType;
220 #endif // defined(_WIN32)||defined(_WIN64)
221 #ifdef __cplusplus
222     #if defined(_WIN32)||defined(_WIN64)
223     // - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - -
224     // WINDOWS CONSTRUCTION
225     // - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - -
227         _portSpec() {
229             PortRate = MN_BAUD_12X;
230             PortNumber = 0;
231             PortType = UNKNOWN_NODES;
232         }
234         _portSpec operator=(const
    _controllerSpec &oldForm) {
235             PortRate = oldForm.PortRate;
236             PortNumber = oldForm.PortNumber;
237             PortType = MERIDIAN_CONMOD;
238             return(*this);
239         }
248         _portSpec(Uint16 number,
249                   portTypes thePortType,
```

```
250                     netRates rate =
    MN_BAUD_108X) {
251             PortNumber = number;
252             PortType = thePortType;
253         }
254     #else
255     // - - -
```

```
290
292 //
    *
293 //*************************************
    ***********************************
294
295
296
297
298
299
300
301 //*************************************
    ***********************************
302 // NAME
303 //  BrakeControl enum
307 enum _BrakeControls
308 {
312     BRAKE_AUTOCONTROL,
316     BRAKE_PREVENT_MOTION,
320     BRAKE_ALLOW_MOTION,
324     GPO_ON,
328     GPO_OFF
329 };
331 typedef enum _BrakeControls BrakeControls;
332 //
    *
333 //*************************************
    ***********************************
334
335
336
337
338
339 #ifndef __TI_COMPILER_VERSION__
340 // - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - -
341 // Standard Error Object
```

```
342 // - - -
```

```
539 #endif // __TI_COMPILER_VERSION__
540
541 // - - -
```

```
//==========================================================================
//  END OF FILE pubNetAPI.h
//==========================================================================
```

sFnd::IAttnNode Member List

This is the complete list of members for **sFnd::IAttnNode**, including all inherited members.

AlertMask	**sFnd::IAttnNode**
ClearAttn(mnStatusReg attnClr)	**sFnd::IAttnNode**
Mask	**sFnd::IAttnNode**
Node()	**sFnd::IObjWithNode**
StatusMask	**sFnd::IAttnNode**
Supported()	**sFnd::IObj**
WaitForAttn(mnStatusReg theAttn, int32_t timeoutMsec, bool autoClear=true)	**sFnd::IAttnNode**
~IAttnNode() (defined in **sFnd::IAttnNode**)	**sFnd::IAttnNode**

Attentions/Interupts for Event Driven Applications

For multi-threaded applications where polling on status events is not feasiable, Advanced ClearPath SC node's offer a more complex "interrupt-like" *Attention Notification* method. For high performance applications, and ones that desire a more central event collection, a multi-threaded attention implementation is generally required to achieve the best performance.

Multi-Threaded Implementation

The most efficient way to be notified of events is to set up the nodes to automatically send data back to the host computer when an event happens. This feature is available to multi-threaded applications. The steps to set up the system to send back attentions are the following:

- Enable attention processing at the port level **sFnd::IAttnPort::Enable**
- Set the desired bits the Node's **sFnd::IAttnNode::Mask** register to enable sending asserting status events to the host.

Once the system is configured to send back Attentions the application must use one of the Attention handling functions provided through the Class Library.

- For applications with a de-centralized Architecture where each node has it's own thread, use the **sFnd::IAttnNode::WaitForAttn** function to prossess attentions by node.
- For applications that desire more centralized event collection you can register an Attention Callback function(**sFnd::IAttnPort::AttnHandler**) to manage all attentions on a Port.

See also

sFnd::IAttnPort for information on enableing attentions, and registering an Attention handler callback function

sFnd::IAttnNode for information on how to set up the Attention masks at the Node level to determine what events trigger attentions

cpmStatusRegFlds for a complete list of status events

The following code sample illustrates this technique.

Coding Example

```
//\/////////////////////////////////////////////////////
    ///////////
//Example showing processing attentions at the node level
// Enable the attentions for this port
myPort.Adv.Attn.Enable(true);

// Set the Node's Attention Mask to generate attentions
    on "Enabled"
attnReg attnReq;
```

```
attnReq.cpm.Enabled = 1;
mynode.Adv.Attn.Mask = attnReg;

// Clear any old attentions and attempt to enable the ndoe
mynode.Adv.Attn.ClearAttn(attnReq);
mynode.EnableReq(true);

//Wait for enable attention, check to make sure we
    didn't time out.
ValueStatus theAttn =
    mynode.Adv.Attn.WaitForAttn(attnReq, 3000);
if (theAttn.isClear()){
    printf("Error: Timed out waiting for enable\n");
    return;
}

//\////////////////////////////////////////////////////
    ////////////////'
// Example showing a Attention Handler function
// Enable the attentions for this port
myPort.Adv.Attn.Enable(true);
// The attentions will be handled by the individual
    nodes, but register
// a handler at the port level, just for illustrative purp
myPort.Adv.Attn.AttnHandler(AttentionDetected);

void MN_DECL AttentionDetected(const mnAttnReqReg &detecte
{
    // Make a local, non-const copy for printing purposes
    mnAttnReqReg myAttns = detected;
    // Create a buffer to hold the attentionReg informatio
    char attnStringBuf[512];
    // Load the buffer with the string representation
    of the attention information
    myAttns.AttentionReg.StateStr(attnStringBuf, 512);
    // Print it out to the console
    printf("  --> ATTENTION: port %d, node=%d, attn=%s\n",
        detected.MultiAddr >> 4, detected.MultiAddr, attnS
}
```

sFoundation 2.0

pubCpmRegs.h File Reference

S-Foundation register definitions for programming ClearPath-SC nodes. More...

Go to the source code of this file.

Classes

struct	**_cpmStatusRegFlds** ClearPath-SC Node Status Register Fields. More...
struct	**_cpmAlertFlds** ClearPath-SC Alert and Warning Register Field Definitions. More...

Typedefs

typedef struct _cpmHwConfigFlds	**cpmHwConfigFlds** ClearPath-SC Hardware Configuration Field Definitions.
typedef struct **_cpmStatusRegFlds**	**cpmStatusRegFlds** ClearPath-SC Status Register Field definitions.

typedef struct **_cpmAlertFlds**	**cpmAlertFlds** ClearPath-SC Alert and Warning Register Field Definitions.

Variables

BFld	**CapturePolarityHiSpd** High-Speed Position Capture control. More...
BFld	**CapturePolarity** Position Capture control. More...
BFld	**DataAcqOversampling** Data acquisition over-sampling (for internal use). More...
BFld	**SensorlessOnEnable** Sensorless Start Mode. More...
BFld	**HomingStyle** Homing style. More...
BFld	**StepDirAsQuadrature** *Step & Direction Command Input Format* More...
unsigned	**UsingAC** Expected Power Source. More...
unsigned	**Using1PhAC** Expected AC Power Type. More...
BFld	**__pad0__**

BFld **ValidStartMethods**
Sensorless start algorithm(s) to use. More...

Detailed Description

This header defines the node specific registers of the ClearPath-SC node. These include:

- Status Register
- Application Configuration
- Hardware Configurations
- Alert Register fields
- Output Register fields

Variable Documentation

BFld __pad0__

INTERNAL_DOC

BFld CapturePolarity

When should the position be captured by the input switch? 0 means to capture position when Input B turns off, 1 means to capture position when Input B turns on.

BFld CapturePolarityHiSpd

When should the position be captured by the high-speed input switch? 0 means to capture position when Input A turns off, 1 means to capture position when Input A turns on.

BFld DataAcqOversampling

INTERNAL_DOC Determines the rate that data acquisition points are sent 0 = every other sample 1 = every sample

BFld HomingStyle

What type of homing mode is being used? 0 is homing to a switch input, 1 is homing to a hardstop

BFld SensorlessOnEnable

Determines when the initial vector is established. 0 = At Power-On 1 = At First Enable

First enable is the recommended method if multiple nodes share a power supply. This minimizes a power supply overload condition when a system is first powered up.

Note

The power-on-delay parameter may be used to stagger the vector initialize for power-on mode.

BFld StepDirAsQuadrature

If asserted, the input format will use the quadrature format. If not asserted, the input format is step & direction.

Note

This feature was introduced with firmware version 1.6

Used by ClearView

unsigned Using1PhAC

On AC capable motors, this field sets the expected AC power source.

Set to 0 for all 3phase powered motors and 1 for single phase.

Note

This feature was introduced with firmware version 1.6.10

Used by ClearView

unsigned UsingAC

On AC capable motors, this field sets the expected power connection.

Set to 0 for all DC powered motors and 1 for AC Powered.

Note

This feature was introduced with firmware version 1.6

Used by ClearView

BFld ValidStartMethods

When SensorlessMode is not set, this value determines the sensorless start method.

pubCoreRegs.h File Reference

Universal Register Definitions. More...

Go to the source code of this file.

Classes

union **_alertReg**
Universal Alert and Warning Register Type. More...

union **_mnStatusReg**
Universal Status Register Type. More...

Typedefs

typedef union MN_EXPORT PACK_BYTES **_alertReg** **alertReg**
Universal Alert and Warning Register Type. More...

typedef union MN_EXPORT PACK_BYTES **_mnStatusReg** **mnStatusReg**
Universal Status Register Type. More...

Detailed Description

These registers are typically implemented as unions. The members of the union include: -a bit-wide interface named *bits* -one or more node specific structs to access node specific fields. For ClearPath-SC this member is named "cpm". -The Meridian-ISC member is named either "isc" or "fld" depending on its original legacy name.

Typedef Documentation

typedef union MN_EXPORT PACK_BYTES _alertReg alertReg

The alert register defines the shutdown state of the node. When a node enters a shutdown or potentially non-operable condition, the node will set bits in this register to tell the application what went wrong. Alerts range from serious to minor.

- Hardware/Software failures during power on self testing and internal consistency checks that occur during operation. These include items such as as:
  - Sensor failures
  - Component failures
  - Sensors drifting out of their nominal ranges
  - Non-volatile configuration errors
- Self preservation shutdowns where the node shuts down to protect itself from being damaged.
  - Over motor bus voltage
  - Shorted circuits
  - Excessive power delivery to motor
  - Jams
- Application errors such as:
  - Parameters out-of-range for requested operations such as commanding a node past it's soft limits

Most shutdowns can be cleared via the **sFnd::IStatus::AlertsClear** function to allow continued node operations, however fatal errors cannot be cleared by this mechanism and require a power-cycle or node restart to clear them.

Note

See the specific bit for potential recovery actions to allow node operations to continue. If the Status Register's **HwFailure** bit is asserted, this is a hardware failure and recovery is probably not possible.

See also

_cpmAlertFlds For the complete list of ClearPath-SC alert Fields.

sFnd::IStatus::Alerts for access to this register.

typedef union MN_EXPORT PACK_BYTES _mnStatusReg mnStatusReg

This Union defines the Universal Status Register for all types of Nodes.

The Status register is used to reflect the state of the Node. Status events occur when bits/fields of this register are set/cleared.

The lower 32 bits of this register can be used to generate an asynchronous attention packet to signal the host an event occurred.

See also

sFnd::ValueStatus Class for accessing Status registers within the Node.

cpmStatusRegFlds For the Complete list of ClearPath-SC alert Fields.

Attentions/Interupts for Event Driven Applications for more information regarding Attentions.

sFoundation 2.0

pubSysCls.h

Go to the documentation of this file.

```
//*********************************************************************
// $Archive: /ClearPath SC/User Driver/inc/pubSysCls.h $
// $Date: 01/19/2017 17:39 $
// $Workfile: pubSysCls.h $
//
// DESCRIPTION:
// PRINCIPLE AUTHORS:
//      DWS
//
// CREATION DATE:
//      2015-12-08 12:36:45
//
// COPYRIGHT NOTICE:
//      (C)Copyright 2015-2016 Teknic, Inc.  All rights rese
//
//      This copyright notice must be reproduced in any copy, modification,
//      or portion thereof merged into another program. A copy of the
//      copyright notice must be included in the object library of a user
//      program.
//
//*********************************************************************
#ifndef __PUBSYSCLS_H__
#define __PUBSYSCLS_H__


//*********************************************************************
// NAME
//  pubSysCls.h headers
//
```

```
38 #include "tekTypes.h"
39 #include "pubCoreRegs.h"
40 #include "pubPwrReg.h"
41 #include "pubNetAPI.h"
42
43 // Use C++ lib strings
44 #include <string>
45 #include <vector>
46
47 //
   *
48 //*********************************************************
   **********************
49
50
51
52
53 //*********************************************************
   **********************
54 // NAME
   *
55 //  pubSysCls.h forward references
57 namespace sFnd
58 {
59     class INode;
60     class IPort;
61     class IAttnPort;
65     typedef void (nodeCallback *mnAttnCallback)(const
   mnAttnReqReg &detected);
66 }
67 //                                                    /**
   \endcond **/
68 //*********************************************************
   **********************
69
70
71
72 //*********************************************************
   **********************
73 // NAME
   *
74 //  pubSysCls.h constants
75 //
76 //
77 //
   *
```

```
78  //*********************************************************
    **********************
79
80  //*********************************************************
    **********************
81
85  namespace sFnd
86  {
87  #if defined(__GNUC__)
88  #pragma GCC diagnostic push
89  #pragma GCC diagnostic ignored "-Wreorder"
90  #endif
91
92  //*********************************************************
    **********************
93  // NAME
    *
94  //  IObj Class
101 class MN_EXPORT IObj
102 {
104     friend class CPMinfo;
106 public:
115     virtual bool Supported();
116
118 private:
119     // Flag showing support
120     bool m_supported;
121
122 protected:
123     IObj();
124     IObj(bool support);
125
126 protected:
128     void Supported(bool newState);
130 };
131 //
    *
132 //*********************************************************
    **********************
133
134
135 //*********************************************************
    **********************
136 // NAME
    *
137 //  IObjWithNode Class
144 class MN_EXPORT IObjWithNode : public IObj
```

```
145 {
146 public:
162     INode &Node() { return *m_pNode; }
163 protected:
165     INode *m_pNode;
166     IObjWithNode(INode &ourNode);
168 };
169 //
    *
170 //*********************************************************
    **********************
171
172
173 //*********************************************************
    **********************
174 // NAME
    *
175 //  IObjWithPort Class
181 class MN_EXPORT IObjWithPort : public IObj
182 {
183 protected:
185     IPort *m_pPort;
188     IObjWithPort(IPort &ourPort);
190 };
191 //
    *
192 //*********************************************************
    **********************
193
194 //*********************************************************
    **********************
195 // NAME
    *
196 //  ValueBase class
210 class MN_EXPORT ValueBase : public IObjWithNode
211 {
213     friend class ValueString;
214     friend class ValueDouble;
215     friend class ValueSigned;
216     friend class ValueUnsigned;
217     friend class ValueAlert;
218     friend class ValueAppConfigReg;
219     friend class ValueHwConfigReg;
220     friend class ValuePowerReg;
221     friend class ValueOutReg;
222     friend class ValueStatus;
223     friend class CPMstring;
```

```
224     friend class CPMstatus;
225     friend class CPMstatusAdv;
226     friend class CPMouts;
227     friend class CPMmotion;
228 private:
229     double m_scaleToUser;
230     nodeparam m_paramNum;
231     bool m_valid;
232     bool m_exists;
233     bool m_isVolatile;
234     bool m_refreshOnAccess;
236 public:
251     virtual void Refresh() = 0;
252
265     bool Exists() { return m_exists; }
266
285     void AutoRefresh(bool newState) { m_refreshOnAccess =
    newState; }
286
301     bool AutoRefresh() { return m_refreshOnAccess; }
302
310     bool IsVolatile() { return m_isVolatile; }
312 protected:
314     void Valid(bool newState);
316     void ParamNum(nodeparam newParamNum);
317     nodeparam ParamNum() { return m_paramNum; }
319     void Scale(double newScale);
321     ValueBase(INode &node, nodeparam pNum);
323     void Exists(bool newState);
325     void IsVolatile(bool newState) {m_isVolatile = newState;
326 public:
327     virtual ~ValueBase() {};
329 };
330 //
    *
331 //*********************************************************
    **********************
332
333
334 //*********************************************************
    **********************
335 // NAME
    *
336 //  ValueDouble class
371 class MN_EXPORT ValueDouble : public ValueBase
372 {
374     friend class IInfo;
```

```
375     friend class IHoming;
376     friend class ILimits;
377     friend class ILimitsAdv;
378     friend class IMotion;
379     friend class IMotionAdv;
380     friend class IStatusAdv;
381     friend class CPMnode;
382     friend class IStatus;
383     friend class ISetup;
384     friend class ISetupEx;
385
386 protected:
387     // Source of parameter value
388     enum _srcs {
389         VIA_PARAM
390     };
391     double m_lastValue;
392     double m_currentValue;
393     enum _srcs m_src;
395 public:
396     /*
397         \brief Query the node and obtain a new copy of
    the parameter.
398
399         Query the node and obtain a new copy of the paramete
400
401
402         \CODE_SAMPLE_HDR
403         //Refresh a volatile parameter before reading
404         mynode.someFeature.someParameter.Refresh();
405         \endcode
406     **/
407     void Refresh();
408
423     double operator=(double newValue);
424
438     operator double();
439
455     operator int32_t();
456
472     operator uint32_t();
473
498     void Value(int32_t newValue, bool makeNonVolatile = fals
499
524     void Value(double newValue, bool makeNonVolatile=false);
525
541     double Value(bool getNonVolatile = false);
```

```

    // - - - - - - - - - - - - - - - - - - - -
    // Common overrides for signed integers
    // - - - - - - - - - - - - - - - - - - - -
    double operator|(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue) | arg);
    }
    double operator&(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue) & arg);
    }
    double operator^(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue) ^ arg);
    }
    double operator+(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue + arg));
    }
    double operator-(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue - arg));
    }
    double operator*(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue * arg));
    }
    double operator/(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue / arg));
    }
    double operator|=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue) | arg);
    }
    double operator&=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue) & arg);
    }
    double operator^=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue) ^ arg);
    }
    double operator+=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue + arg));
    }
    double operator-=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue - arg));
    }
    double operator*=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue * arg));
    }
    double operator/=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue / arg));
    }

```

```
    // Common overrides for unsigned integers

    double operator|(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue) | arg);
    }
    double operator&(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue) & arg);
    }
    double operator^(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue) ^ arg);
    }
    double operator+(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    double operator-(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    double operator*(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    double operator/(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue / arg));
    }
    double operator|=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue) | arg);
    }
    double operator&=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue) & arg);
    }
    double operator^=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue) ^ arg);
    }
    double operator+=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    double operator-=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    double operator*=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    double operator/=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue / arg));
    }

    // Common overrides for doubles
    double operator+(double arg) {
```

```
637         return(m_currentValue + arg);
638     }
639     double operator-(double arg) {
640         return(m_currentValue - arg);
641     }
642     double operator*(double arg) {
643         return(m_currentValue * arg);
644     }
645     double operator/(double arg) {
646         return(m_currentValue / arg);
647     }
648     double operator+=(double arg) {
649         return(m_currentValue + arg);
650     }
651     double operator-=(double arg) {
652         return(m_currentValue - arg);
653     }
654     double operator*=(double arg) {
655         return(m_currentValue * arg);
656     }
657     double operator/=(double arg) {
658         return(m_currentValue / arg);
659     }
661 protected:
663     // Construction to wire to our node object
664     ValueDouble(INode &node, nodeparam pNum, enum _srcs
    src = VIA_PARAM);
666 };
667 //
    *
668 //*********************************************************
    **********************
669
670
671 //*********************************************************
    **********************
672 // NAME
    *
673 //  ValueSigned class
708 class MN_EXPORT ValueSigned : public ValueBase
709 {
711     friend class IInfo;
712     friend class IHoming;
713     friend class ILimits;
714     friend class IMotion;
715     friend class IMotionAdv;
716     friend class IStatusAdv;
```

```
717
718 private:
719     int32_t m_lastValue;
720     int32_t m_currentValue;
722 public:
734     void Refresh();
735
750     int32_t operator=(int32_t newValue);
751
765     operator double();
783     operator int32_t();
800     operator uint32_t();
822     void Value(int32_t newValue, bool makeNonVolatile = fals
839     int32_t Value(bool getNonVolatile=false);
841     // - - - - - - - - - - - - - - - - - - - -
842     // Common overrides for signed integers
843     // - - - - - - - - - - - - - - - - - - - -
844     int32_t operator|(int32_t arg) {
845         return(static_cast<int32_t>(m_currentValue | arg));
846     }
847     int32_t operator&(int32_t arg) {
848         return(static_cast<int32_t>(m_currentValue & arg));
849     }
850     int32_t operator^(int32_t arg) {
851         return(static_cast<int32_t>(m_currentValue ^ arg));
852     }
853     int32_t operator+(int32_t arg) {
854         return(static_cast<int32_t>(m_currentValue + arg));
855     }
856     int32_t operator-(int32_t arg) {
857         return(static_cast<int32_t>(m_currentValue - arg));
858     }
859     int32_t operator*(int32_t arg) {
860         return(static_cast<int32_t>(m_currentValue * arg));
861     }
862     int32_t operator/(int32_t arg) {
863         return(static_cast<int32_t>(m_currentValue / arg));
864     }
865     int32_t operator|=(int32_t arg) {
866         return(static_cast<int32_t>(m_currentValue | arg));
867     }
868     int32_t operator&=(int32_t arg) {
869         return(static_cast<int32_t>(m_currentValue & arg));
870     }
871     int32_t operator^=(int32_t arg) {
872         return(static_cast<int32_t>(m_currentValue ^ arg));
873     }
```

```
    int32_t operator+=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue + arg));
    }
    int32_t operator-=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue - arg));
    }
    int32_t operator*=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue * arg));
    }
    int32_t operator/=(int32_t arg) {
        return(static_cast<int32_t>(m_currentValue / arg));
    }

    // Common overrides for unsigned integers

    int32_t operator|(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue | arg));
    }
    int32_t operator&(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue & arg));
    }
    int32_t operator^(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue ^ arg));
    }
    int32_t operator+(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    int32_t operator-(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    int32_t operator*(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    int32_t operator/(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue / arg));
    }
    int32_t operator|=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue | arg));
    }
    int32_t operator&=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue & arg));
    }
    int32_t operator^=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue ^ arg));
    }
    int32_t operator+=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
```

```
    }
    int32_t operator-=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    int32_t operator*=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    int32_t operator/=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue / arg));
    }

    // Common overrides for doubles
    int32_t operator+(double arg) {
        return(static_cast<int32_t>(m_currentValue + arg));
    }
    int32_t operator-(double arg) {
        return(static_cast<int32_t>(m_currentValue - arg));
    }
    int32_t operator*(double arg) {
        return(static_cast<int32_t>(m_currentValue * arg));
    }
    int32_t operator/(double arg) {
        return(static_cast<int32_t>(m_currentValue / arg));
    }
    int32_t operator+=(double arg) {
        return(static_cast<int32_t>(m_currentValue + arg));
    }
    int32_t operator-=(double arg) {
        return(static_cast<int32_t>(m_currentValue - arg));
    }
    int32_t operator*=(double arg) {
        return(static_cast<int32_t>(m_currentValue * arg));
    }
    int32_t operator/=(double arg) {
        return(static_cast<int32_t>(m_currentValue / arg));
    }
protected:
    // Construction to wire to our node object
    ValueSigned(INode &node, nodeparam pNum);
};
//
*
//*********************************************************
**********************


```

```
969 //*****************************************************
    **********************
970 // NAME
    *
971 //  ValueUnsigned class
1007 class MN_EXPORT ValueUnsigned : public ValueBase
1008 {
1010     friend class IInfo;
1011     friend class IHoming;
1012     friend class ILimits;
1013     friend class IMotion;
1014     friend class IMotionAdv;
1015 private:
1016     uint32_t m_lastValue;
1017     uint32_t m_currentValue;
1019 public:
1031     void Refresh();
1032
1045         uint32_t operator=(uint32_t newValue);
1046
1061     operator double();
1062
1079     operator uint32_t();
1080
1105     void Value(uint32_t newValue, bool makeNonVolatile = fal
1106
1125     uint32_t Value(bool getNonVolatile = false);
1127     // - - - - - - - - - - - - - - - - - - - -
1128     // Common overrides for signed integers
1129     // - - - - - - - - - - - - - - - - - - - -
1130     uint32_t operator|(int32_t arg) {
1131         return(m_currentValue | arg);
1132     }
1133     uint32_t operator&(int32_t arg) {
1134         return(m_currentValue & arg);
1135     }
1136     uint32_t operator^(int32_t arg) {
1137         return(m_currentValue ^ arg);
1138     }
1139     uint32_t operator+(int32_t arg) {
1140         return(m_currentValue + arg);
1141     }
1142     uint32_t operator-(int32_t arg) {
1143         return(m_currentValue - arg);
1144     }
1145     uint32_t operator*(int32_t arg) {
1146         return(m_currentValue * arg);
```

```
    }
    uint32_t operator/(int32_t arg) {
        return(m_currentValue / arg);
    }
    uint32_t operator|=(int32_t arg) {
        return(m_currentValue | arg);
    }
    uint32_t operator&=(int32_t arg) {
        return(m_currentValue & arg);
    }
    uint32_t operator^=(int32_t arg) {
        return(m_currentValue ^ arg);
    }
    uint32_t operator+=(int32_t arg) {
        return(m_currentValue + arg);
    }
    uint32_t operator-=(int32_t arg) {
        return(m_currentValue - arg);
    }
    uint32_t operator*=(int32_t arg) {
        return(m_currentValue * arg);
    }
    uint32_t operator/=(int32_t arg) {
        return(m_currentValue / arg);
    }

    // Common overrides for unsigned integers

    uint32_t operator|(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue | arg));
    }
    uint32_t operator&(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue & arg));
    }
    uint32_t operator^(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue ^ arg));
    }
    uint32_t operator+(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    uint32_t operator-(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    uint32_t operator*(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    uint32_t operator/(uint32_t arg) {
```

```
        return(static_cast<uint32_t>(m_currentValue / arg));
    }
    uint32_t operator|=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue | arg));
    }
    uint32_t operator&=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue & arg));
    }
    uint32_t operator^=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue ^ arg));
    }
    uint32_t operator+=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    uint32_t operator-=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    uint32_t operator*=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    uint32_t operator/=(uint32_t arg) {
        return(static_cast<uint32_t>(m_currentValue / arg));
    }

    // Common overrides for doubles
    uint32_t operator+(double arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    uint32_t operator-(double arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    uint32_t operator*(double arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    uint32_t operator/(double arg) {
        return(static_cast<uint32_t>(m_currentValue / arg));
    }
    uint32_t operator+=(double arg) {
        return(static_cast<uint32_t>(m_currentValue + arg));
    }
    uint32_t operator-=(double arg) {
        return(static_cast<uint32_t>(m_currentValue - arg));
    }
    uint32_t operator*=(double arg) {
        return(static_cast<uint32_t>(m_currentValue * arg));
    }
    uint32_t operator/=(double arg) {
```

```
1241            return(static_cast<uint32_t>(m_currentValue / arg));
1242        }
1245    protected:
1247        // Construction to wire to our node object
1248        ValueUnsigned(INode &node, nodeparam pNum);
1250    };
1251    //
    *
1252    //*********************************************************
    **********************
1253
1254
1255    //*********************************************************
    **********************
1256    // NAME
    *
1257    //  ValueString class
1293    class MN_EXPORT ValueString : public ValueBase
1294    {
1296        friend class IInfo;
1297        friend class CPMinfo;
1298    protected:
1299        enum _srcs {
1303            STR_USERID,
1307            STR_PARAM,
1311            STR_CONFIG_NAME,
1315            STR_FW_VERSION,
1319            STR_HW_VERSION,
1323            STR_MODEL_STR
1324        };
1325
1326        // C++ hack to create a constant
1327        enum _MAX_STR { MAX_STR = MN_UI_STR_BUF_SIZE };
1328        // Where to find string
1329        enum _srcs  m_location;
1330
1331        char m_lastValue[MAX_STR];
1332        char m_currentValue[MAX_STR];
1334    public:
1346        virtual void Refresh() = 0;
1347
1359        virtual void Value(const char *newValue) = 0;
1360
1373        virtual const char *Value() = 0;
1374
1386        const char *operator=(const char *newValue);
1387
```

```
1397     operator const char *();
1398
1399 protected:
1401     // Construction to wire to our node object
1402     ValueString(INode &node, nodeparam pNum, enum _srcs loca
1404 };
1405 //
     *
1406 //*********************************************************
     **********************
1407
1408
1409 //*********************************************************
     **********************
1410 // NAME
     *
1411 //  ValueStatus class
1444 class MN_EXPORT ValueStatus : public ValueBase
1445 {
1447     friend class IStatus;
1448     friend class IAttnNode;
1449 private:
1450     mnStatusReg m_lastValue;
1451     mnStatusReg m_currentValue;
1452     mnStatusReg m_testResult;
1453     bool m_clearOnRead;
1455 public:
1475     void Refresh();
1476
1511     bool TestAndClear(const mnStatusReg &mask,
     mnStatusReg &result);
1512
1543     bool TestAndClear(const mnStatusReg &mask);
1544
1558     void Clear();
1559
1583     mnStatusReg &operator=(mnStatusReg &newValue);
1584
1592     operator mnStatusReg();
1593
1599     mnStatusReg Value();
1600
1624     void Value(const mnStatusReg &newValue);
1625
1633     mnStatusReg Last();
1634
1635 protected:
```

```
1637     // Construction to wire to our node object
1638     ValueStatus(INode &node, nodeparam pNum, bool clearOnRea
1640 };
1641 //
     *
1642 //*********************************************************
     **********************
1643
1644
1645 //*********************************************************
     **********************
1646 // NAME
     *
1647 //  ValueAlert class
1713 class MN_EXPORT ValueAlert : public ValueBase
1714 {
1716     friend class IStatus;
1717     friend class IAttnNode;
1718 private:
1719     alertReg m_lastValue;
1720     alertReg m_currentValue;
1721     alertReg m_testResult;
1722     bool m_clearOnRead;
1724 public:
1731     void Refresh();
1746     bool TestAndClear(const alertReg &mask, alertReg &result
1747
1754     void Clear();
1778     alertReg &operator=(alertReg &newValue);
1779
1785     operator alertReg();
1786
1792     alertReg Value();
1793
1818     void Value(const alertReg &newValue);
1819
1825     alertReg Last() { return m_lastValue; }
1826 protected:
1828     // Construction to wire to our node object
1829     ValueAlert(INode &node, nodeparam pNum, bool clearOnRead
1831 };
1832 //
     *
1833 //*********************************************************
     **********************
1834
1835
```

```
1836 //*********************************************************
     **********************
1837 // NAME
     *
1838 //  ValuePowerReg class
1888 class MN_EXPORT ValuePowerReg : public ValueBase
1889 {
1891     friend class IStatus;
1892     friend class IAttnNode;
1893 private:
1894     mnPowerReg m_lastValue;
1895     mnPowerReg m_currentValue;
1896     mnPowerReg m_testResult;
1897     bool m_clearOnRead;
1899 public:
1906     void Refresh();
1921     bool TestAndClear(const mnPowerReg &mask, mnPowerReg &re
1922
1929     void Clear();
1937     operator mnPowerReg();
1938
1944     mnPowerReg Value();
1945
1951     mnPowerReg Last() { return m_lastValue; }
1952 protected:
1954     // Construction to wire to our node object
1955     ValuePowerReg(INode &node, nodeparam pNum, bool clearOnR
1957 };
1958 //
     *
1959 //*********************************************************
     **********************
1960
1961
1962 //*********************************************************
     **********************
1963 // NAME
     *
1964 // ValueOutReg class
1971 class MN_EXPORT ValueOutReg : public ValueBase
1972 {
1973
1974     friend class IOuts;
1975
1976 private:
1977     mnOutReg m_lastValue;
1978     mnOutReg m_currentValue;
```

```
1979     mnOutReg m_testResult;
1980     bool m_clearOnRead;
1981     bool m_writeable;
1982
1983 public:
1991     void Refresh();
1998     void Clear(const mnOutReg &mask);
2014     void Set(const mnOutReg &mask);
2015
2025     mnOutReg &operator=(mnOutReg &newValue);
2033     operator mnOutReg();
2041     operator double();
2049     operator int32_t();
2057     operator uint32_t();
2058
2068     void Value(const mnOutReg &newValue);
2069
2075     mnOutReg Value();
2076
2082     mnOutReg Last() { return m_lastValue; }
2083
2084 protected:
2085
2086     // Construction to wire to our node object
2087     ValueOutReg(INode &node, nodeparam pNum,
2088         bool clearOnReadType, bool writeable);
2089
2090 };
2092 //
     *
2093 //*****************************************************
     *********************
2094
2095
2096 //*****************************************************
     *********************
2097 // NAME
     *
2098 //  ValueAppConfigReg class
2113 class MN_EXPORT ValueAppConfigReg : public ValueBase
2114 {
2116     friend class ISetupEx;
2117     friend class CPMmotionAdv;
2118
2119 private:
2120     mnAppConfigReg m_lastValue;
2121     mnAppConfigReg m_currentValue;
```

```
2122     mnAppConfigReg m_testResult;
2124 public:
2132     void Refresh();
2140     void Clear(const mnAppConfigReg &mask);
2148     void Set(const mnAppConfigReg &mask);
2149
2157     mnAppConfigReg &operator=(mnAppConfigReg &newValue);
2158
2164     operator mnAppConfigReg();
2165
2171     operator double();
2172
2178     operator int32_t();
2179
2185     operator uint32_t();
2186
2192     void Value(const mnAppConfigReg &newValue);
2198     mnAppConfigReg Value();
2199
2205     mnAppConfigReg Last() { return m_lastValue; }
2206
2207
2208 protected:
2210     // Update the internal value to new state
2211     void setNewVal(const mnAppConfigReg &newValue);
2212     // Construction to wire to our node object
2213     ValueAppConfigReg(INode &node, nodeparam pNum);
2215 };
2217 //
     *
2218 //*****************************************************
     *********************
2219
2220
2221 //*****************************************************
     *********************
2222 // NAME
     *
2223 //  ValueHwConfigReg class
2238 class MN_EXPORT ValueHwConfigReg : public ValueBase
2239 {
2241     friend class ISetupEx;
2242
2243 private:
2244     mnHwConfigReg m_lastValue;
2245     mnHwConfigReg m_currentValue;
2246     mnHwConfigReg m_testResult;
```

```
2248 public:
2256     void Refresh();
2257
2265     void Clear(const mnHwConfigReg &mask);
2266
2274     void Set(const mnHwConfigReg &mask);
2275
2276
2286     mnHwConfigReg &operator=(mnHwConfigReg &newValue);
2287
2293     operator mnHwConfigReg();
2294
2301     operator double();
2302
2303
2309     operator int32_t();
2310
2316     operator uint32_t();
2317
2323     mnHwConfigReg Value();
2324
2332     void Value(const mnHwConfigReg &newValue);
2333
2339     mnHwConfigReg Last() { return m_lastValue; }
2340
2341 protected:
2343     ValueHwConfigReg(INode &node, nodeparam pNume);
2346 };
2348 //
     *
2349 //*****************************************************
     *********************
2350
2351
2352
2353 //*****************************************************
     *********************
2354 // NAME
     *
2355 //  IBrakeControl class
2398 class MN_EXPORT IBrakeControl : public IObjWithPort
2399 {
2401     friend class IPort;
2403 public:
2416     virtual void BrakeSetting(size_t brakeNum,
     BrakeControls brakeMode) = 0;
```

```
2427     virtual BrakeControls BrakeSetting(size_t brakeNum) = 0;
2428
2439     virtual bool BrakeEngaged(size_t brakeNum) = 0;
2440
2441 protected:
2443     /* Construction */
2444     IBrakeControl(IPort &port);
2445 public:
2446     virtual ~IBrakeControl() {}
2448 };
2449 //*
2450 //*********************************************************************
2451
2452
2453 //*********************************************************************
2454 // NAME*
2455 //  IGrpShutdown class
2496 class MN_EXPORT IGrpShutdown : public IObjWithPort
2497 {
2499     friend class IPort;
2501 public:
2516     virtual void ShutdownWhen(size_t nodeIndex, const ShutdownInfo &theInfo) = 0;
2528     virtual void ShutdownWhenGet(size_t nodeIndex, ShutdownInfo &theInfo) = 0;
2535     virtual void ShutdownInitiate() = 0;
2536 protected:
2538     // Construct our group shutdown implementation
2539     IGrpShutdown(IPort &port);
2540 public:
2541     virtual ~IGrpShutdown() {}
2543 };
2544 //*
2545 //*********************************************************************
2546
2547
2548
2549 //*********************************************************************
```

```
2550 // NAME
     *
2551 //  IPortAdv class
2558 class MN_EXPORT IPortAdv : public IObjWithPort
2559 {
2561     friend class IPort;
2563 public:
2564
2572     IAttnPort &Attn;
2573
2606     virtual void TriggerMovesInGroup(size_t groupNumber)
     = 0;
2607
2608
2609     bool Supported();
2611     // Construction
2612 protected:
2613     IPortAdv(IPort &ourPort, IAttnPort &attnPort);
2614 public:
2615     virtual ~IPortAdv() {}
2617 };
2618 //
     *
2619 //*********************************************************
     **********************
2620
2621 //*********************************************************
     **********************
2622 // NAME
     *
2623 //  IPort class
2659 class MN_EXPORT IPort
2660 {
2662     friend class CPMportAdv;
2663 public:
2664 protected:
2666     netaddr m_netNumber;
2667 public:
2670 public:
2678     INode &Nodes(size_t index);
2679
2696     virtual Uint16 NodeCount() = 0;
2697
2704     virtual void RestartCold() = 0;
2705
2716     virtual openStates OpenState() = 0;
2717
```

```
2727     virtual bool WaitForOnline(int32_t timeoutMsec =
    15000) = 0;
2728
2739     virtual void CommandTraceSave(const char *filePath) =
    0;
2740
2762     virtual void NodeStop(mgNodeStopReg stopType =
    STOP_TYPE_IGNORE) = 0;
2763
2768     IGrpShutdown &GrpShutdown;
2769
2773     IBrakeControl &BrakeControl;
2774
2778     IPortAdv &Adv;
2779
2781     netaddr NetNumber() { return m_netNumber; }
2782     void NetNumber(netaddr index) { m_netNumber = index; }
2785 public:
2787     // Construction
2788     IPort(netaddr index, IBrakeControl &brake,
    IGrpShutdown &shtDwn,
2789         IPortAdv &advPort);
2790     virtual ~IPort();
2792 };
2793 //
    *
2794 //*********************************************************
    **********************
2795
2796
2797 //*********************************************************
    **********************
2798 // NAME
    *
2799 //  IMotionAudit class
2810 class MN_EXPORT IMotionAudit : public IObjWithNode
2811 {
2812 private:
2813 public:
2817     enum _testPoints {
2821         MON_POS_TRK = 6,
2825         MON_TRQ_MEAS = 7,
2829         MON_TRQ_CMD = 8
2830     };
2838     mnAuditData Results;
2839
2848     virtual void Refresh() = 0;
```

```
2849
2866     virtual void SelectTestPoint(enum _testPoints testPoint,
2867         double fullScale, double filterTCmsec) = 0;
2868
2869     // Construction
2871 protected:
2872     IMotionAudit(INode &ourNode);
2873 public:
2874     virtual ~IMotionAudit() {}
2876 };
2878 //
     *
2879 //*********************************************************
     **********************
2880
2881
2882 //*********************************************************
     **********************
2883 // NAME
     *
2884 //  IAttnNode class
2904 class MN_EXPORT IAttnNode : public IObjWithNode
2905 {
2907     friend class INodeAdv;
2908     friend class INode;
2910 protected:
2911 public:
2924     ValueStatus Mask;
2938     ValueAlert WarnMask;
2951     ValueAlert AlertMask;
2952
2965     ValueStatus StatusMask;
2966
3012     mnStatusReg WaitForAttn(mnStatusReg theAttn, int32_t tim
3013                             bool autoClear = true);
3023     void ClearAttn(mnStatusReg attnClr);
3024
3032     void SignalAttn(mnStatusReg theAttn);
3033     // Construction
3034 protected:
3035     // Attention handling implementation.
3039     virtual void AttnRegister() = 0;
3044     virtual void AttnUnregister() = 0;
3054     virtual bool AttnWait(size_t timeoutMsec) = 0;
3058     virtual void AttnSignal() = 0;
3059
3063     mnStatusReg m_attn;
```

```
3064
3065     IAttnNode(INode &ourNode,
3066         nodeparam attnMaskPnum, nodeparam usrWarnMaskPnum,
3067         nodeparam usrAlertMaskPnum, nodeparam usrStatMaskPnu
3069 public:
3070     virtual ~IAttnNode() {};
3071 };
3072 //
    *
3073 //*****************************************************
    **********************
3074
3075
3076 //*****************************************************
    **********************
3077 // NAME
    *
3078 //  IAttnPort class
3079
3089 class MN_EXPORT IAttnPort : public IObjWithPort
3090 {
3092     friend class SysCPMattnPort;
3093 private:
3094     bool AttnsEnabled;
3095     mnAttnCallback AttnCallback;
3097 public:
3106     typedef enum _attnState {
3110         HAS_ATTN,
3114         TIMEOUT,
3118         DISABLED,
3123         TERMINATING
3124     } attnState;
3137     virtual void Enable(bool newState) = 0;
3138
3145     virtual bool Enabled() = 0;
3158     virtual attnState WaitForAttn(mnAttnReqReg
    &attnRecvd) = 0;
3168     bool HasAttnHandler();
3169
3206     void AttnHandler(mnAttnCallback theNewHandler);
3215     void InvokeAttnHandler(const mnAttnReqReg &detected);
3216
3217     // Construction
3218 protected:
3219     IAttnPort(IPort &ourPort);
3220     virtual ~IAttnPort();
3222 };
```

```
3223
3224 //
     *
3225 //*********************************************************
     **********************
3226
3227
3228 //*********************************************************
     **********************
3229 // NAME
     *
3230 //  ILimitsAdv Class
3237 class MN_EXPORT ILimitsAdv : public IObjWithNode
3238 {
3240     friend class CPMnode;
3242 public:
3254     ValueDouble PositiveTrq;
3255
3266     ValueDouble PositiveRelaxTCmsec;
3267
3279     ValueDouble NegativeTrq;
3280
3291     ValueDouble NegativeRelaxTCmsec;
3292
3303     virtual void StartPosFoldback(bool engage) = 0;
3304
3314     virtual bool StartPosFoldback() = 0;
3315
3325     virtual void StartNegFoldback(bool engage) = 0;
3326
3336     virtual bool StartNegFoldback() = 0;
3337
3338 protected:
3340     // Construction
3341     ILimitsAdv(INode &ourNode,
3342         nodeparam posFldkPnum, nodeparam posFldkTCpNum,
3343         nodeparam negFldkPnum, nodeparam negFldkTCpNum);
3344 public:
3345     virtual ~ILimitsAdv() {}
3347 };
3349 //
     *
3350 //*********************************************************
     **********************
3351
3352
3353
```

```
3354 //*********************************************************
     **********************
3355 // NAME
     *
3356 //  INodeAdv class
3366 class MN_EXPORT INodeAdv : public IObjWithNode
3367 {
3368 public:
3375     IAttnNode &Attn;
3383     IMotionAudit &MotionAudit;
3386     // Construction
3387 protected:
3388     INodeAdv(INode &ourNode, IMotionAudit &audit,
     IAttnNode &attnNode);
3389     virtual ~INodeAdv() {};
3391 };
3392 //
     *
3393 //*********************************************************
     **********************
3394
3395
3396 //*********************************************************
     **********************
3397 // NAME
     *
3398 //  INodeEx class
3406 class MN_EXPORT INodeEx : public IObjWithNode
3407 {
3409     friend class INode;
3412 protected:
3413     // Read-modify-write parameter mutex access via INode
     implementations
3414     // using the INode::UseMutex class.
3423     virtual void MutexTake() = 0;
3434     virtual void MutexRelease() = 0;
3435     // Construction
3437 protected:
3439     INodeEx(INode &ourNode);
3440 public:
3441     virtual ~INodeEx() {}
3443 };
3445 //
     *
3446 //*********************************************************
     **********************
3447
```

```
3448
3449
3450 //*********************************************************
     **********************
3451 // NAME
     *
3452 //  IHoming class
3465 class MN_EXPORT IHoming : public IObjWithNode
3466 {
3468     friend class INode;
3471 public:
3483     virtual void Initiate() = 0;
3484
3504     virtual bool IsHoming() = 0;
3505
3525     virtual bool WasHomed() = 0;
3526
3555     virtual void SignalComplete() = 0;
3556
3571     virtual void SignalInvalid() = 0;
3572
3578     virtual bool HomingValid() = 0;
3579
3580 protected:
3581     // Construction
3583     IHoming(INode &ourNode, nodeparam accPnum, nodeparam vel
3584         nodeparam clampPnum, nodeparam offsPnum,
     nodeparam postMvDlyPnum);
3585 public:
3586     virtual ~IHoming() {}
3588 };
3589 //
     *
3590 //*********************************************************
     **********************
3591
3592
3593 //*********************************************************
     **********************
3594 // NAME
     *
3595 //  IInfoAdv class
3596
3605 class MN_EXPORT IInfoAdv : public IObjWithNode
3606 {
3607 private:
3608 public:
```

```
3609 protected:
3611     // Constructor
3612     IInfoAdv(INode &ourNode);
3614 };
3616 //
     *
3617 //*********************************************************
     **********************
3618
3619
3620 //*********************************************************
     **********************
3621 // NAME
     *
3622 //  IInfoEx class
3623
3629 class MN_EXPORT IInfoEx : public IObjWithNode
3630 {
3632 private:
3633     multiaddr m_addr;
3634 protected:
3635     void Addr(multiaddr assignAddr);
3637 public:
3645     multiaddr Addr();
3646
3655     nodeaddr NodeIndex();
3656
3668     virtual double Parameter(nodeparam index) = 0;
3678     virtual void Parameter(nodeparam index, double
     newValue) = 0;
3680 protected:
3681     IInfoEx(INode &ourNode);
3682 public:
3683     virtual ~IInfoEx() {}
3685 };
3686
3687 //
     *
3688 //*********************************************************
     **********************
3689
3690
3691 //*********************************************************
     **********************
3692 // NAME
     *
3693 //  IInfo class
```

```
3706 class MN_EXPORT IInfo : public IObjWithNode
3707 {
3708 private:
3709 public:
3711     typedef enum _nodeTypes {
3713         UNKNOWN,
3715         MERIDIAN_ISC,
3718         CLEARPATH_SC,
3721         CLEARPATH_SC_ADV
3722     } nodeTypes;
3723
3733     ValueUnsigned SerialNumber;
3734
3745     ValueString &FirmwareVersion;
3758     ValueUnsigned FirmwareVersionCode;
3759
3770     ValueString &HardwareVersion;
3782     ValueString &Model;
3783
3795     ValueUnsigned PositioningResolution;
3796
3814     virtual nodeTypes NodeType() = 0;
3815
3834     ValueString &UserID;
3835
3861     virtual void UserData(size_t bank,
3862                           const uint8_t
    m_lastUserData[MN_USER_NV_SIZE]) = 0;
3863
3879     virtual std::vector<uint8_t> UserData(size_t bank) = 0;
3880
3881
3899     ValueUnsigned UserRAM;
3900
3911     IInfoEx &Ex;
3920     IInfoAdv &Adv;
3922 protected:
3924     // Constructor
3925     IInfo(INode &ourNode, IInfoAdv &adv, IInfoEx &ex,
3926         nodeparam snPnum, nodeparam fwPnum, nodeparam usrRam
3927         ValueString &userID, ValueString &hwVer,
    ValueString &fwStr,
3928         ValueString &model, nodeparam resPnum);
3929 public:
3930     virtual ~IInfo() {}
3932 };
```

```
3933 //
     *
3934 //*********************************************************
     **********************
3935
3936
3937 //*********************************************************
     **********************
3938 // NAME
     *
3939 //  ILimits Class
3940
3947 class MN_EXPORT ILimits : public IObjWithNode
3948 {
3949 public:
3950     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
3951     // Torque Group
3952     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
3969     ValueDouble TrqGlobal;
3970
3971     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
3972     // Position Group
3973     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
3982     ValueSigned SoftLimit1;
3983
3991     ValueSigned SoftLimit2;
4001     ValueUnsigned PosnTrackingLimit;
4002
4003     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
4004     // Velocity Group
4005     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
4013     ValueDouble MotorSpeedLimit;
4019     ILimitsAdv &Adv;
4022 protected:
4024     // Construction
4025     ILimits(INode &ourNode, ILimitsAdv &adv,
4026         nodeparam globTrqPnum, nodeparam trkLimPnum,
4027         nodeparam softLim1Pnum, nodeparam softLim2Pnum,
4028         nodeparam mtrSpdLimPnum);
4030 };
```

```
4031 //
     *
4032 //*****************************************************
     *********************
4033
4034
4035 //*****************************************************
     *********************
4036 // NAME
     *
4037 //  IMotionAdv Class
4053 class MN_EXPORT IMotionAdv : public IObjWithNode
4054 {
4055 public:
4056     // - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
4057     // Kinematic Constraint Group
4058     // - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
4074     ValueDouble DecelLimit;
4075
4093     ValueDouble HeadTailVelLimit;
4094
4110     ValueUnsigned HeadDistance;
4111
4126     ValueUnsigned TailDistance;
4127
4143     ValueUnsigned AfterStartDistance;
4159     ValueUnsigned BeforeEndDistance;
4160
4161     // - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
4162     // Motion Duration Calculation Group
4163     // - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
4196     double MovePosnDurationMsec(int32_t targetPosn,
4197         bool targetIsAbsolute = false);
4198
4236     virtual double MovePosnHeadTailDurationMsec(int32_t targ
4237         bool targetIsAbsolute = false,
4238         bool hasHead = true,
4239         bool hasTail = true);
4240
4274     virtual double MovePosnAsymDurationMsec(int32_t targetPo
4275         bool targetIsAbsolute = false);
4276
4277
```

```
4304     double MoveVelDurationMsec(double targetCountsPerSec);
4305
4306
4307
4308     // - - -
```

```
4528            nodeparam tailDxPnum, nodeparam aDxPnum,
    nodeparam bDxPnum);
4529        // Destruction
4530    public:
4531        virtual ~IMotionAdv() {};
4533    };
4534    //
    *
4535    //*****************************************************
    *********************
4536
4537
4538
4539    //*****************************************************
    *********************
4540    // NAME
    *
4541    //  IMotion Class
4547    class MN_EXPORT IMotion : public IObjWithNode
4548    {
4549    public:
4550        // - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - -
4551        // Kinematic Constraint Group
4552        // - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - -
4568        ValueDouble AccLimit;
4569
4585        ValueDouble VelLimit;
4586
4591        ValueUnsigned JrkLimit;
4592
4596        ValueDouble JrkLimitDelay;
4612        ValueUnsigned DwellMs;
4613
4614        // - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - -
4615        // Motion Duration Calculation Group
4616        // - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - -
4648        virtual double MovePosnDurationMsec(int32_t target,
4649            bool targetIsAbsolute = false);
4650
4676        virtual double MoveVelDurationMsec(double target);
4677
4678
```

```
4679     // - - -
```

```
5042
5043     // - - -
```

```
5154
5158     virtual void EnableReq(bool newState) = 0;
5162     virtual bool EnableReq() = 0;
5163     // Access to whole output register
5173     ValueOutReg User;
5181     ValueOutReg Out;
5182 protected:
5183
5184     // Construction
5185     IOuts(INode &ourNode, nodeparam usrOutPnum, nodeparam
     outPnum);
5186     virtual ~IOuts() {};
5187
5188 };
5190 //
     *
5191 //*********************************************************
     **********************
5192
5193
5194 //*********************************************************
     **********************
5195 // NAME
     *
5196 //  ISetupEx class
5203 class MN_EXPORT ISetupEx : public IObjWithNode
5204 {
5206     friend class ISetup;
5208 public:
5224     ValueDouble NetWatchdogMsec;
5232     ValueAppConfigReg App;
5233
5240     ValueHwConfigReg HW;
5242 protected:
5244     // Construction
5245     ISetupEx(INode &ourNode,
5246         nodeparam appPnum, nodeparam hwPnum, nodeparam netWd
5248 };
5249 //
     *
5250 //*********************************************************
     **********************
5251
5252
5253 //*********************************************************
     **********************
```

```
5254 // NAME
     *
5255 //  ISetup class
5263 class MN_EXPORT ISetup : public IObjWithNode
5264 {
5265 public:
5280     ValueDouble DelayToDisableMsecs;
5281
5282     // Configuration Load/Save
5283
5301     virtual void ConfigLoad(const char *filePath, bool
     doReset = false) = 0;
5302
5320     virtual void ConfigSave(const char *filePath) = 0;
5321
5337     virtual bool AccessLevelIsFull() = 0;
5341     ISetupEx Ex;
5342
5343 protected:
5345     // Construction
5346     ISetup(INode &ourNode,
5347         nodeparam appPnum, nodeparam hwPnum, nodeparam wdogP
5348         nodeparam delayPnum);
5349 public:
5350     virtual ~ISetup() {}
5352 };
5353 //
     *
5354 //*********************************************************
     **********************
5355
5356
5357 //*********************************************************
     **********************
5358 // NAME
     *
5359 //  IStatusAdv class
5369 class MN_EXPORT IStatusAdv : public IObjWithNode
5370 {
5371 public:
5379     ValueSigned CapturedHiResPosn;
5387     ValueSigned CapturedPos;
5388 protected:
5390     // Construction
5391     IStatusAdv(INode &ourNode, nodeparam hiResCapPnum,
     nodeparam capPnum);
5393 };
```

```
5394 //
     *
5395 //*********************************************************
     **********************
5396
5397
5398 //*********************************************************
     **********************
5399 // NAME
     *
5400 //  IStatus class
5470 class MN_EXPORT IStatus : public IObjWithNode
5471 {
5472 public:
5473     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
5474     // Status Register Instances
5475     // - - - - - - - - - - - - - - - - - - - - - - - - - -
     - - - - -
5481     ValueStatus RT;
5482
5488     ValueStatus Rise;
5489
5495     ValueStatus Fall;
5496
5502     ValueStatus Accum;
5509     ValueAlert Warnings;
5517     ValueAlert Alerts;
5518
5524     ValueDouble RMSlevel;
5525
5531     ValuePowerReg Power;
5532
5538     IStatusAdv &Adv;
5539 public:
5546     virtual void AlertsClear() = 0;
5547
5555     virtual bool IsReady() = 0;
5556
5566     virtual bool HadTorqueSaturation() = 0;
5567
5568
5569 protected:
5571     // Construction
5572     IStatus(INode &ourNode, IStatusAdv &adv,
5573         nodeparam rtPnum, nodeparam risePnum, nodeparam
     fallPnum, nodeparam accumPnum,
```

```
5574            nodeparam warnPnum, nodeparam alertPnum,
    nodeparam rmsLvlPnum,
5575            nodeparam powerPnum);
5576    public:
5577        virtual ~IStatus() {}
5579    };
5580    //
    *
5581    //*********************************************************
    **********************
5582
5583
5584    //*********************************************************
    **********************
5585    // NAME
    *
5586    //  INode class
5629    class MN_EXPORT INode
5630    {
5632        friend class IInfoEx;
5634    public:
5665        class MN_EXPORT UseMutex {
5666        private:
5668            INode &m_node;
5669        public:
5676            UseMutex(INode &ourNode);
5678            ~UseMutex();
5680        };
5685        class MN_EXPORT MyEvent {
5686        private:
5687            // Hold a reference to our node class's internal mut
5688            INode &m_node;
5689            int32_t m_eventCount;
5690        public:
5691            MyEvent(INode &ourNode);
5692            bool Wait(size_t timeoutMs);
5693            void Signal();
5694            void Register();
5695            void Unregister();
5696            int32_t Count();
5697            void Reset(int32_t count);
5698        };
5700    protected:
5702        // Address of this node
5703        multiaddr m_addr;
5707        // - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - -
```

```
5708     // Acceleration unit handling
5709     // - - -
```

```
5830     // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
5831 public:
5835     IPort &Port;
5840     IInfo &Info;
5844     IMotion &Motion;
5848     IStatus &Status;
5852     ILimits &Limits;
5857     IOuts &Outs;
5862     ISetup &Setup;
5867     INodeEx &Ex;
5872     INodeAdv &Adv;
5873
5874     // Easy access to enable request
5886     void EnableReq(bool newState) {
5887         Outs.EnableReq(newState);
5888     }
5898     bool EnableReq(){
5899         return Outs.EnableReq();
5900     }
5901 protected:
5903     // Construction
5904     INode(IPort &port, multiaddr theAddr,
5905         IInfo &info, IMotion &motion, IStatus &status,
5906         ILimits &limits, IOuts &outs, INodeAdv &adv, ISetup &setup,
5907         INodeEx &ex);
5908 public:
5909     // Destruct
5910     virtual ~INode() {};
5912 };
5913 //*
5914 //*****************************************************************************
5915
5916
5917 //*****************************************************************************
5918 // NAME*
5919 //  SysManager class
5946 class MN_EXPORT SysManager
5947 {
5948 public:
5983     class MN_EXPORT ThreadLock {
5984     public:
```

```
5989            ThreadLock();
5991            ~ThreadLock();
5993        };
5995    private:
5996        /*
5997        Port Specifications
5998        */
5999        portSpec m_ports[NET_CONTROLLER_MAX];
6000        // Count specified from last PortsOpen call
6001        size_t m_portOpenCount;
6003    public:
6004
6005
6012        static void FindComHubPorts(std::vector<std::string>&
    comHubPorts);
6013
6014        #if defined(_WIN32)||defined(_WIN64)
6015
6037        void ComHubPort(size_t netNumber,
6038            int portNumber,
6039            netRates portRate = MN_BAUD_12X);
6040        #endif
6041
6042        void ComHubPort(size_t netNumber,
6043            const char *portPath,
6044            netRates portRate = MN_BAUD_12X);
6055        portSpec &PortSetup(size_t netNumber);
6083        void PortsOpen(size_t portCount);
6084
6090        void PortsOpen(int portCount);
6091
6100        void PortsClose();
6101
6115        void RestartCold();
6116
6128        void RestartWarm();
6129
6157        double TimeStampMsec();
6158
6179        std::string &TimeStampMsecStr(std::string &toUpdString);
6180
6196        std::string TimeStampMsecStr();
6197
6215        void Delay(uint32_t msec);
6216
6228        IPort &Ports(size_t index);
6229
```

```
    INode &NodeGet(multiaddr theMultiAddr);
    // Constructor
    SysManager();
    ~SysManager();
};
//*
//*****************************************************************
#if defined(__GNUC__)
#pragma GCC diagnostic pop
#endif


//*****************************************************************
}  // namespace sFnd
//*****************************************************************
#endif
//=================================================================
//  END OF FILE pubSysCls.h
//=================================================================
```

sFnd::IAttnPort Member List

This is the complete list of members for **sFnd::IAttnPort**, including all inherited members.

AttnHandler(mnAttnCallback theNewHandler)	**sFnd::IAttnPort**	
Enable(bool newState)=0	**sFnd::IAttnPort**	pure virtual
Enabled()=0	**sFnd::IAttnPort**	pure virtual
HasAttnHandler()	**sFnd::IAttnPort**	
Supported()	**sFnd::IObj**	virtual

sFnd::IBrakeControl Member List

This is the complete list of members for **sFnd::IBrakeControl**, including all inherited members.

BrakeEngaged(size_t brakeNum)=0	**sFnd::IBrakeControl**	pure virtual
BrakeSetting(size_t brakeNum, BrakeControls brakeMode)=0	**sFnd::IBrakeControl**	pure virtual
BrakeSetting(size_t brakeNum)=0	**sFnd::IBrakeControl**	pure virtual
Supported()	**sFnd::IObj**	virtual

sFnd::IGrpShutdown Member List

This is the complete list of members for **sFnd::IGrpShutdown**, including all inherited members.

ShutdownInitiate()=0	**sFnd::IGrpShutdown**	pure virtual
ShutdownWhen(size_t nodeIndex, const ShutdownInfo &theInfo)=0	**sFnd::IGrpShutdown**	pure virtual
ShutdownWhenGet(size_t nodeIndex, ShutdownInfo &theInfo)=0	**sFnd::IGrpShutdown**	pure virtual
Supported()	**sFnd::IObj**	virtual

CPM NodeStops and E-Stops to Cancel Motion

The *NodeStop* feature is a part of the motion safety system. When asserted, a specified action is taken against the current motion and conditionally against future motion.

Creating NodeStops

The following actions can cause a *NodeStop* when properly configured:

- If the *Network Watchdog* has expired due to lack of host traffic.
- The **sFnd::IMotion::NodeStop** function will send a NodeStop to the node.
- The **sFnd::IPort::NodeStop** function will send a NodeStop to each node connected to the port.
- The **sFnd::IGrpShutdown** object can define when a "Group Shutdown" can be triggered, and whether particular nodes will respond to a "Group Shutdown".
- A Group NodeStop is automatically sent when 24V is removed from the Global Stop Input on an SC hub.

The **NodeStop Command** allows the selection of a desired stopping style. For all other methods, the node utilizes the stop style programmed into the *Default NodeStop* configured in the ClearView software.

The **Stop Type Codes** provide a convenient mechanism for issuing common NodeStops.

Stop Types

There are three basic stopping styles:

- **Abrupt** (STOP_TYPE_ABRUPT)
- **Ramp to Zero Speed using active decel rate** (STOP_TYPE_RAMP_AT_DECEL)
- **Ramp to Zero Speed using stop decel rate** (STOP_TYPE_RAMP)

Remarks

The NodeStop persists only for one sample-time in the *Status Register* unless the stop type includes E-Stop In this case the bit will stay set until the E-Stop state is cleared.

Stop Type and Modifier Priorities

If a node is in the process of stopping with a ramp-style stop and another NodeStop type is issued, there are rules that govern which stop types have priority, as well as how NodeStop modifiers of either of the stop types are handled. An example of when this might happen is if, during a ramp-style node stop of *E-Stop* type, a limit switch is activated and the limit stop style is an abrupt stop.

In general, an abrupt-style stop will always override a ramp-style stop. Additionally, any of the modifier bits (E-Stop or Controlled Outputs) that are set in either of the NodeStops will persist after the motion has stopped. In the example given above, this means that the E-Stop ramp will begin to stop the motion, then the motion will abruptly stop when the abrupt-style stop is issued, and the node will be in the E-Stop state at the end of the move.

Group Shutdown

The *Group Shutdown* feature allows an electrical connection or a change in a node's status to initiate a *Group Shutdown Node Stop*. This Group Shutdown will automatically propagate and initiate a *NodeStop* on all nodes in the port.

Note

The **sFnd::IGrpShutdown** object can be used to configure how a node responds to a group shutdown.

A normally closed switch is attached to the global stop Input on the SC hub board to "arm" the NodeStop. If this switch is opened or the wire to the input is removed, a group shutdown will be generated. This provides a level of fail safe should the wire to the switch become disconnected.

NodeStop Codes

NodeStopCodes	Description
STOP_TYPE_ABRUPT	Stop the executing motion and flush any pending motion commands immediately.
STOP_TYPE_RAMP	Flush any pending motion commands and ramp the node's speed to zero using the *E-Stop Deceleration Rate.*
STOP_TYPE_RAMP_AT_DECEL	Stop Motion with Deceleration at the active decel rate and flush any pending motion commands immediately.
STOP_TYPE_ESTOP_ABRUPT	• Stop the executing motion and flush any pending motion commands immediately. • Future motion is blocked until a directed **NodeStopClear** is sent.

STOP_TYPE_ESTOP_RAMP	• Flush any pending motion commands and ramp the node's speed to zero using the *E-Stop Deceleration Rate.* • Future motion is blocked until a directed **NodeStopClear** is sent.
STOP_TYPE_ESTOP_RAMP_AT_DECEL	• Flush any pending motion commands and ramp the node's speed to zero using the *active decel rate.* • Future motion is blocked until a directed **NodeStopClear** is sent.
STOP_TYPE_DISABLE_ABRUPT	• Stop the executing motion and flush any pending motion commands immediately. • Once the node has stopped,

	the node will be disabled.
STOP_TYPE_DISABLE_RAMP	• Flush any pending motion commands and ramp the node's speed to zero using the *E-Stop Deceleration Rate.* • Once the node has stopped, the node will be disabled.
STOP_TYPE_DISABLE_RAMP_AT_DECEL	• Flush any pending motion commands and ramp the node's speed to zero using the *active decel rate.* • Once the node has stopped, the node will be disabled.

This is an example calling different types of NodeStops

```
// Stop the node with a normal Abrupt NodeStop
mynode.motion.NodeStop(STOP_TYPE_ABRUPT);
// Issue a ramped E-stop
mynode.motion.NodeStop(STOP_TYPE_ESTOP_RAMP);
// Issue an NodeStop to each node on a Port
myport.motion.NodeStop(STOP_TYPE_ABRUPT);
// Clear the E-Stop condition on mynode
mynode.motion.NodeStopClear();
```

sFoundation 2.0

sFnd::IHoming Member List

This is the complete list of members for **sFnd::IHoming**, including all inherited members.

HomingValid()=0	**sFnd::IHoming**	pure virtual
Initiate()=0	**sFnd::IHoming**	pure virtual
IsHoming()=0	**sFnd::IHoming**	pure virtual
Node()	**sFnd::IObjWithNode**	inline
SignalComplete()=0	**sFnd::IHoming**	pure virtual
SignalInvalid()=0	**sFnd::IHoming**	pure virtual
Supported()	**sFnd::IObj**	virtual
WasHomed()=0	**sFnd::IHoming**	pure virtual

sFnd::IInfo Member List

This is the complete list of members for **sFnd::IInfo**, including all inherited members.

_nodeTypes enum name	**sFnd::IInfo**
CLEARPATH_SC enum value	**sFnd::IInfo**
CLEARPATH_SC_ADV enum value	**sFnd::IInfo**
Ex	**sFnd::IInfo**
FirmwareVersion	**sFnd::IInfo**
Model	**sFnd::IInfo**
Node()	**sFnd::IObjWithNode**
NodeType()=0	**sFnd::IInfo**
nodeTypes typedef	**sFnd::IInfo**
PositioningResolution	**sFnd::IInfo**
SerialNumber	**sFnd::IInfo**
Supported()	**sFnd::IObj**
UNKNOWN enum value	**sFnd::IInfo**
UserData(size_t bank, const uint8_t m_lastUserData[MN_USER_NV_SIZE])=0	**sFnd::IInfo**
UserData(size_t bank)=0	**sFnd::IInfo**
UserID	**sFnd::IInfo**
UserRAM	**sFnd::IInfo**

sFoundation 2.0

Example-MultiThreaded/Axis.h

Return to Example-MultiThreaded

Related Files:

- Main Function File: Example-MultiThreaded.cpp
- Axis Class Implementaion: Axis.cpp
- Supervisor Class Header File: Axis.h
- Supervisor Class Implementaion: Axis.cpp

```
/*************************************************************************
     ****
 * $Source: $
 * $Revision: 7 $ $State: $ $Date: 6/02/16 5:32p $
 *
 * !NAME!
 *      Axis
 *
 * DESCRIPTION:
 *      This class provides an interface to the ClearPath SC node
 *      in our example program
 *
 * PRINCIPLE AUTHORS:
 *      JS
 *
 * CREATION DATE:
 *      03/30/2016
 *
 * !end!
     *
 *************************************************************************

#ifndef __AXIS_H__
#define __AXIS_H__

/*************************************************************************
     ****
 * !NAME!
     *
 *  Axis.h headers
 *
     !0! */
#include <thread>
#include <condition_variable>
#include <mutex>
#include "pubSysCls.h"
using namespace sFnd;
```

```
using namespace std;
/*                                                                      !end!
 ************************************************************************

/*********************************************************************
     ****
 * !NAME!
     *
 *  Axis.h constants
 *
     !0! */
#define RAS_CODE            3
#define TRQ_PCT             100
#define ACC_LIM_RPM_PER_SEC 8000
#define VEL_LIM_RPM         2000
#define TRACKING_LIM        (m_node->Info.PositioningResolution.Value() /
     4)
/* !end!
     *
 ************************************************************************

// This class needs to have a reference to its Supervisor
class Supervisor;

//*********************************************************************
     *******
// NAME
     *
//  Axis class
//
// DESCRIPTION
//  This is the interface to the ClearPath SC. It includes a thread
//  which runs the state machine for the node.
//
class Axis {
private:
    Supervisor *m_super;        // The supervisor
    INode *m_node;              // The ClearPath-SC for this axis
    thread m_thread;            // Handle to this Axis's thread

    // Filename for the config file stored at the start
#define MAX_CONFIG_PATH_LENGTH 256
#define MAX_CONFIG_FILENAME_LENGTH 384
    char m_configFile[MAX_CONFIG_FILENAME_LENGTH];

    // State machine information
    enum StateEnum {
        STATE_IDLE,
        STATE_WAIT_FOR_KICK,
        STATE_SEND_MOVE,
        STATE_SIGNAL_MOVE_SENT,
        STATE_WAIT_FOR_MOVE_DONE,
        STATE_EXITED
    };
```

```
    StateEnum m_state;
    StateEnum m_lastState;

    // Move information
    mgMoveProfiledInfo m_move;
    ::uint32_t m_moveCount;
    ::int32_t m_moveTimeoutMs;

    bool m_quitting;            // Axis quitting

    // Synchronization
    condition_variable m_cond;
    mutex m_mutex;

    // Enable the node and get it ready to go
    void Enable();

    // Initialize accLim, velLim, etc
    void InitMotionParams();

    // Initiate a move
    void Move();

    // Wait for attention that move has completed
    bool WaitForMove(::int32_t timeoutMs);

    // The state machine
    void AxisMain(Supervisor *theSuper);

public:
    // Constructor/Destructor
    Axis(INode *node);
    ~Axis();

    // Return a reference to our node
    INode* MyNode() {
        return(m_node);
    };

    // Print the current stats (number of moves performed and
    // current measured position)
    void PrintStats(bool onInit = false){
        m_node->Motion.PosnMeasured.Refresh();
        if (onInit){
            printf("  [%2d]:\t\t**at startup**\t\t%8.0f\n", m_node-
    >Info.Ex.Addr(),
                m_node->Motion.PosnMeasured.Value());
        }
        else{
            printf("  [%2d]:\t\t%8d\t\t%8.0f\n", m_node->Info.Ex.Addr(), m_m
                m_node->Motion.PosnMeasured.Value());
        }
    }
```

```
    // Set the move distance to the given number of revolutions
    void SetMoveRevs(int numRevs){
        m_move.value = (long)(m_node->Info.PositioningResolution.Value())*nu
        printf("[%s] m_move.value=%d\n", m_node->Info.UserID.Value(), m_move
        m_node->Motion.Adv.HeadDistance = m_move.value/4;
        m_node->Motion.Adv.TailDistance = m_move.value/4;
    }

    // Create the thread and get it going
    void CreateThread(Supervisor *theSuper){
        m_thread = thread(&Axis::AxisMain, this, theSuper);
    };

    // Reset sequencer to "idle" state
    void ResetToIdle(){
        if (m_state != STATE_IDLE)
            m_state = m_lastState = STATE_IDLE;
    };

    // Time to quit
    void Quit(){
        m_quitting = true;
    }

    // Park here to wait for the thread to exit
    void Join(){
        m_thread.join();
    }
};
//
     *
//**********************************************************************
     *******

#endif
```

sFoundation 2.0

Example-SingleThreaded/Axis.h

Return to **Example-SingleThreaded**

Related Files:

- Main Function File: **Example-SingleThreaded.cpp**
- Axis Class Implementaion: **Axis.cpp**

```
/*************************************************************************
    ****
 * NAME:
 *      Axis
 *
 * DESCRIPTION:
 *      This class provides an interface to the ClearPath SC node
 *      in our example program
 *
 *************************************************************************

#ifndef __AXIS_H__
#define __AXIS_H__

 /************************************************************************
    *****
 * !NAME!
    *
 *  Axis.h headers
 *
    !0! */
#include <time.h>
#include "pubSysCls.h"
#if defined(_WIN32) || defined(_WIN64)
#include "windows.h"
#endif

// Use the Teknic library's namespace
using namespace sFnd;
// Also use the standard namespace
using namespace std;
/*                                                                    !end!
*************************************************************************
    ***/

/************************************************************************
    ****
* !NAME!
    *
```

```
*   Axis.h constants
*
    !0! */
#define RAS_CODE            3
#define TRQ_PCT             100
#define ACC_LIM_RPM_PER_SEC 8000
#define VEL_LIM_RPM         2000
#define TRACKING_LIM        (m_node->Info.PositioningResolution.Value() /
    4)
/* !end!
    *
*************************************************************************
    ***/

//*************************************************************************
    *******
// NAME
    *
//  Axis class
//
// DESCRIPTION
//  This is the interface to the ClearPath SC.
//
class Axis {
private:
    INode *m_node;              // The ClearPath-SC for this axis
    SysManager& m_sysMgr;

// Filename for the stored config file
#define MAX_CONFIG_PATH_LENGTH 256
#define MAX_CONFIG_FILENAME_LENGTH 384
    char m_configFile[MAX_CONFIG_FILENAME_LENGTH];

    // State machine information
    enum StateEnum {
        STATE_IDLE,
        STATE_SEND_MOVE,
        STATE_WAIT_FOR_MOVE_DONE,
        STATE_EXITED
    };
    StateEnum m_state;
    StateEnum m_lastState;

    // Move information
    uint32_t m_moveCount;
    int32_t m_positionWrapCount;
    int32_t m_moveTimeoutMs;

    bool m_quitting;            // Axis quitting

    // Enable the node and get it ready to go
    void Enable();

    // Initialize accLim, velLim, etc
```

```
    void InitMotionParams();

    // Edit the current motion parameters
    void ChangeMotionParams(double newVelLimit,
        double newAccelLimit,
        uint32_t newJrkLimit,
        uint32_t newTrackingLimit,
        int32_t newTrqGlobal);

    // Initiate a move
    void PosnMove(int32_t targetPosn,
        bool isAbsolute = false,
        bool addDwell = false);
    void VelMove(double targetVel);

    // Wait for attention that move has completed
    bool WaitForMove(::int32_t timeoutMs, bool secondCall = false);

    // Alert handlers
    void CheckForAlerts();

    // HW Brake Setup
    void SetBrakeControl(size_t brakeNum, BrakeControls brakeMode);

    // Return scaled position
    int64_t GetPosition(bool includeWraps = true);

public:
    // Constructor/Destructor
    Axis(SysManager&, INode *node);
    ~Axis();

    // The state machine
    void AxisMain();

    // Return a reference to our node
    INode* MyNode() {
        return(m_node);
    };

    // Print the current stats (number of moves performed and
    // current commanded position)
    void PrintStats(bool onInit = false) {
        // Refresh the measures position register
        m_node->Motion.PosnCommanded.Refresh();
        if (onInit) {
            printf("  [%2d]:\t\t**at startup**\t\t%jd\n",
                m_node->Info.Ex.Addr(),
                (intmax_t) GetPosition());
        }
        else {
            printf("  [%2d]:\t\t%8d\t\t%jd\n",
                m_node->Info.Ex.Addr(), m_moveCount,
                (intmax_t) GetPosition());
```

```
        }
    }

    // Reset sequencer to "idle" state
    void ResetToIdle() {
        if (m_state != STATE_IDLE)
            m_state = m_lastState = STATE_IDLE;
    };

    // Time to quit
    void Quit() {
        m_quitting = true;
    }
};
//
     *
//*************************************************************************
     *******

#endif
```

Saving/Loading Node Configurations

Node Configuration Files

One of the most powerful features of the sFoundation library is the ability to automatically save, and load configuration files. A configuration file is a file containing all application parameters in a node including; tuning parameters, setups for inputs, torque limits, homing parameters, limit modes, operating mode parameters etc. Most of these parameters will be configured through the ClearView user interface, or through the Auto-Tuning process.

This feature allows you to load/save hundreds of parameters to the Node, ensuring the correct parameters are loaded on machine start-up, and allowing you to save a snapshot of parameters in the event of an unexpected shutdown.

See also

sFnd::ISetup::ConfigLoad

sFnd::ISetup::ConfigSave

---

sFnd::IInfoEx Member List

This is the complete list of members for **sFnd::IInfoEx**, including all inherited members.

Addr()	**sFnd::IInfoEx**	
Node()	**sFnd::IObjWithNode**	inline
NodeIndex()	**sFnd::IInfoEx**	
Parameter(nodeparam index)=0	**sFnd::IInfoEx**	pure virtual
Parameter(nodeparam index, double newValue)=0	**sFnd::IInfoEx**	pure virtual
Supported()	**sFnd::IObj**	virtual

sFoundation 2.0

sFnd::ILimits Member List

This is the complete list of members for **sFnd::ILimits**, including all inherited members.

Node()	**sFnd::IObjWithNode**	inline
PosnTrackingLimit	**sFnd::ILimits**	
Supported()	**sFnd::IObj**	virtual
TrqGlobal	**sFnd::ILimits**	

sFnd::IMotion Member List

This is the complete list of members for **sFnd::IMotion**, including all inherited members.

AccLimit	sFnd::IM
AddToPosition(double adjAmount)=0	sFnd::IM
Adv	sFnd::IM
DwellMs	sFnd::IM
GroupNodeStop(nodeStopCodes howToStop)=0	sFnd::IM
Homing	sFnd::IM
IsReady()=0	sFnd::IM
MoveIsDone()=0	sFnd::IM
MovePosnDurationMsec(int32_t target, bool targetIsAbsolute=false)	sFnd::IM
MovePosnStart(int32_t target, bool targetIsAbsolute=false, bool addPostMoveDwell=false)=0	sFnd::IM
MoveVelDurationMsec(double target)	sFnd::IM
MoveVelStart(double target)=0	sFnd::IM
MoveWentDone()=0	sFnd::IM
Node()	sFnd::IC
NodeStop(nodeStopCodes howToStop)=0	sFnd::IM
NodeStopClear()=0	sFnd::IM
PosnCommanded	sFnd::IM
PosnMeasured	sFnd::IM
PosnTracking	sFnd::IM
Supported()	sFnd::IC
TrqCommanded	sFnd::IM
TrqMeasured	sFnd::IM
VelCommanded	sFnd::IM
VelLimit	sFnd::IM
VelMeasured	sFnd::IM
VelocityAtTarget()=0	sFnd::IM
VelocityReachedTarget()=0	sFnd::IM
WentNotReady()=0	sFnd::IM

sFnd::IMotionAdv Member List

This is the complete list of members for **sFnd::IMotionAdv**, including all inherited members.

AfterStartDistance
BeforeEndDistance
DecelLimit
HeadDistance
HeadTailVelLimit
MovePosnAsymDurationMsec(int32_t targetPosn, bool targetIsAbsolute=false)
MovePosnAsymStart(int32_t targetPosn, bool targetIsAbsolute=false, bool isTriggered=false, bool has
MovePosnDurationMsec(int32_t targetPosn, bool targetIsAbsolute=false)
MovePosnHeadTailDurationMsec(int32_t targetPosn, bool targetIsAbsolute=false, bool hasHead=true
MovePosnHeadTailStart(int32_t targetPosn, bool targetIsAbsolute=false, bool isTriggered=false, bool h
MovePosnStart(int32_t targetPosn, bool targetIsAbsolute=false, bool isTriggered=false, bool hasDwell=
MoveVelDurationMsec(double targetCountsPerSec)
MoveVelStart(double targetPosn, bool isTriggered=false)=0
Node()
Supported()
TailDistance
TriggerGroup()=0
TriggerGroup(size_t groupNumber)=0
TriggerMove()=0
TriggerMovesInMyGroup()=0

Advanced Move Types

This section describes advanced move types only available on Advanced SC Nodes. For a system with both basic and advanced nodes you can use the **sFnd::IObj::Supported**, or **sFnd::IInfo::NodeType** functions to determine whether the **IMotionAdv** object is supported.

Head/Tail Moves

Head/Tail moves are a special type of positional move with additional sections at the beginning and/or end of the move. These Head or Tail sections are defined by relative distances from the beginning, and/or end of the commanded move. Head/Tail sections have alternate velocities compared to the main part of the move making these moves useful for applications such as grippers and plunger style axes. They combine a high speed "get it close" section with a more gentler "until the axis touches the part" section. Pick and place machines and test tube grabbers can make good use of these style moves.

Once a Head/Tail move is commanded, the node will accelerate at the **Accel Limit** until it reaches the **Head/Tail Velocity limit** at which point the node will continue until it has traveled the **Head Distance** from the start of the move. After this, the node will Accelerate at the **Accel Limit** until it reaches the **Velocity limit**. The node will continue at this velocity until the node is at the specified **tail distance** away from the end of the move. The node will then start to decelerate at the **Accel Limit** until it reaches the **Head/Tail Velocity limit** which it will remain at until the move terminates at the final position.

Note

The head or tail sections of a move may be merged if the overall distances are too short.

Asymmetrical Moves

Asymmetrical moves are a special type of positional move with different Acceleration, and Deacceleration parameters. These moves behave the same as normal **Trapezoidal Moves** except the **Acceleration** and **Deacceleration** parameters are different. Asymmetrical moves are useful for increasing throughput on mechanics that may be able to handle higher accelerations in certain cases.

Constraint Parameters

The following table provides a list of constraint parameters related to the moves:

Code	Description	Parameter
V-Lim	Velocity Limit constraint.	**sFnd::IMotion::VelLimit**
A-Lim	Acceleration Limit constraint.	**sFnd::IMotion::AccLimit**
D-Lim	Deceleration Limit constraint. (*Asymmetrical Moves Only*)	**sFnd::IMotionAdv::DecelLimit**
@Head	The distance from start of move that the *Head/Tail Velocity Limit* becomes the *Velocity Limit* constraint.	**sFnd::IMotionAdv::HeadDistance**
@Tail	The distance from end of the move that the *Head/Tail Velocity Limit* becomes the *Velocity Limit* constraint.	**sFnd::IMotionAdv::TailDistance**
Head/Tail V-Lim	This is the *Velocity Limit* constraint during a head or tail section of a	**sFnd::IMotionAdv::HeadTailVelLimit**

	move(Advanced SC nodes only).	

Advanced Motion Profiles

The following diagrams show the various velocity verses time profiles for all the moves.

Trapezoidal Style

Asymmetrical Style

Head Style

Tail Style

Head/Tail Style

Velocity Style

See also

sFnd::IMotionAdv::MovePosnHeadTailStart Function for Initiating Head/Tail Moves

sFnd::IMotionAdv::MovePosnAsymStart Function for Initiating Asymmetrical Moves

CPM NodeStops and E-Stops to Cancel Motion For cancelling commanded moves

Triggered Moves

Overview

Triggering Moves

The *Trigger* feature allows the user application to set up a motion command in a ClearPath SC motor that waits for a *Trigger Event* to start motion. This feature can reduce the latency and increase synchronization of motion between multiple nodes. These performance enhancements are accomplished by masking the motion command download with the execution initiation.

Triggered move requests are downloaded to the motor and forced to await an internal triggering event before starting. Transmission time of the command may be masked using this method. Alternately you may use an external signal to release a waiting move.

A trigger event only releases a waiting triggered move and is ignored by any other types of queued moves in the move buffer. A Trigger Event is also ignored if the Node is currently executing a move.

The **sFnd::IMotionAdv::TriggerMove** function is a common host based method for starting a waiting move on any particular node.

Note

Triggered moves are only available in Advance ClearPath SC Nodes.

See also

Move Types for different move types

Advanced Move Types for additional move types available on Advanced ClearPath-SC motors

Move Done Feature for determining when a move has completed

The Trigger Command

The trigger command comes in two forms: *Node Addressed* and *Broadcast*. The *Node Addressed* form will generate a trigger event only at the addressed node (**sFnd::IMotionAdv::TriggerMove**).

In the *Broadcast* case, the address takes the form of a *Trigger Group*. Trigger events are generated at each node that has a matching *Trigger Group*. The *Trigger Group* of a node can be checked or changed through the **sFnd::IMotionAdv::TriggerGroup** functions.

To trigger a *Trigger Group*, call either the **sFnd::IMotionAdv::TriggerMovesInMyGroup** function from an instance of a Node in the group or call the Port level function **sFnd::IPortAdv::TriggerMovesInGroup** to specify a specify a group directly.

Trigger Groups are often used in multi-axis systems where there are times when a small group of nodes have related motion. For example: a system may have two sections of X-Y-Z axes. With a multithreaded application there will most likely be an execution thread dedicated to each group of axes. By assigning each group of axes to a unique *Trigger Group*, the threads may issue a *Trigger Command* to the associated group and not have to serialize the issuance of the moves and the triggering command.

Input-Based Triggers

A Node can also be configured to issue a *Node Addressed* trigger upon the assertion of the Node's input A or B. Wiring a single electrical signal to multiple node's inputs allows for these to be triggered simultaneously. To configure a Node for Input based triggers, use the ClearView input actions dialog for each Node(see below).

Configuring Inputs Through Clearview

sFoundation 2.0

sFnd::INode Member List

This is the complete list of members for **sFnd::INode**, including all inherited members.

_accUnits enum name	**sFnd::INode**	
_trqUnits enum name	**sFnd::INode**	
_velUnits enum name	**sFnd::INode**	
AccUnit(enum _accUnits newUnits)=0	**sFnd::INode**	pure virtual
AccUnit()	**sFnd::INode**	inline
Adv	**sFnd::INode**	
AMPS enum value	**sFnd::INode**	
COUNTS_PER_SEC enum value	**sFnd::INode**	
COUNTS_PER_SEC2 enum value	**sFnd::INode**	
EnableReq(bool newState)	**sFnd::INode**	inline
EnableReq()	**sFnd::INode**	inline
Info	**sFnd::INode**	
Limits	**sFnd::INode**	
Motion	**sFnd::INode**	
PCT_MAX enum value	**sFnd::INode**	
Port	**sFnd::INode**	
RPM enum value	**sFnd::INode**	
RPM_PER_SEC enum value	**sFnd::INode**	
Setup	**sFnd::INode**	

Status	**sFnd::INode**	
TrqUnit(enum _trqUnits newUnits)=0	**sFnd::INode**	pure virtual
TrqUnit()	**sFnd::INode**	inline
VelUnit(enum _velUnits newUnits)=0	**sFnd::INode**	pure virtual
VelUnit()	**sFnd::INode**	inline

sFnd::INode::UseMutex Member List

This is the complete list of members for **sFnd::INode::UseMutex**, including all inherited members.

UseMutex(INode &ourNode)	**sFnd::INode::UseMutex**

sFnd::INodeAdv Member List

This is the complete list of members for **sFnd::INodeAdv**, including all inherited members.

Attn	**sFnd::INodeAdv**	
Node()	**sFnd::IObjWithNode**	inline
Supported()	**sFnd::IObj**	virtual

sFnd::IObj Member List

This is the complete list of members for **sFnd::IObj**, including all inherited members.

Supported() **sFnd::IObj** virtual

sFoundation 2.0

sFnd::IObjWithNode Member List

This is the complete list of members for **sFnd::IObjWithNode**, including all inherited members.

Node()	**sFnd::IObjWithNode**	inline
Supported()	**sFnd::IObj**	virtual

sFoundation 2.0

sFnd::IObjWithPort Member List

This is the complete list of members for **sFnd::IObjWithPort**, including all inherited members.

Supported() **sFnd::IObj** virtual

sFnd::IPort Member List

This is the complete list of members for **sFnd::IPort**, including all inherited members.

Adv	**sFnd::IPort**	
BrakeControl	**sFnd::IPort**	
CommandTraceSave(const char *filePath)=0	**sFnd::IPort**	pure virtual
GrpShutdown	**sFnd::IPort**	
NodeCount()=0	**sFnd::IPort**	pure virtual
Nodes(size_t index)	**sFnd::IPort**	
NodeStop(mgNodeStopReg stopType=STOP_TYPE_IGNORE)=0	**sFnd::IPort**	pure virtual
OpenState()=0	**sFnd::IPort**	pure virtual
RestartCold()=0	**sFnd::IPort**	pure virtual
WaitForOnline(int32_t timeoutMsec=15000)=0	**sFnd::IPort**	pure virtual

sFnd::IPortAdv Member List

This is the complete list of members for **sFnd::IPortAdv**, including all inherited members.

Attn	**sFnd::IPortAdv**	
Supported()	**sFnd::IPortAdv**	virtual
TriggerMovesInGroup(size_t groupNumber)=0	**sFnd::IPortAdv**	pure virtual

sFnd::ISetup Member List

This is the complete list of members for **sFnd::ISetup**, including all inherited members.

AccessLevelIsFull()=0	**sFnd::ISetup**	pure virtual
ConfigLoad(const char *filePath, bool doReset=false)=0	**sFnd::ISetup**	pure virtual
ConfigSave(const char *filePath)=0	**sFnd::ISetup**	pure virtual
DelayToDisableMsecs	**sFnd::ISetup**	
Ex	**sFnd::ISetup**	
Node()	**sFnd::IObjWithNode**	inline
Supported()	**sFnd::IObj**	virtual

sFnd::ISetupEx Member List

This is the complete list of members for **sFnd::ISetupEx**, including all inherited members.

NetWatchdogMsec	**sFnd::ISetupEx**	
Node()	**sFnd::IObjWithNode**	inline
Supported()	**sFnd::IObj**	virtual

sFoundation 2.0

sFnd::IStatus Member List

This is the complete list of members for **sFnd::IStatus**, including all inherited members.

Accum	**sFnd::IStatus**	
Adv	**sFnd::IStatus**	
Alerts	**sFnd::IStatus**	
AlertsClear()=0	**sFnd::IStatus**	pure virtual
Fall	**sFnd::IStatus**	
HadTorqueSaturation()=0	**sFnd::IStatus**	pure virtual
IsReady()=0	**sFnd::IStatus**	pure virtual
Node()	**sFnd::IObjWithNode**	inline
Power	**sFnd::IStatus**	
Rise	**sFnd::IStatus**	
RMSlevel	**sFnd::IStatus**	
RT	**sFnd::IStatus**	
Supported()	**sFnd::IObj**	virtual

sFoundation 2.0

sFnd::IStatusAdv Member List

This is the complete list of members for **sFnd::IStatusAdv**, including all inherited members.

CapturedHiResPosn	**sFnd::IStatusAdv**	
CapturedPos	**sFnd::IStatusAdv**	
Node()	**sFnd::IObjWithNode**	inline
Supported()	**sFnd::IObj**	virtual

Position Capture

In some applications, it is required to know the exact position at a certain point in time. For these applications, it is possible to have the motor automatically capture position upon the assertion, or de-assertion, of an electrical input. This feature is set up using the ClearView interface, under the Input Actions portion of the setup menu(see below).

There are two position capture parameters available. The first is the High Resolution Position capture **sFnd::IStatusAdv::CapturedHiResPosn** which captures the position within 0.1us of the state change of the input A.

The second is the Standard resolution position capture **sFnd::IStatusAdv::CapturedPos** which captures the position within 100 us of the state change of the input B.

Note

Position capture is only available on Advanced ClearPath-SC nodes

See also

sFnd::IStatusAdv for more information on accessing these values.

Configuring Inputs for Position Capture

sFnd::SysManager Member List

This is the complete list of members for **sFnd::SysManager**, including all inherited members.

ComHubPort(size_t netNumber, int portNumber, netRates portRate=MN_BAUD_12X)	**sFnd::SysMana**
Delay(uint32_t msec)	**sFnd::SysMana**
FindComHubPorts(std::vector< std::string > &comHubPorts)	**sFnd::SysMana**
NodeGet(multiaddr theMultiAddr)	**sFnd::SysMana**
Ports(size_t index)	**sFnd::SysMana**
PortsClose()	**sFnd::SysMana**
PortsOpen(size_t portCount)	**sFnd::SysMana**
PortsOpen(int portCount)	**sFnd::SysMana**
RestartCold()	**sFnd::SysMana**
RestartWarm()	**sFnd::SysMana**
TimeStampMsec()	**sFnd::SysMana**
TimeStampMsecStr(std::string &toUpdString)	**sFnd::SysMana**
TimeStampMsecStr()	**sFnd::SysMana**

sFoundation 2.0

Class Library Overview

Class Library

The sFoundation comprises three main classes: a "Node" class, a "Port" class and a "System" class. The term "Node" in ClearPath-SC terminology means an individual ClearPath-SC motor, so most of your application software will access Node class objects. You can think of the sFoundation structure as a hierarchy where an instance of a System class object “owns” all of the instances of the Port class objects, which in turn “own” all of the instances of the Node class objects.

You use Node class objects to, for example, create and execute positional moves, and query real-time Node status (e.g., Is a Node enabled? Is a move done?, etc.).

Port class objects let you manage individual ports, as well as the collection of all Nodes on a given port. Port class object also provide access to port level features such as general purpose outputs, and group shutdowns.

Finally, a System class object is used to define and configure all of the USB or serial ports in your system, as well as access system timestamps, thread safe delays and provides a threadlocking mechanism.

Class Hierarchy

Node Level Classes
One instance of each class for every ClearPath-SC motor
ISetup
Node Setup Interface
IInfo
Node Information Interface
ILimits
Torque and Position Limits Interface
INode
Node Interface
IStatus
Node Status Interface
INodeAdv
Advanced Node Interface
IMotion
Motion Control Interface
IStatusAdv
Advanced Node Status Interface
IHoming
Homing Interface
IMotionAdv
Advanced Motion Control Interface
Port Level Classes
One instance of each class for every USB/Serial Port
IPort
Serial Port Interface
IGrpShutdown
Group Shutdown Interface
IBrakeControl
Brake Control Interface
IPortAdv
Advanced Port Interface
System Level Classes
Only one instance of this class for each application
SysManager
sFoundation Top Level System Object

System Level

System Level functions and structures can be accesed from the **SysManager class.** This is the main application class used to interface with Teknic nodes attached to various serial and USB ports. Its purpose is to coordinate the actions among the various ports and within nodes. This is sometimes called the root object for the sFoundation.

Common System Level features are:

- Port Specification
- Opening ports and detecting nodes attached to them
- Closing the system gracefully
- Network maintenance features, such as restarting nodes, or re-enumerating the nodes on the port

See also

sFnd::SysManager for more information on System Level functions.

Port Level

Port Level functions and structures are defined separately for each port in a system. Therefore your application will have one instance of each of these structures for each serial port in your system. Port Level functions and structures can be accessed from the **Port class.** This class is used to interface with a serial port on your computer which electrically interfaces to a series of nodes.

Common Port Level features are:

- Node counts
- Restarting nodes on a port network
- Determining the port network operational state
- Creating command trace files
- Issuing *Node Stops* to nodes on this port
- Access to the Port Level functions of the SC-Hub board such as the *Group Shutdown* feature and the *Brake Control* feature
- Access to advanced Port Level features if your nodes support them

See also

sFnd::IPort for more information on Port Level Functions

Node Level

Node Level functions and structures are defined separately for each node in a system. Therefore your application will have one instance of each of these structures for each node in your system. Node Level functions and structures can be accessed from the **Node class.** This class is used to interface with all nodes in an application where each ClearPath-SC Motor is an individual node.

Common Node Level features are:

- Execute Motion
- Read real time status information from the node
- Read/Modify various node parameters
- Read Node Level inputs
- Issuing controlled *Node Stops* to the node
- Access to advanced Node Level features if your node supports them

See also

sFnd::INode for more information on Node Level Functions

Value Parameters

The ValueXXXX objects are required to be an intermediary between your application and the underlying node parameters. These classes provide convenient functions to refresh, set, clear, and test the value of Node parameters. Some of the parameters are Simple types, like **Signed Integers**, **Doubles** or **Strings**. Some parameters are register values with a variety of different fields (ie: **Alert Register**, **Status Register**).

There is a caching mechanism built in to reduce network traffic if a value is queried from multiple locations in your code. For dynamic data, such as position information, this cache can be used to keep a "central" copy of the current information. The *Refresh* function can be used to update the value from the node. The base class function **sFnd::ValueBase::IsVolatile** can be used to determine if this parameter is dynamic.

Many node parameters consist of two parts: a run-time value and a power-on default value which is stored in non-volatile memory. Most application level interactions should use the run-time value since adjustments to the run-time value go into effect immediately; whereas, an adjustment to the power-on default value will go into effect only after the node is power cycled.

A number of convenience members allow easy conversion from the parameter's native data type to other common data types. The underlying run-time value in its native format is always available via the *Value function*. The optional argument of these functions allows access to the power-on default value.

The non-volatile memory used to store the power-on default value is only capable of being written a finite number of times, and is quite slow in being written to. If you think you need to update the power-on default value of a parameter, you should check the current power-on default value before proceeding. You can safely

ignore the update if the current value matches the value to be written.

sFnd::SysManager::ThreadLock Member List

This is the complete list of members for **sFnd::SysManager::ThreadLock**, including all inherited members.

ThreadLock()	**sFnd::SysManager::ThreadLock**

sFnd::ValueAlert Member List

This is the complete list of members for **sFnd::ValueAlert**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Last()	**sFnd::ValueAlert**	inline
Node()	**sFnd::IObjWithNode**	inline
operator alertReg()	**sFnd::ValueAlert**	
operator=(alertReg &newValue)	**sFnd::ValueAlert**	
Refresh()	**sFnd::ValueAlert**	virtual
Supported()	**sFnd::IObj**	virtual
Value()	**sFnd::ValueAlert**	
Value(const alertReg &newValue)	**sFnd::ValueAlert**	

sFoundation 2.0

sFnd::ValueBase Member List

This is the complete list of members for **sFnd::ValueBase**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Node()	**sFnd::IObjWithNode**	inline
Refresh()=0	**sFnd::ValueBase**	pure virtual
Supported()	**sFnd::IObj**	virtual

sFnd::ValueDouble Member List

This is the complete list of members for **sFnd::ValueDouble**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Node()	**sFnd::IObjWithNode**	inline
operator double()	**sFnd::ValueDouble**	
operator int32_t()	**sFnd::ValueDouble**	
operator uint32_t()	**sFnd::ValueDouble**	
operator=(double newValue)	**sFnd::ValueDouble**	
Refresh()	**sFnd::ValueDouble**	virtual
Supported()	**sFnd::IObj**	virtual
Value(int32_t newValue, bool makeNonVolatile=false)	**sFnd::ValueDouble**	
Value(double newValue, bool makeNonVolatile=false)	**sFnd::ValueDouble**	
Value(bool getNonVolatile=false)	**sFnd::ValueDouble**	

sFoundation 2.0

sFnd::ValuePowerReg Member List

This is the complete list of members for **sFnd::ValuePowerReg**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Last()	**sFnd::ValuePowerReg**	inline
Node()	**sFnd::IObjWithNode**	inline
operator mnPowerReg()	**sFnd::ValuePowerReg**	
Refresh()	**sFnd::ValuePowerReg**	virtual
Supported()	**sFnd::IObj**	virtual
Value()	**sFnd::ValuePowerReg**	

sFnd::ValueSigned Member List

This is the complete list of members for **sFnd::ValueSigned**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Node()	**sFnd::IObjWithNode**	inline
operator double()	**sFnd::ValueSigned**	
operator int32_t()	**sFnd::ValueSigned**	
operator uint32_t()	**sFnd::ValueSigned**	
operator=(int32_t newValue)	**sFnd::ValueSigned**	
Refresh()	**sFnd::ValueSigned**	virtual
Supported()	**sFnd::IObj**	virtual
Value(int32_t newValue, bool makeNonVolatile=false)	**sFnd::ValueSigned**	
Value(bool getNonVolatile=false)	**sFnd::ValueSigned**	

sFnd::ValueStatus Member List

This is the complete list of members for **sFnd::ValueStatus**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Clear()	**sFnd::ValueStatus**	
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Last()	**sFnd::ValueStatus**	
Node()	**sFnd::IObjWithNode**	inline
operator mnStatusReg()	**sFnd::ValueStatus**	
operator=(mnStatusReg &newValue)	**sFnd::ValueStatus**	
Refresh()	**sFnd::ValueStatus**	virtual
Supported()	**sFnd::IObj**	virtual
TestAndClear(const mnStatusReg &mask, mnStatusReg &result)	**sFnd::ValueStatus**	
TestAndClear(const mnStatusReg &mask)	**sFnd::ValueStatus**	
Value()	**sFnd::ValueStatus**	
Value(const mnStatusReg &newValue)	**sFnd::ValueStatus**	

sFoundation 2.0

sFnd::ValueString Member List

This is the complete list of members for **sFnd::ValueString**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Node()	**sFnd::IObjWithNode**	inline
operator const char *()	**sFnd::ValueString**	
operator=(const char *newValue)	**sFnd::ValueString**	
Refresh()=0	**sFnd::ValueString**	pure virtual
Supported()	**sFnd::IObj**	virtual
Value(const char *newValue)=0	**sFnd::ValueString**	pure virtual
Value()=0	**sFnd::ValueString**	pure virtual

sFnd::ValueUnsigned Member List

This is the complete list of members for **sFnd::ValueUnsigned**, including all inherited members.

AutoRefresh(bool newState)	**sFnd::ValueBase**	inline
AutoRefresh()	**sFnd::ValueBase**	inline
Exists()	**sFnd::ValueBase**	inline
IsVolatile()	**sFnd::ValueBase**	inline
Node()	**sFnd::IObjWithNode**	inline
operator double()	**sFnd::ValueUnsigned**	
operator uint32_t()	**sFnd::ValueUnsigned**	
operator=(uint32_t newValue)	**sFnd::ValueUnsigned**	
Refresh()	**sFnd::ValueUnsigned**	virtual
Supported()	**sFnd::IObj**	virtual
Value(uint32_t newValue, bool makeNonVolatile=false)	**sFnd::ValueUnsigned**	
Value(bool getNonVolatile=false)	**sFnd::ValueUnsigned**	

sFoundation 2.0

_alertReg Member List

This is the complete list of members for **_alertReg**, including all inherited members.

bits	**_alertReg**	
cpm	**_alertReg**	
isClear()	**_alertReg**	inline
isInAlert()	**_alertReg**	inline
StateStr(char *buffer, size_t size)	**_alertReg**	inline

sFoundation 2.0

pubCoreRegs.h

Go to the documentation of this file.

```
//*****************************************************************************
// DESCRIPTION:
//
// NAME
//      pubCoreRegs.h
//
// CREATION DATE:
//      06/19/2008 13:46:39
//
// COPYRIGHT NOTICE:
//      (C)Copyright 2008-2016 Teknic, Inc.  All rights reserved.
//
//      This copyright notice must be reproduced in any copy, modification,
//      or portion thereof merged into another program. A copy of the
//      copyright notice must be included in the object library of a user
//      program.
//
*
//*****************************************************************************
#ifndef __PUBCOREREGS__H__
#define __PUBCOREREGS__H__


//*****************************************************************************
```

```
36 // NAME
 *
37 //  pubCoreRegs.h headers
38 //
39     #include "tekTypes.h"
40     #include "pubRegFldCommon.h"
41     #include "pubIscRegs.h"
42 #if defined(IMPL_NODE_CP)
43     #include "pubClearPathRegs.h"
44 #endif
45 #if !defined(__TI_COMPILER_VERSION__) ||
 defined(_CPM_SC) || defined(_CLEARPATH)
46     #include "pubCpmRegs.h"
47 #endif
48 //
 *
49 //*************************************************
 ****************************
50
51
52 //*************************************************
 ****************************
53 // NAME
 *
54 //  pubCoreRegs.h constants
55 //
57 #define ATTN_MASK_OCTETS 4
58
59 //
 *
60 //*************************************************
 ****************************
61
62
63 //*************************************************
 ****************************
64 // Insure structs are byte packed on target
65 #ifdef _MSC_VER
66 #pragma pack(push, 1)
67 #endif
68 //*************************************************
 ****************************
```

```
 69
 70
 71
 72 //*************************************************
    ****************************
 73 // NAME
    *
 74 //  optionReg union
 75 //
 76 // DESCRIPTION
 83 //
 84 typedef union PACK_BYTES _optionReg {
 88     enum _hwPlatforms {
 89         HW_CLEARPATH,
 90         HW_AC_GREEN
 91     };
 95     Uint32 bits;
 96     // - - - - - - - - - - - - - - - - - - - -
 97     // COMMON PRODUCT OPTION DEFINITIONS
 98     // - - - - - - - - - - - - - - - - - - - -
102     struct _optionsCommonFlds {
103         /*                    -----------  -------
    ---------------------------- *
104          *                        bit #
    description                         *
105          *                    -----------  -------
    ---------------------------- */
107         int Setups          :4; // 0-3:   Factory
    setup modes
109         unsigned Product    :2; // 4-5:   Product
    options
111         unsigned Advanced   :1; // 6 :
    Advanced features available
113         unsigned VectorLock :1; // 7 :    Lock
    Vector Mode
116         unsigned OptionBrd  :2; // 8-9:
    Installed option board configuration
117         unsigned IgnoreIndx :1; // 10:    Prevent
    Ref Offset on index
119         int                 :1; // 11 :
122         unsigned HwPlatform :4; // 12-15:
    Hardware platform
```

```
124         int                     :9; // 16-24
127         unsigned Debug          :3; // 25-27: Debug flag
128     };
129     struct _drvCommonFlds {
130         /*                       -----------  -------
    ---------------------------- *
131          *                          bit #
    description                         *
132          *                       -----------  -------
    ---------------------------- */
133         int VoltMode            : 1; // 0:  Volt mode
    setup
134         int RuntimeCal          : 1; // 1:  1=Enable
    run time calibration
135         int ClockCal            : 1; // 2:  1=clock cali
136         int DirCwPositive       : 1; // 3:  1=CW
    direction is positive motion
137
138         unsigned Product        : 3; // 4:  3-bit
    product number
139                                      // 5:  ""
140                                      // 6:  ""
147         unsigned VectorLock : 1; // 7   Lock
    Vector Mode
148
156         unsigned IndexTolerance : 2;    // 8-9
164         unsigned IgnoreIndx  : 1;//  10 Prevent
    Ref Offset on index
165         unsigned NoVectUpdate: 1;//  11 do not
    apply the angle from the enhanced sensorless algorithm
167         unsigned HwPlatform :4; // 12-15:
    Hardware platform (enum _hwPlatforms)
168     };
172     struct _optionsCommonFlds Common;
176     struct _drvCommonFlds DrvCommon;
177     // - - - - - - - - - - - - - - - - - - - -
178     // DRIVE PRODUCT OPTION DEFINITIONS
179     // - - - - - - - - - - - - - - - - - - - -
182     iscOptionsFlds DrvFld;
183 #if defined(IMPL_NODE_CP)
184     // - - - - - - - - - - - - - - - - - - - -
185     // CLEARPATH PRODUCT OPTION DEFINITIONS
```

```
186     // - - - - - - - - - - - - - - - - - - - -
188     clearPathOptionsFlds ClrPathFld;
189 #endif
190 #if !defined(__TI_COMPILER_VERSION__) || defined(_CP
192     cpmOptionsFlds cpm;
194 #endif
195     #ifdef __cplusplus
196     _optionReg() {
199         bits = 0;
200     }
201         #ifdef __TI_COMPILER_VERSION__
202         // Returns TRUE if the option register
    has us in debug mode
203         tBOOL InDebugMode() {
204             return((Common.Debug > 0) &&
    (Common.Debug < 7));
205         }
206         #endif
207     #endif
208 } optionReg;
211 //
    *
212 //*************************************************
    ******************************
213
214
215 //*************************************************
    ******************************
216 // NAME
    *
217 //  alertReg union
218 //
219 // DESCRIPTION
220 // Alert Register Size in octets: 96 bits = 3 32-
    bit entities = 12 octets
221 #define ALERTREG_SIZEOCTETS 12
222
255 typedef union MN_EXPORT PACK_BYTES _alertReg {
257     Uint32 bits[3];
266     commonAlertFlds Common;
268 #if !defined(__TI_COMPILER_VERSION__) || defined(_CP
269
```

```
273         cpmAlertFlds cpm;
274     #endif
275         // - - -
```

```
            bool InAlert();
        #endif
        // Constructor, create with bits cleared
        _alertReg() {                   // Clear bits on construction
            STATIC_ASSERT(ALERTREG_SIZEOCTETS==12);
            clear();
        }
    #endif

} alertReg;

// These bit numbers must match the locations in the alertReg union
// They are used by coding that requires either a mask or a bit number.
/*
    Bit Number index for the Alert Register

    These number can be used to construct masks.
*/
typedef enum _drvAlertBits {
    DRV_ALERT_FW_BAD_ACQ_STATE_BIT              = 32,
    DRV_ALERT_FW_MOVEGEN_MISC_BIT               = 33,
    DRV_ALERT_FW_MOVEGEN_OFF_TARGET_BIT         = 34,
    DRV_ALERT_FW_MOVEGEN_RANGE_BIT              = 35,
    DRV_ALERT_FW_BAD_RAS_REQUEST_BIT            = 36,
    DRV_ALERT_FW_MOVE_BUF_UNDERRUN_BIT          = 37,
    DRV_ALERT_FW_BAD_RAS_VELOCITY_BIT           = 38,
    DRV_ALERT_FW_MOVE_SPEC_ALTERED_BIT          = 39,
    DRV_ALERT_HW_PHASE_SENSOR_BIT               = 40,   // HW Shdn
    DRV_ALERT_FW_MOVEGEN_LIMSW_BIT              = 41,
```

```
    DRV_ALERT_FW_MOVEGEN_SOFTLIM_BIT                =  42,
    DRV_ALERT_FW_MOVE_BUF_OVERRUN_LOCKDN_BIT        =  43,
    DRV_ALERT_IO_5V_SUPPLY_OVERLOAD_BIT             =  44,  // mtn Shdn
    DRV_ALERT_IO_BRAKE_OVERLOAD_BIT                 =  45,
    DRV_ALERT_IO_GPO0_OVERLOAD_BIT                  =  46,
    DRV_ALERT_IO_GPO1_OVERLOAD_BIT                  =  47,
    DRV_ALERT_IO_GPO_DETECTOR_BIT                   =  48,
    DRV_ALERT_IO_GPO_OVERLOAD_BIT                   =  49,
    DRV_ALERT_IO_INVALID_BIT                        =  50,
    DRV_ALERT_IO_INVALID_LOCKDN_BIT                 =  51,
    DRV_ALERT_AC_POWER_LOST_BIT                     =  52,
    DRV_ALERT_IEX_LINK_GLITCH_BIT                   =  53,
    DRV_ALERT_IEX_LINK_MULTI_GLITCH_BIT             =  54,
    DRV_ALERT_MTR_UNDERTEMP_BIT                     =  55,
    DRV_ALERT_MTR_COMM_BAD_SEQ_BIT                  =  56,  // mtn Shdn
    DRV_ALERT_MTR_COMM_BAD_STATE_BIT                =  57,  // mtn Shdn
    DRV_ALERT_MTR_VECTOR_ERR_BIT                    =  58,  // mtn Shdn
    DRV_ALERT_MTR_VECTOR_CRIT_ERR_BIT               =  59,  // mtn Shdn
    DRV_ALERT_MTR_ENC_GLITCH_BIT                    =  60,  // mtn Shdn
    DRV_ALERT_MTR_ENC_OVERSPEED_BIT                 =  61,  // mtn Shdn
    DRV_ALERT_LD_ENC_GLITCH_BIT                     =  62,  // mtn Shdn
```

```
    DRV_ALERT_LD_ENC_OVERSPEED_BIT                =
63,   // mtn Shdn
    DRV_ALERT_MTR_PHASE_OVERLOAD_BIT              =
64,   // mtn Shdn
    DRV_ALERT_MTR_THERMAL_OVERLOAD_BIT            =
65,   // mtn Shdn
    DRV_ALERT_MTR_BAD_SETUP_PARAMS_BIT            =
66,   // sw Shdn
    DRV_ALERT_SVO_HARDSTOP_RELEASE_BIT            =
67,   // mtn Shdn
    DRV_ALERT_SVO_TRACKING_BIT                    =
68,   // mtn Shdn
    DRV_ALERT_SVO_TRACKING_CRIT_BIT               =
69,   // mtn Shdn
    DRV_ALERT_SVO_RMS_OVERLOAD_BIT                =
70,   // mtn Shdn
    DRV_ALERT_SVO_RMS_CRIT_OVERLOAD_BIT           =
71,   // mtn Shdn
    DRV_ALERT_MTR_COUPLING_ERROR_BIT              =
72,   // mtn Shdn
    DRV_ALERT_SVO_STEP_NOISE_BIT                  =
73,   // mtn Shdn
    DRV_ALERT_SVO_VOLT_SAT_BIT                    =
74,   // mtn Shdn
    DRV_ALERT_SVO_TRQ_SAT_BIT                     =
75,   // mtn Shdn
    DRV_ALERT_NOCOMM_DIST_ERROR_BIT               =
76,   // mtn Shdn
    DRV_ALERT_NOCOMM_DIRECTION_ERROR_BIT          =
77,   // mtn Shdn
    DRV_ALERT_NOCOMM_FAIL_BIT                     =
78,
    DRV_ALERT_MTR_ENC_INDEX_MISSING_WARN_BIT      =
79,
    DRV_ALERT_TEMP_AMBIENT_BIT                    =
80,   // mtn Shdn (drive hot)
    DRV_ALERT_TEMP_HEATSINK_BIT                   =
81,   // mtn Shdn (stator hot)
    DRV_ALERT_BUS_OVERCURRENT_BIT                 =
82,   // mtn Shdn
    DRV_ALERT_BUS_OVERVOLTAGE_BIT                 =
83,   // mtn Shdn
```

```
    DRV_ALERT_BUS_VOLTAGE_LOW_BIT                =  84,   // mtn Shdn
    DRV_ALERT_BUS_RMS_OVERLOAD_BIT               =  85,   // mtn Shdn
    DRV_ALERT_FAN_SPEED_LOW_BIT                  =  86,   // mtn Shdn
    DRV_ALERT_MTR_ENC_INDEX_MISSING_BIT          =  87,
    DRV_ALERT_NOCOMM_HW_FAULT_BIT                =  88,
    DRV_ALERT_NOCOMM_BUS_LOW_BIT                 =  89,
    DRV_ALERT_NOCOMM_SW_FAULT_BIT                =  90,
    DRV_ALERT_FW_MOVEGEN_LIMSW_LOCKDN_BIT        =  91,
    DRV_ALERT_FW_MOVEGEN_SOFTLIM_LOCKDN_BIT      =  92,
    DRV_ALERT_MTR_ENC_INDEX_MISPLACED_BIT        =  93,
} drvAlertBits;

// These bit numbers must match the locations in the alertReg union
// They are used by coding that requires either a mask or a bit number.
/*
    Bit Number index for the Alert Register

    These number can be used to construct masks.
*/
typedef enum _cpmAlertBits {
    CPM_ALERT_FW_BAD_RAS_REQUEST_BIT             = DRV_ALERT_FW_BAD_RAS_REQUEST_BIT,     // 36
    CPM_ALERT_FW_MOVE_BUF_UNDERRUN_BIT           = DRV_ALERT_FW_MOVE_BUF_UNDERRUN_BIT,   // 37
    CPM_ALERT_FW_BAD_RAS_VELOCITY_BIT            = DRV_ALERT_FW_BAD_RAS_VELOCITY_BIT,    // 38
    CPM_ALERT_FW_MOVE_SPEC_ALTERED_BIT           = DRV_ALERT_FW_MOVE_SPEC_ALTERED_BIT,   // 39
    CPM_ALERT_HW_PHASE_SENSOR_BIT                = DRV_ALERT_HW_PHASE_SENSOR_BIT,        // 40 HW Shdn
```

```
    CPM_ALERT_FW_MOVEGEN_LIMSW_BIT                  = DRV_ALERT_FW_MOVEGEN_LIMSW_BIT,       // 41
    CPM_ALERT_FW_MOVEGEN_SOFTLIM_BIT                = DRV_ALERT_FW_MOVEGEN_SOFTLIM_BIT,     // 42
    CPM_ALERT_FW_MOVE_BUF_OVERRUN_LOCKDN_BIT        = DRV_ALERT_FW_MOVE_BUF_OVERRUN_LOCKDN_BIT,// 43
    CPM_ALERT_IO_INVALID_BIT                        = DRV_ALERT_IO_INVALID_BIT,             // 50
    CPM_ALERT_IO_INVALID_LOCKDN_BIT                 = DRV_ALERT_IO_INVALID_LOCKDN_BIT,      // 51
    CPM_ALERT_AC_POWER_LOST_BIT                     = DRV_ALERT_AC_POWER_LOST_BIT,          // 52
    CPM_ALERT_AC_PHASE_WIRING_ERROR_BIT             = 53,                                   // 53
    CPM_ALERT_MTR_UNDERTEMP_BIT                     = DRV_ALERT_MTR_UNDERTEMP_BIT,          // 55
    CPM_ALERT_MTR_VECTOR_ERR_BIT                    = DRV_ALERT_MTR_VECTOR_ERR_BIT,         // 58 mtn Shdn
    CPM_ALERT_MTR_ENC_GLITCH_BIT                    = DRV_ALERT_MTR_ENC_GLITCH_BIT,         // 60 mtn Shdn
    CPM_ALERT_MTR_ENC_OVERSPEED_BIT                 = DRV_ALERT_MTR_ENC_OVERSPEED_BIT,      // 61 mtn Shdn
    CPM_ALERT_MTR_PHASE_OVERLOAD_BIT                = DRV_ALERT_MTR_PHASE_OVERLOAD_BIT,     // 64 mtn Shdn
    CPM_ALERT_MTR_BAD_SETUP_PARAMS_BIT              = DRV_ALERT_MTR_BAD_SETUP_PARAMS_BIT,   // 66 sw Shdn
    CPM_ALERT_SVO_HARDSTOP_RELEASE_BIT              = DRV_ALERT_SVO_HARDSTOP_RELEASE_BIT,   // 67 mtn Shdn
    CPM_ALERT_SVO_TRACKING_CRIT_BIT                 = DRV_ALERT_SVO_TRACKING_CRIT_BIT,      // 69 mtn Shdn
    CPM_ALERT_SVO_RMS_OVERLOAD_BIT                  = DRV_ALERT_SVO_RMS_OVERLOAD_BIT,       // 70 mtn Shdn
    CPM_ALERT_SVO_RMS_CRIT_OVERLOAD_BIT             = DRV_ALERT_SVO_RMS_CRIT_OVERLOAD_BIT,  // 71 mtn Shdn
    CPM_ALERT_SVO_STEP_NOISE_BIT                    = DRV_ALERT_SVO_STEP_NOISE_BIT,         // 73 mtn Shdn
    CPM_ALERT_SVO_VOLT_SAT_BIT                      = DRV_ALERT_SVO_VOLT_SAT_BIT,           // 74 mtn Shdn
    CPM_ALERT_SVO_TRQ_SAT_BIT                       = DRV_ALERT_SVO_TRQ_SAT_BIT,            // 75 mtn Shdn
    CPM_ALERT_NOCOMM_DIST_ERROR_BIT                 = DRV_ALERT_NOCOMM_DIST_ERROR_BIT,      // 76 mtn Shdn
```

```
    CPM_ALERT_NOCOMM_DIRECTION_ERROR_BIT        = DRV_ALERT_NOCOMM_DIRECTION_ERROR_BIT, // 77 mtn Shdn
    CPM_ALERT_NOCOMM_FAIL_BIT                   = DRV_ALERT_NOCOMM_FAIL_BIT,            // 78
    CPM_ALERT_MTR_ENC_INDEX_MISSING_WARN_BIT    = DRV_ALERT_MTR_ENC_INDEX_MISSING_WARN_BIT, // 79
    CPM_ALERT_TEMP_DRIVE_BIT                    = DRV_ALERT_TEMP_AMBIENT_BIT,           // 80 mtn Shdn
    CPM_ALERT_TEMP_STATOR_BIT                   = DRV_ALERT_TEMP_HEATSINK_BIT,          // 81 mtn Shdn
    CPM_ALERT_BUS_OVERCURRENT_BIT               = DRV_ALERT_BUS_OVERCURRENT_BIT,        // 82 mtn Shdn
    CPM_ALERT_BUS_OVERVOLTAGE_BIT               = DRV_ALERT_BUS_OVERVOLTAGE_BIT,        // 83 mtn Shdn
    CPM_ALERT_BUS_VOLTAGE_LOW_BIT               = DRV_ALERT_BUS_VOLTAGE_LOW_BIT,        // 84 mtn Shdn
    CPM_ALERT_BUS_RMS_OVERLOAD_BIT              = DRV_ALERT_BUS_RMS_OVERLOAD_BIT,       // 85 mtn Shdn
    CPM_ALERT_BUS_PWR_MISMATCH_BIT              = 86,                                   // 86 Power issue (v1.6)
    CPM_ALERT_MTR_ENC_INDEX_MISSING_BIT         = DRV_ALERT_MTR_ENC_INDEX_MISSING_BIT,  // 87
    CPM_ALERT_BUS_VOLTAGE_UNDER_OPERATING_V_BIT = 88,                                   // 88 Power issue (v1.6)
    CPM_ALERT_FW_MOVEGEN_LIMSW_LOCKDN_BIT       = DRV_ALERT_FW_MOVEGEN_LIMSW_LOCKDN_BIT,// 91
    CPM_ALERT_FW_MOVEGEN_SOFTLIM_LOCKDN_BIT     = DRV_ALERT_FW_MOVEGEN_SOFTLIM_LOCKDN_BIT, // 92
    CPM_ALERT_MTR_ENC_INDEX_MISPLACED_BIT       = DRV_ALERT_MTR_ENC_INDEX_MISPLACED_BIT,// 93
    CPM_ALERT_STEPS_DURING_POSN_RECOVERY_BIT    = 94,                                   // 94
} cpmAlertBits;

//*
//*****************************************************************************
//*****************************************************************************
```

```
500 //  NAME
    *
501 //      alertLogEntry structure and definitions
502 //
503 // CREATION DATE:
504 //      02/11/2010 17:40:47
505 //
506 //  DESCRIPTION:
523 typedef struct _alertLogEntry {
524     Uint32 Timestamp;
525     union _alertReasons {
526         commonAlertBits common;
527         drvAlertBits isc;
528         cpmAlertBits cpm;
529         Uint32 bits;
530         // Common translators
531         void operator=(const commonAlertBits
    updVal) { common = updVal; }
532         void operator=(const drvAlertBits updVal)
    { isc = updVal; }
533         void operator=(const cpmAlertBits updVal)
    { cpm = updVal; }
534         bool operator==(const commonAlertBits
    lVal) { return common == lVal; }
535         bool operator!=(const commonAlertBits
    lVal) { return common != lVal; }
536         // Common constructors
537         _alertReasons(const commonAlertBits
    initVal) { common=initVal; }
538         _alertReasons(const drvAlertBits initVal)
    { isc=initVal; }
539         _alertReasons(const cpmAlertBits initVal)
    { cpm=initVal; }
540         _alertReasons() { bits = 0; }
541     } Reason;
542 } alertLogEntry;
543 // - - - - - - - - - - - - - - - - - -
544 // Alert command arguments
545 // - - - - - - - - - - - - - - - - - -
546 // Clear out all entries
547 #define ALERT_ARG_CLEAR         ALERT_LOG_FREE_CODE
548 // Update/create the wall clock time epoch value
```

```
549 #define ALERT_ARG_GET_OLDEST    0
550 //
    *
551 //*************************************************
    ****************************
555 //=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
556 // STATUS REGISTER UNIONS AS "REGISTERS" WITH
    BIT-WISE AND FLD ACCESS
557 //=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
558
560 //*************************************************
    ****************************
561 // NAME
    *
562 //  lowAttnReg union
563 //
564 // DESCRIPTION
573 typedef union PACK_BYTES _lowAttnReg {
577     Uint16 bits;
581     lowAttnCommonFlds Common;
585     iscLowAttnFlds DrvFld;
586 } lowAttnReg;
587 //
    *
588 //*************************************************
    ****************************
593 //*************************************************
    ****************************
594 // NAME
    *
595 //  nonAttnReg union
596 //
597 // DESCRIPTION
598
608 typedef union PACK_BYTES _nonAttnReg {
610     Uint16 bits;
614     nonAttnCommonFlds Common;
618     iscNonAttnFlds DrvFld;
619     // 12/21/2015 Delete me as this is handled by
    the new status register def
```

```
620     //ClearPathNonAttnReg ClrPathFld;
621 } nonAttnReg;
622 //
    *
624 //*************************************************
    ****************************
629 //*************************************************
    ****************************
630 // NAME
    *
631 //  plaInReg union
632 //
633 // DESCRIPTION
644 typedef union PACK_BYTES _plaInReg {
646     Uint16 bits;
648     plaInCommonFlds Common;
653     iscPlaInFlds DrvFld;
654 } plaInReg;
655 //
    *
656 //*************************************************
    ****************************
662 //*************************************************
    ****************************
663 //  NAME
    *
664 //      plaOutReg union
665 //
666 // CREATION DATE:
667 //      02/17/2010 16:08:32
668 //
669 //  DESCRIPTION:
703 typedef union PACK_BYTES _plaOutReg {
705     Uint16 bits;
706     // Common outputs for all nodes
707     struct _plaOutCommonFlds {
708         int na0_4       : 5;
715         int NodeStop    : 1;    // Issue NodeStop
716     } Common;
717     iscOutFlds DrvFld;
719 #if defined(IMPL_NODE_CP)
720     clearPathOutFlds ClearPathFld;
```

```
722 #endif
723 #if !defined(__TI_COMPILER_VERSION__) || defined(_CP
724     cpmOutFlds cpm;
726 #endif
727     #ifdef __cplusplus
728     void clear() {
730         bits = 0;
731     };
732     #endif
733 } plaOutReg;
734 //
    *
735 //**********************************************
    ****************************
740 //**********************************************
    ****************************
741 //  NAME
    *
742 //      plaFuses structure
743 //
744 // CREATION DATE:
745 //      10/12/2010 16:13
746 //
747 //  DESCRIPTION:
769 typedef union _plaFuses {
771     Uint16 bits;
774     plaInReg andFuse;
777     struct  _orFuses {
778         int and00        : 1;
779         int and01        : 1;
780         int and02        : 1;
781         int and03        : 1;
782         int and04        : 1;
783         int and05        : 1;
784         int and06        : 1;
785         int and07        : 1;
786         int and08        : 1;
787         int and09        : 1;
788         int and10        : 1;
789         int and11        : 1;
790         int and12        : 1;
791         int and13        : 1;
```

```
792         int and14        : 1;
793         int and15        : 1;
794     } orFuse;
795     plaOutReg outFuse;
797     #ifdef __cplusplus
798     _plaFuses() {
800         bits = 0;
801     };
803     _plaFuses(Uint16 initialBits) {
804         bits = initialBits;
805     };
806     #endif
807 } plaFuses;
808 //
    *
809 //*********************************************
    ****************************
814 //*********************************************
    ****************************
815 //  NAME
    *
816 //      plaFuseBuffer structure
817 //
818 // CREATION DATE:
819 //      10/12/2010 16:13
820 //
821 //  DESCRIPTION:
878 typedef struct _plaFuseBuffer {
880     typedef enum _PLA_AREAS {
881         PLA_AND,                /* 0 */
882         PLA_NAND,               /* 1 */
883         PLA_OR,                 /* 2 */
884         PLA_OPTION,             /* 3 */
885         PLA_ARCH                /* 4 */
886     } PLA_AREAS;
888     PLA_AREAS FuseArea;
890     typedef union _plaIndex {
891         Uint16   bits;
892         struct {
893             BFld Index      : 7;
894             BFld EEalso     : 1;
895         } fld;
```

```
896         _plaIndex() {
898             bits = 0;
899         };
900     } plaIndex;                         // PLA
    Index type.
902     plaIndex FuseIndex;
904     plaFuses Fuses;
906     typedef enum _PLA_OPT_INDICES {
913         OPTION_INVERTER                 /* 0 */
914     } PLA_OPT_INDICES;
916     typedef enum _ARCH_INDICES {
917         AND_TERMS,                      /* 0 */
918         OR_TERMS,                       /* 1 */
919         OPTION_REGS                     /* 2 */
920     } ARCH_INDICES;
921     #ifdef __cplusplus
922     _plaFuseBuffer() {
924         FuseArea = PLA_AND;
925         FuseIndex.bits = 0;
926         Fuses.bits = 0;
927     };
929     _plaFuseBuffer(PLA_AREAS area, Uint16 index,
    Uint16 fuses) {
930         FuseArea = area;
931         FuseIndex.bits = index;
932         Fuses.bits = fuses;
933     };
935     void Setup(PLA_AREAS area, Uint16 index,
    Uint16 fuses) {
936         FuseArea = area;
937         FuseIndex.bits = index;
938         Fuses.bits = fuses;
939     };
941     void Setup(PLA_AREAS area, Uint16 index) {
942         FuseArea = area;
943         FuseIndex.bits = index;
944         Fuses.bits = 0;
945     };
946     #endif
947 } plaFuseBuffer;
948 //
    *
```

```
949 //*************************************************
    ******************************
953 //*************************************************
    ******************************
954 //  NAME
    *
955 //      xpsFuseBuffer structure
956 //
957 // CREATION DATE:
958 //      09/29/2015 16:13
959 //
960 //  DESCRIPTION:
1016 typedef struct _xpsFuseBuffer {
1018     typedef enum _XPS_AREAS {
1019         XPS_IN_OPTION,          /* 0 */
1020         XPS_OR,                 /* 1 */
1021         XPS_OPTION,             /* 2 */
1022         XPS_ARCH                /* 3 */
1023     } XPS_AREAS;
1025     XPS_AREAS FuseArea;
1027     typedef union _xpsIndex {
1028         Uint16   bits;
1029         struct {
1030             BFld Index      : 7;
1031             BFld EEalso     : 1;
1032         } fld;
1033         _xpsIndex() {
1035             bits = 0;
1036         };
1037     } xpsIndex;                         // XPS
    Index type.
1039     xpsIndex FuseIndex;
1041     plaFuses Fuses;
1043     typedef enum _XPS_OPT_INDICES {
1050         OPTION_INVERTER                 /* 0 */
1051     } XPS_OPT_INDICES;
1053     typedef enum _ARCH_INDICES {
1054         AND_TERMS,                      /* 0 */
1055         OR_TERMS,                       /* 1 */
1056         OPTION_REGS                     /* 2 */
1057     } ARCH_INDICES;
1058     #ifdef __cplusplus
```

```
1059     _xpsFuseBuffer() {
1061         FuseArea = XPS_OR;
1062         FuseIndex.bits = 0;
1063         Fuses.bits = 0;
1064     };
1066     _xpsFuseBuffer(XPS_AREAS area, Uint16 index,
    Uint16 fuses) {
1067         FuseArea = area;
1068         FuseIndex.bits = index;
1069         Fuses.bits = fuses;
1070     };
1072     void Setup(XPS_AREAS area, Uint16 index,
    Uint16 fuses) {
1073         FuseArea = area;
1074         FuseIndex.bits = index;
1075         Fuses.bits = fuses;
1076     };
1078     void Setup(XPS_AREAS area, Uint16 index) {
1079         FuseArea = area;
1080         FuseIndex.bits = index;
1081         Fuses.bits = 0;
1082     };
1083     #endif
1084 } xpsFuseBuffer;
1085 //
    *
1086 //**********************************************
    ****************************
1092 //**********************************************
    ****************************
1093 //  NAME
    *
1094 //      mnOutReg structure
1095 //
1096 // CREATION DATE:
1097 //      12/16/2015
1098 //
1099 //  DESCRIPTION:
1109 typedef union MN_EXPORT PACK_BYTES _mnOutReg
1110 {
1112     Uint16 bits;
```

```
1113     // Meridian ISC outputs (TODO - Restore for
    ISC support)
1114     //plaOutReg isc;
1115 #if !defined(__TI_COMPILER_VERSION__) || defined(_CP
1116     cpmOutFlds cpm;
1118 #endif
1119     //
1120     #ifdef __cplusplus
1121     void clear() {
1123         bits = 0;
1124     }
1128     bool isClear() {
1129         return bits == 0;
1130     }
1131
1133     _mnOutReg()
1134     {
1135         clear();
1136     }
1137
1138     #endif
1139 } mnOutReg;
1141 //
    *
1142 //*************************************************
    ******************************
1143
1144
1145 //*************************************************
    ******************************
1146 // NAME
    *
1147 //  mnStatusReg union
1148 //
1149 // DESCRIPTION
1166 typedef union MN_EXPORT PACK_BYTES _mnStatusReg {
1168     Uint16 bits[3];
1170     Uint32 attnBits;
1175     commonStatusFlds Common;
1176
1181     iscStatusRegFlds isc;
1186     iscStatusRegOldFlds Fld;
```

```
1191        cpmStatusRegFlds cpm;
1193        #ifdef __cplusplus
1194        typedef enum _bitsTypes {
1196            LOW_ATTN,
1197            IN_REG,
1198            NON_ATTN
1199        } bitsTypes;
1201        enum {N_BITS = 3};
1202
1206        void clear() {
1207            bits[LOW_ATTN] = bits[IN_REG] = bits[NON_ATTN] = 0;
1208        }
1212        bool isClear() {
1213            return (bits[LOW_ATTN] | bits[IN_REG] | bits[NON_ATTN]) == 0;
1214        }
1217    #ifndef __TI_COMPILER_VERSION__
1218
1231        char * StateStr(char *buffer, size_t size) {
1232            return (cpm.StateStr(buffer, size));
1233        }
1234    #endif
1235
1236        // Construct a cleared register
1237        _mnStatusReg() {
1238            clear();
1239        }
1240        #endif
1241
1242    } mnStatusReg;
1244    typedef mnStatusReg attnReg;            // 1/5/16 Made equivalent
1246    // True size of the mnStatusReg
1247    #define MN_STATUS_REG_OCTETS    6
1248
1249    //*
1250    //*****************************************************************************
1251
1252
```

```
1253 //*************************************************
     ******************************
1254 // NAME
     *
1255 //  mnAuditData structure
1256 //
1257 // DESCRIPTION
1269 typedef struct PACK_BYTES _mnAuditData {
1270     double          LowPassRMS;
1271     double          LowPassMaxPos;
1272     double          LowPassMaxNeg;
1273     double          HighPassRMS;
1274     double          DurationMS;
1275     nodelong        MaxTrackingPos;
1276     nodelong        MaxTrackingNeg;
1277 } mnAuditData;
1280 typedef struct _mnAuditData iscDataCollect;
1283 //
     *
1284 //*************************************************
     ******************************
1285
1286 //*************************************************
     ******************************
1287 // NAME
     *
1288 //  mnAppConfigReg union
1289 //
1290 // DESCRIPTION
     /** \cond ExpertDoc **/
1297 typedef union PACK_BYTES _mnAppConfigReg {
1299     nodeulong       bits;
1300 #if !defined(__TI_COMPILER_VERSION__) || defined(_CP
1301     cpmAppConfigFlds cpm;
1303 #endif
1304     iscAppConfigFlds isc;
1306 } mnAppConfigReg;
1308 //
     *
1309 //*************************************************
     ******************************
1310
```

```
//*****************************************************************************
// NAME
//  mnHwConfigReg union
//
// DESCRIPTION
typedef union PACK_BYTES _mnHwConfigReg {
    nodeulong       bits;
#if !defined(__TI_COMPILER_VERSION__) || defined(_CP
    cpmHwConfigFlds cpm;
#endif
    iscHwConfigFlds isc;
} mnHwConfigReg;
//
//*****************************************************************************



//*****************************************************************************
// Restore packing rules
#ifdef _MSC_VER
#pragma pack(pop)
#endif
//*****************************************************************************

#endif
//=============================================================================
//  END OF FILE pubCoreRegs.h
//=============================================================================

```

_cpmAlertFlds Member List

This is the complete list of members for _cpmAlertFlds, including all inherited members.

ACPhaseLost	_cpmAlertFlds
ACPowerLost	_cpmAlertFlds
BusOverCurrent	_cpmAlertFlds
BusOverVoltage	_cpmAlertFlds
BusRMSOverload	_cpmAlertFlds
BusVoltageLow	_cpmAlertFlds
Common	_cpmAlertFlds
DriveTempHigh	_cpmAlertFlds
HardStopBrokeFree	_cpmAlertFlds
JrkLimVelRequestBad	_cpmAlertFlds
LimSwitchActivated	_cpmAlertFlds
MtrBadSetupParams	_cpmAlertFlds
MtrEncGlitch	_cpmAlertFlds
MtrEncIndexMisplaced	_cpmAlertFlds
MtrPhaseOverload	_cpmAlertFlds
MtrTempLow	_cpmAlertFlds
MtrVectorBad	_cpmAlertFlds
NoCommFailed	_cpmAlertFlds
NoCommSweepFailed	_cpmAlertFlds
NoCommSweepReversed	_cpmAlertFlds

PhaseSensorFailed	**_cpmAlertFlds**
RMSOverloadShutdown	**_cpmAlertFlds**
SoftLimitExceeded	**_cpmAlertFlds**
StateStr(char *buffer, size_t size) const	**_cpmAlertFlds**
StatorTempHigh	**_cpmAlertFlds**
StepsDuringPosnRecovery	**_cpmAlertFlds**
TrackingShutdown	**_cpmAlertFlds**

pubCpmRegs.h

Go to the documentation of this file.

```
//*****************************************************************************
// DESCRIPTION:
// PRINCIPLE AUTHORS:
//      DWS
//
// CREATION DATE:
//      2015-12-23 16:26:15
//
// COPYRIGHT NOTICE:
//      (C)Copyright 2015-2017  Teknic, Inc.  All rights reserved.
//
//      This copyright notice must be reproduced in any copy, modification,
//      or portion thereof merged into another program. A copy of the
//      copyright notice must be included in the object library of a user
//      program.
//
//*****************************************************************************
#ifndef __PUBCPM_REGS_H__
#define __PUBCPM_REGS_H__

//*****************************************************************************
// NAME
//  pubCPM_API.h headers
//
    #include "tekTypes.h"
    #include "pubMotion.h"
    #include "pubRegFldCommon.h"
//
//*****************************************************************************

//*****************************************************************************
// NAME
//
```

```
//  pubCpmRegs.h constants
//
#define ALERTREG_SIZEOCTETS     12
#define SC_USER_DESCR_SPAN      5       // Number parameters in
#define SC_USER_DESCR_CHUNK     25      // # chars per parameter
// Max description size
#define SC_USER_DESCR_SZ        (CP_USER_DESCR_SPAN*CP_USER_DESCR_CHU

// - - -
```

```
102     CPM_P_NETERR_APP_FRAG,                  // 37 25 Application Net
    Fragment counter
103     CPM_P_NETERR_APP_STRAY,                 // 38 26 Application Net
    Stray data counter
104     CPM_P_NETERR_APP_OVERRUN,               // 39 27 Application Net
    Overrun counter
105     CPM_P_NETERR_DIAG_CHKSUM,               // 40 28 Diagnostic Net
    Checksum counter
106     CPM_P_NETERR_DIAG_FRAG,                 // 41 29 Diagnostic Net
    Fragment counter
107     CPM_P_NETERR_DIAG_STRAY,                // 42 2a Diagnostic Net
    Stray data counter
108     CPM_P_NETERR_DIAG_OVERRUN,              // 43 2b Diagnostic Net
    Overrun counter
109
110     CPM_P_ATTN_MASK=45,                     // 45 2d Attention mask regi
111     CPM_P_GP_TIMER,                         // 46 2e General-purpose
    timer period
112     CPM_P_AT_POSN_LOC,                      // 47 2f At Position locatio
113     CPM_P_A_START,                          // 48 30 A after start locat
114     CPM_P_B_END,                            // 49 31 B before end locati
115     CPM_P_USER_OUT_REG,                     // 50 32 User output setting
116
117     CPM_P_OUT_REG = 52,                     // 52 34 Output register
118     CPM_P_DRV_MTR_SPEED_LIM,                // 53 35 Motor Speed
    Limit (quads/msec)
119
120     CPM_P_ATTN_DRVR_MASK=55,                // 55 37 Driver's
    attention register
121     CPM_P_STATUS_EVENT_MASK,                // 56 38 Status Event Mask
122 // Motion Constraint Group
123     CPM_P_APP_RAS_DELAY,                    // 57 39 Stopping delay
    caused by RAS
124     CPM_P_RAS_MAX_VELOCITY=60,              // 60 3c Maximum velocity
    for current RAS
125     CPM_P_VEL_LIM,                          // 61 3d Trapezoidal
    velocity limit
126     CPM_P_ACC_LIM,                          // 62 3e Trapezoidal
    acceleration limit
127     CPM_P_JERK_LIM,                         // 63 3f Jerk limit
128     CPM_P_DEC_LIM,                          // 64 40 Deceleration Limit
129     CPM_P_STOP_ACC_LIM,                     // 65 41 Stop
    acceleration limit
130     CPM_P_HEAD_DISTANCE,                    // 66 42 Head distance
131     CPM_P_TAIL_DISTANCE,                    // 67 43 Tail distance
132     CPM_P_HEADTAIL_VEL,                     // 68 44 Head/Tail velocity
133     CPM_P_MOVE_DWELL,                       // 69 45 Post-move dwell
    time (sec)
134
135     CPM_P_DRV_STOP_QUALIFY_TC=72,           // 72 48 Stopped
    qualification time (ms)
136
137     CPM_P_DRV_STOP_QUALIFY_VEL=74,          // 74 4a Stopped qualify vel
138 // Motion Status Group
```

```
    CPM_P_DRV_TRQ_CMD=80,                // 80 50 Commanded torque
    CPM_P_DRV_TRQ_MEAS,                  // 81 51 Measured torque
    CPM_P_DRV_RMS_LVL,                   // 82 52 RMS level
    CPM_P_POSN_MEAS,                     // 83 53 Measured position
    CPM_P_POSN_CMD,                      // 84 54 Commanded position
    CPM_P_VEL_MEAS,                      // 85 55 Measured velocity
    CPM_P_VEL_CMD,                       // 86 56 Commanded velocity

    CPM_P_DRV_RMS_SLOW_LVL=88,           // 88 58 RMS level (slow act

    CPM_P_POSN_TRK=90,                   // 90 5a Position tracking error
    CPM_P_POSN_TRK_RANGE,                // 91 5b Tracking in-range window
    CPM_P_DRV_MV_DN_TC,                  // 92 5c Move Done: Time Constant (ms)

    CPM_P_VEL_TRK_RANGE = 97,            // 97 61 Vel tracking pct (1.15)

    CPM_P_INA_TC = 99,                   // 99 Input A filter (ms)
    CPM_P_INB_TC = 102,                  // 102 Input B filter (ms)

    // BANK 1 items (Drive/Motor Configuration)
// Factory Settings
    CPM_P_DRV_ADC_MAX=256,               // 256 0  ADC full scale (A)
    CPM_P_DRV_I_MAX,                     // 257 1  Current Full Scale (A)
    CPM_P_DRV_RMS_MAX,                   // 258 2  Factory RMS limit
    CPM_P_DRV_FACT_IR_CAL,               // 259 3  Ir Sensor Calibrat
    CPM_P_DRV_FACT_IS_CAL,               // 260 4  Is Sensor Calibrat
    CPM_P_DRV_FACT_KM_FACTOR,            // 261 5  Km Factor
    CPM_P_DRV_FACT_MTR_MIN_RES,          // 262 6  Mtr Min Res
    CPM_P_DRV_FACT_IB_TC,                // 263 7  IB time constant
    CPM_P_DRV_FACT_IB_TRIP,              // 264 8  IB Trip Point (A)
    CPM_P_DRV_FACT_PHASE_TRIP,           // 265 9  Phase Trip Point (A)
    CPM_P_DRV_FACT_FS_BUSV,              // 266 A Full scale bus volt
    CPM_P_FACT_OVER_V_TRIP=268,          // 268 0c Over-voltage shutdown point
    CPM_P_FACT_VBUS_CAL,                 // 269 0d Bus voltage calibration factor
// Safety Items
    CPM_P_DRV_RMS_LIM,                   // 270 0e RMS shutdown limit (A)
    CPM_P_DRV_RMS_TC,                    // 271 0f RMS time constant (sec) @ 100% trq
    CPM_P_POSN_TRK_ERR_LIM,              // 272 10 Position tracking rrror limit
    CPM_P_DRV_RMS_INIT,                  // 273 11 RMS initial value on reset (%)
    CPM_P_DRV_IB_PEAK,                   // 274 12 IB Level Peak
    CPM_P_DRV_RMS_SLOW_LIM,              // 275 13 RMS shutdown limit (A) (Slow acting)
```

```
    CPM_P_TP_PHASE_BD = 276,              // 276 14 Phase board test point

    CPM_P_POSN_CAP_INB=277,               // 277 15 Captured position (Input B)
// Drive State Items
    CPM_P_DRV_BUS_VOLTS=280,              // 280 18 Real-time Bus Voltage (V)
    CPM_P_DRV_CONFIG_CHANGED,             // 281 19 Configuration chan
    CPM_P_DRIVE_TEMP=291,                 // 291 23 Drive temperature (deg C)
    CPM_P_DRV_IB_LVL=294,                 // 294 26 IB Level

    CPM_P_OVER_TEMP_TRIP=308,             // 308 34 Over-Temp Trip Poi
// Motor Parameters
    CPM_P_DRV_KIP_ADJ,                    // 309 35 Vector Torque: Kip Adjust
    CPM_P_DRV_KIP=318,                    // 318 3e Vector Torque: Kip
    CPM_P_DRV_KII,                        // 319 3f Vector Torque: Kii
    CPM_P_DRV_STEP_DENS,                  // 320 40 Step+Dir Resolution (counts/rev)

    CPM_P_CMD_CNTS_PER_REV=322,           // 322 42 Command encoder density (counts/rev)
    CPM_P_DRV_KR,                         // 323 43 Compensator: Kr
    CPM_P_DRV_KII_ADJ,                    // 324 44 Vector Torque: Kii Adjust
// Tuning parameters
    CPM_P_DRV_TUNE_CONFIG_REG,            // 325 45 Tuning Configuration Register
    CPM_P_DRV_KV,                         // 326 46 Compensator: Kv
    CPM_P_DRV_KP,                         // 327 47 Compensator: Kp
    CPM_P_DRV_KI,                         // 328 48 Compensator: Ki
    CPM_P_DRV_KFV,                        // 329 49 Compensator: Kfv
    CPM_P_DRV_KFA,                        // 330 4a Compensator: Kfa
    CPM_P_DRV_KFJ,                        // 331 4b Compensator: Kfj (jerk ff)

    CPM_P_DRV_KNV=333,                    // 333 4d Compensator: Knv
    CPM_P_DRV_AH_FILT_TC,                 // 334 4e Anti-hunt filter time constant
    CPM_P_DRV_TRQ_BIAS,                   // 335 4f Compensator: Torque Bias (A)
    CPM_P_DRV_FUZZY_APERTURE,             // 336 50 Fuzzy Aperture
    CPM_P_DRV_FUZZY_HYST,                 // 337 51 Fuzzy Hystersis
    CPM_P_AH_HOLDOFF,                     // 338 52 Anti-hunt holdoff
    CPM_P_DRV_KZERO,                      // 339 53 Compensator: Kzero
    CPM_P_DRV_IZERO,                      // 340 54 Compensator: Izero
    CPM_P_DRV_TARGET_WIN,                 // 341 55 Compensator: Target Window
    CPM_P_DRV_STAB_WIN,                   // 342 56 Compensator: Stability Window

    CPM_P_DRV_KP_FACTOR=346,              // 346 5a KpOut Factor
```

```
    CPM_P_DRV_KPZ_FACTOR,                // 347 5b KpZero Factor
    CPM_P_DRV_KI_FACTOR,                 // 348 5c Ki Factor
    CPM_P_DRV_FINE_TUNE,                 // 349 5d Fine Tuning
// Foldback Settings
    CPM_P_DRV_TRQ_LIM ,                  // 350 5e Torque Limit  (A)
    CPM_P_DRV_POS_FLDBK_TRQ,             // 351 5f + Torque foldback limit (A)
    CPM_P_DRV_POS_FLDBK_TRQ_TC,          // 352 60 + Torque foldback time const (ms)
    CPM_P_DRV_NEG_FLDBK_TRQ,             // 353 61 - Torque foldback limit (A)
    CPM_P_DRV_NEG_FLDBK_TRQ_TC,          // 354 62 - Torque foldback time const (ms)

    CPM_P_DRV_HARDSTOP_FLDBK_TRQ=357,    // 357 65 Hardstop foldback limit (A)
    CPM_P_DRV_HARDSTOP_FLDBK_TRQ_TC,     // 358 66 Hardstop foldback time const (ms)
    CPM_P_DRV_HARDSTOP_ENTRY_SPD,        // 359 67 Hard-stop speed qualifier
    CPM_P_DRV_HARDSTOP_ENTRY_TRQ,        // 360 68 Foldback Torque Trip Level (A)
    CPM_P_DRV_HARDSTOP_ENTRY_TC,         // 361 69 Hard Stop: entry delay (ms)

    CPM_P_DRV_HARDSTOP_TRQ_LIM=363,      // 363 6b Alt Trq Lim for Hardstop homing (A)
    CPM_P_DRV_RMS_SLOW_TC,               // 364 6c RMS time constant (sec) @ 100% trq (slow acting)

    CPM_P_DLY_TO_DISABLE=368,            // 368 70 Delay to Disable for brakes (ms)
    CPM_P_FACT_SOFT_START_FREQ,          // 369 71 Soft start PWM carrier frequency (Hz)
// VRS settings
    CPM_P_DRV_VRS_TRQ_FILT=370,          // 370 72 VRS Filtering (ms)
    CPM_P_DRV_VRS_FULL_ON=371,           // 371 73 VRS High Limit (V)
    CPM_P_DRV_VRS_FULL_OFF=362,          // 362 6a VRS Low Limit (V)
    CPM_P_DRV_BUSV_ADJ_RATE=372,         // 372 74 VBus runtime adjust rate (ms)

// Drive Set Flags Register
    CPM_P_DRV_SET_FLAGS=375,             // 375 77 Drive SetFlags Reg
    CPM_P_FACT_UNDER_TEMP_TRIP=379,      // 379 7b Under-Temp Trip (deg C 10.6)
    CPM_P_DRV_ENCODER_TEMP,              // 380 7c Internal temp of encoder (deg C 10.6)
    CPM_P_DRV_STATOR_TEMP,               // 381 7d Internal temp of stator (deg C 10.6)
    CPM_P_FACT_OVER_TEMP_TRIP_STATOR,    // 382 7e Over-Temp Trip Stator (deg. C)
    CPM_P_FACT_OVER_TEMP_WARN_STATOR,    // 383 7f Over-Temp Warn Stator (deg. C)

```

```
255 // Bank 2 Items - Application Specific Settings
256     // Homing Settings
257     CPM_P_APP_HOMING_VEL=512,            // 512 00 Homing Velocity
258     CPM_P_APP_HOMING_OFFSET,             // 513 01 Initial offset
    index after homing
259     CPM_P_APP_HOMING_ACCEL,              // 514 02 Homing acceleratio
260     CPM_P_APP_HOMING_DELAY=516,          // 516 04 Homing delay
    after hardstop
261     CPM_P_EVENT_SHUTDOWN_MASK,           // 517 05 Power event
    shutdown/warning selector (0=warning, 1=shutdown)
262
263     CPM_P_PWR_AC_LOSS_TC,                // 518 06 AC Loss
    shutdown/warning time constant
264     CPM_P_PWR_AC_WIRING_ERROR_TC,        // 519 07 AC Wiring Error
    shutdown/warning time constant
265
266     CPM_P_FACT_LPB_RESTORE_V = 520,      // 520 08 Bus V
    restoration voltage
267     CPM_P_PWRUP_HOLDOFF_TIME = 521,      // 521 09 Sensorless
    startup holdoff after powerup
268     CPM_P_FACT_LPB_ENTRY_V = 522,        // 522 0a Bus V minimum
    operation voltage
269
270     CPM_P_XPS_OUT_SRC_REG = 545,         // 545 21 XPS output
    register MUX
271     CPM_P_XPS_IN_SRC_REG,                // 546 22 XPS input
    register MUX
272
273     CPM_P_XPS_USER_IN_REG=548,           // 548 24 User driven
    input register
274     CPM_P_XPS_STATUS_IN_REG,             // 549 25 Drive
    controlled input register
275     CPM_P_XPS_INVERT_INPUT,              // 550 26 XPS bits
    driving pos limit feature
276     CPM_P_XPS_INVERT_OUTPUT,             // 551 27 XPS bits
    driving pos limit feature
277     CPM_P_XPS_FEAT_ENABLE=554,           // 554 2a XPS bits
    driving enable feature
278     CPM_P_XPS_FEAT_TRIGGER=556,          // 556 2c XPS bits
    driving trigger feature
279     CPM_P_XPS_FEAT_NODE_STOP,            // 557 2d XPS bits
    driving node stop feature
280     CPM_P_XPS_FEAT_RESET_TIMER,          // 558 2e XPS bits
    driving timer reset feature
281     CPM_P_XPS_FEAT_ATTN0,                // 559 2f XPS bits
    driving attn 0 feature
282     CPM_P_XPS_FEAT_ATTN1,                // 560 30 XPS bits
    driving attn 1 feature
283     CPM_P_XPS_FEAT_AT_HOME=563,          // 563 33 XPS bits
    driving homing feature
284     CPM_P_XPS_FEAT_POS_TRQ_FLDBK,        // 564 34 XPS bits
    driving pos torque limit feature
285     CPM_P_XPS_FEAT_NEG_TRQ_FLDBK,        // 565 35 XPS bits
    driving neg torque limit feature
```

```
286     CPM_P_XPS_FEAT_IN_POS_LIM,           // 566 36 XPS bits
    driving pos limit feature
287     CPM_P_XPS_FEAT_IN_NEG_LIM,           // 567 37 XPS bits
    driving neg limit feature
288
289     CPM_P_USER_SOFT_LIM_POSN_1 = 568,    // 568 38 User defined
    software limit 1
290     CPM_P_USER_SOFT_LIM_POSN_2,          // 569 39 User defined
    software limit 2
291     CPM_P_POWER_REG = 572,               // 572 3c Power Status Regis
292     CPM_P_FACT_PHASE_BD_SN = 574,        // 574 3e Phase board
    serial num
293     CPM_P_FACT_PHASE_BD_REV,             // 575 3f Phase board rev
294     CPM_P_DRV_MIN_OPER_VOLTS = 578,      // 578 42 Minimum
    operating voltage
295     CPM_P_OVER_TEMP_TRIP_USER,           // 579 43 User Over-Temp Tri
296 } cpmParams;
299 //*****************************************************************
    *************
300 // Insure structs are byte packed on target
301 #ifdef _MSC_VER
302 #pragma pack(push, 1)
303 #endif
304 //*****************************************************************
    *************
305
306
307
308
309
310
311 //*****************************************************************
    *************
312 // NAME
    *
313 //  cpmHwConfigReg union and cpmHwConfigFlds
327 struct _cpmHwConfigFlds {
329     enum _homingStyles {
330         HOME_TO_SWITCH = 0,
331         HOME_TO_HARDSTOP = 1
332     };
334     typedef enum _homingStyles homingStyles;
336     BFld: 4;                                  //  0-3
344     BFld AAfterStartMeas : 1;                 // 4
351     BFld BBeforeEndMeas : 1;                  // 5
358     BFld AtPositionMeas : 1;                  // 6
364     BFld VRSEnable : 1;                       // 7
371     BFld VectorValidation : 1;                // 8
373     BFld    : 1;                              // 9
383     BFld CapturePolarityHiSpd : 1;            // 10
391     BFld CapturePolarity : 1;                 // 11
400     BFld DataAcqOversampling : 1;             // 12
416     BFld SensorlessOnEnable : 1;              // 13
423     BFld HomingStyle : 1;                     // 14
```

```
424
435     BFld StepDirAsQuadrature: 1;                // 15
436
448     unsigned UsingAC : 1;                       // 16 0=DC, 1=AC
460     unsigned Using1PhAC : 1;                    // 17 0=3ph, 1=1ph
462     BFld: 2;                                    // 18-19
470     BFld ValidStartMethods       : 4;           // 20-23
471 };
473 typedef struct _cpmHwConfigFlds cpmHwConfigFlds;
475 //
    *
476 //*****************************************************************
    *************
477
478
479 //*****************************************************************
    *************
480 // NAME
    *
481 //  cpmHwConfigReg union
482 //
483 // DESCRIPTION
496 union PACK_BYTES _cpmHwConfigReg {
498     Uint32 bits;
500     cpmHwConfigFlds Fld;
501
502 #ifdef __cplusplus
503 #ifndef __TI_COMPILER_VERSION__
504
506     _cpmHwConfigReg() {
507         bits = 0;
508     }
509
510 #endif
511 #endif
512 };
514 typedef union _cpmHwConfigReg  cpmHwConfigReg;
515
517 //
    *
518 //*****************************************************************
    *************
519
520
521 //*****************************************************************
    *************
522 // NAME
    *
523 //  cpmAppConfigFlds union
524 //
525 // DESCRIPTION
541 struct PACK_BYTES _cpmAppConfigFlds {
545     typedef enum _limitModes {
549         LIMIT_MODE_OFF,                     //  0: ignore limits
```

```
554         LIMIT_MODE_POSITION,                //  1: Limit position
558         LIMIT_MODE_SOFT_POSN = 4,            //  4: Software
    position limit
563         LIMIT_MODE_SOFT_AND_INPS = 5
564     } limitModes;
565
569     typedef enum _operationalMode {
573         MOVE_STEP_DIR = 11,
577         IDLE_MODE = 20
578     } operationalMode;
579
580     // Note: the numbers in the field descriptions denote their bit
581     // range in the register.
589     BFld Mode : 6;                          // 0-5 Operational Mode
597     BFld TriggerGroup : 2;                  // 6-7
605     BFld LimitMode : 3;                     // 8-10 Limit switch mode
616     BFld LimitRampStop : 1;                 // 11 [TODO] Limit
    stop type
618     BFld: 2;                                // 12-13
626     BFld DirInputReverse : 1;               // 14
627
629     BFld: 1;                                // 15
630
639     BFld OffsetMoveInHomeDir : 1;           // 16
641     BFld: 1;                                // 17
642     BFld: 1;                                // 18
652     BFld HomingDirection : 1;               // 19
653
663     mnDisableActions DisableAction : 3;     // 20-22
664
666     BFld: 7;                                // 23-29
675     BFld ManualHoming : 1;                  // 30 Manual Homing
676 };
678 typedef struct  _cpmAppConfigFlds cpmAppConfigFlds;
681 //
    *
682 //*************************************************************
    *************
683
684
685 //*************************************************************
    *************
686 // NAME
    *
687 //  cpmAppConfigReg union
688 //
689 // DESCRIPTION
702 union PACK_BYTES _cpmAppConfigReg {
703
709     Uint32 bits;
714     cpmAppConfigFlds Fld;
715     // For C++ non-firmware usage, include a constructor.
716     #ifdef __cplusplus
717         #ifndef __TI_COMPILER_VERSION__
```

```
718
719         _cpmAppConfigReg() {
721             bits = 0;
722         }
724         #endif
725     #endif
726 };
728 typedef union _cpmAppConfigReg cpmAppConfigReg;
730 //
    *
731 //*****************************************************************
    *************
732
733
734 //*****************************************************************
    *************
735 //  NAME
    *
736 //      cpmOutFlds structure
737 //
738 //  DESCRIPTION:
766 struct PACK_BYTES _cpmOutFlds {
767     /*                              ------
768      *                               bit #
769      *                              ------ */
770
771     BFld                : 2;        // 1
772
781     BFld EnableReq   : 1;           // 2
789     BFld PosnCapture  : 1;          // 3
796     BFld Trigger     : 1;           // 4
804     BFld NodeStop    : 1;           // 5
813     BFld ResetTimer  : 1;           // 6
823     BFld XPSattn0    : 1;           // 7
833     BFld XPSattn1    : 1;           // 8
834
835     BFld             : 1;           // 9
836     BFld             : 1;           // 10
837
844     BFld TerminateHomeMove  : 1;    // 11
854     BFld StartPosTrqFldBk : 1;      // 12
864     BFld StartNegTrqFldBk : 1;      // 13
873     BFld PosLimitHit  : 1;          // 14
881     BFld NegLimitHit  : 1;          // 15
882 };
884 typedef struct _cpmOutFlds cpmOutFlds;
885 //              \endcond
    *
886 //*****************************************************************
    *************
887
888
889
```

```
890 //*****************************************************************
    *************
891 // NAME
    *
892 //  cpmOutReg union
893 //
894 // DESCRIPTION
901 union PACK_BYTES _cpmOutReg
902 {
906     Uint16 bits;
910     cpmOutFlds Fld;
911
912 #ifdef __cplusplus
913 #ifndef __TI_COMPILER_VERSION__
914
916     _cpmOutReg() {
917         bits = 0;
918     }
919
920 #endif
921 #endif
922 };
923
925 typedef union _cpmOutReg cpmOutReg;
926 //       \endcond
    *
927 //*****************************************************************
    *************
928
929
930
931 //*****************************************************************
    *************
932 // NAME
    *
933 //  cpmStatusRegFlds structure
934 //
935 // DESCRIPTION
936
950 struct PACK_BYTES _cpmStatusRegFlds {
960     typedef enum _velStates {
961         VEL_STOPPED,                    //< 0: Stopped
962         VEL_POSITIVE,                   //< 1: Positive velocity dete
963         VEL_NEGATIVE,                   //< 2: Negative velocity dete
966         VEL_BOTH
967     } velStates;
974     typedef enum _shutdownStates {
975         SHDN_STATE_IDLE         = 0,
976
977         SHDN_STATE_DELAY        = 1,
978         SHDN_STATE_RAMP         = 2,
979
980         SHDN_STATE_STOPPED      = 3
981     } shutdownStates;
```

```
    /*  bits 00-15 - - -
```

```
1380     // Avoid VS compiler padding bug, don't use enum
1381     BFld ShutdownState              : 2;    // 40-41
1382
1389     BFld HwFailure                  : 1;    // 42
1397     BFld TriggerArmed               : 1;    // 43
1405     BFld StepsActive                : 1;    // 44
1412     BFld IndexMtr                   : 1;    // 45
1421     BFld SoftwareInputs             : 1;    // 46
1423     // ------------------------------------------------------------
    -------------
1438     MN_EXPORT char * MN_DECL StateStr(char *buffer, size_t size) cons
1440 };
1442 typedef struct _cpmStatusRegFlds cpmStatusRegFlds;
1443 //
    *
1444 //*****************************************************************
    *************
1445
1446
1447 //*****************************************************************
    *************
1448 // NAME
    *
1449 //  cpmStatusReg union
1450 //
1451 // DESCRIPTION
1473 union PACK_BYTES _cpmStatusReg {
1477     Uint16 bits[3];
1481     cpmStatusRegFlds Fld;
1482
1483 #ifdef __cplusplus
1484
1485     // Constructor, create with bits cleared
1486     _cpmStatusReg() {                   // Clear bits on construction
1487         bits[0] = 0;
1488         bits[1] = 0;
1489         bits[2] = 0;
1490     }
1491 #endif
1492
1493
1494 };
1496 typedef union _cpmStatusReg cpmStatusReg;
1498 //
    *
1499 //*****************************************************************
    *************
1500
1501
1502 //*****************************************************************
    *************
1503 // NAME
    *
1504 //  cpmAlertFlds struct
```

```
1505 //
1506 // DESCRIPTION
1518 struct PACK_BYTES _cpmAlertFlds {
1520     commonAlertFlds Common;
1521     //                              ------------------------------
    ----------
1522     //                              |len|bit #|description
    |
1523     //                              ------------------------------
    ----------
1525     BFld : 4; /* 32-35 */
1538     BFld JrkLimRequestBad : 1; /* 36 */
1553     BFld MoveBufUnderrun : 1; /* 37 */
1566     BFld JrkLimVelRequestBad : 1; /* 38 */
1581     BFld MoveSpecAltered : 1; /* 39 */
1591     BFld PhaseSensorFailed : 1; /* 40 */
1602     BFld LimSwitchActivated : 1; /* 41 */
1615     BFld SoftLimitExceeded : 1; /* 42 */
1617                                 // Note: Must be separate around
    word boundary
1618     BFld : 5; /* 43-47 */
1619     BFld : 4; /* 48-51 */
1629     BFld ACPowerLost            : 1; /* 52 */ // AC Power lost
1638     BFld ACPhaseLost            : 1; /* 53 */ // AC Phase lost
1640     BFld : 1; /* 54 */
1652     BFld MtrTempLow             : 1; /* 55 */
1654     BFld : 2; /* 56-57 */
1668     BFld MtrVectorBad : 1; /* 58 */
1671     BFld : 1; /* 59 */
1681     BFld MtrEncGlitch : 1; /* 60 */
1693     BFld MtrEncOverspeed : 1; /* 61 */
1694
1695     BFld : 2; /* 62-63 */
1708     BFld MtrPhaseOverload : 1; /* 64 */
1710     BFld : 1; /* 65 */
1723     BFld MtrBadSetupParams : 1; /* 66 */
1737     BFld HardStopBrokeFree : 1; /* 67 */
1739     BFld : 1; /* 68 */
1754     BFld TrackingShutdown : 1; /* 69 */
1768     BFld RMSOverload : 1; /* 70 */
1783     BFld RMSOverloadShutdown : 1; /* 71 */
1785     BFld : 2; /* 72-73 */
1801     BFld BusVoltSat : 1; /* 74 */
1819     BFld TrqSat : 1; /* 75 */
1837     BFld NoCommSweepFailed : 1; /* 76 */
1852     BFld NoCommSweepReversed : 1; /* 77 */
1867     BFld NoCommFailed : 1; /* 78 */
1877     BFld IndexCountZeroWarn : 1; /* 79 */
1889     BFld DriveTempHigh : 1;     /* 80 */
1890
1901     BFld StatorTempHigh : 1; /* 81 */
1902
1914     BFld BusOverCurrent : 1; /* 82 */
1926     BFld BusOverVoltage : 1; /* 83 */
```

```
1939     BFld BusVoltageLow : 1; /* 84 */
1952     BFld BusRMSOverload : 1; /* 85 */
1954     BFld : 1; /* 86 */
1965     BFld MtrEncIndexMissing : 1; /* 87 */
1966
1978     BFld BusVoltageUnderOperatingV : 1; /* 88 */
1979
1980     BFld : 1; /* 89 */
1983     BFld : 3; /* 90-92 */
1995     BFld MtrEncIndexMisplaced : 1; /* 93 */
2007     BFld StepsDuringPosnRecovery : 1; /* 94 */
2008
2020     MN_EXPORT char * MN_DECL StateStr(char *buffer, size_t size) cons
2022 };
2023
2025 typedef struct _cpmAlertFlds  cpmAlertFlds;
2026 //
     *
2027 //*****************************************************************
     *************
2028
2029
2030 //*****************************************************************
     *************
2031 // NAME
     *
2032 //  cpmAlertReg union
2033 //
2034 // DESCRIPTION
2035 // Alert Register Size in octets: 96 bits = 3 32-bit entities = 12
     octets
2066 union _cpmAlertReg {
2068     Uint32 bits[3];
2070     cpmAlertFlds Fld;
2071     // If we have C++, include these useful member functions.
2073     #ifdef __cplusplus
2074     #ifdef __TI_COMPILER_VERSION__
2075     bool InShutdown();
2076     bool InLockdown();
2077     bool InBlockMotionShutdown();
2078     bool InBlockMotionSelfClearShutdown();
2079     bool InNodeNeedsRepairShutdown();
2080     bool InSeriousShutdown();
2081     bool InKillOutputsShutdown();
2082     bool InAlert();
2083     #endif
2084     // Constructor, create with bits cleared
2085     _cpmAlertReg() {                    // Clear bits on construction
2086         STATIC_ASSERT(ALERTREG_SIZEOCTETS == 12);
2087         bits[0] = 0;
2088         bits[1] = 0;
2089         bits[2] = 0;
2090     }
2091     #endif
```

```
2092 };
2095 typedef union _cpmAlertReg cpmAlertReg;
2097 //
     *
2098 //*****************************************************************
     *************
2099
2100
2101
2102 //*****************************************************************
     ************
2103 // NAME
     *
2104 //  cpmTuneConfigReg union
2105 //
2106 // DESCRIPTION
2116 union PACK_BYTES _cpmTuneConfigReg {
2118     typedef enum _antihuntModes {
2119         ANTIHUNT_OFF,
2120         ANTIHUNT_STD
2121     } antihuntModes;
2122
2128     Uint32 bits;
2129
2130     // Note: the numbers in the field descriptions denote their bit
2131     // range in the register.
2141     struct _cpmTuneConfigFlds {
2147         BFld AntiHunt           : 2;    // 0-1
2148
2149         BFld                    : 2;    // 2
2150
2156         BFld MtrVelocityEst     : 3;    // 4-6
2157
2166         BFld DontAskAboutLoad   : 1;    // 7
2174         BFld AutoTuning         : 1;    // 8
2175
2181         BFld UseVelCompInStim   : 1;    // 9
2189         BFld TrackGains         : 1;    // 10
2196         BFld EnableFineTuning   : 1;    // 11
2197
2198         BFld                    : 1;    // 12
2199
2206         BFld DisableNonLinearGains  : 1;// 13
2218         BFld DataCollectFreeRunApp  : 1;    // 14 Data collector
     free runs on App
2219
2220         BFld DataCollectFreeRunDiag : 1;    // 15 Data collector
     free runs on Diag
2221
2222         BFld VelocityCompensatorOverride : 1; // 16 Velocity
     Compensator Override
2223
2224     };
2228     struct _cpmTuneConfigFlds Fld;
```

```
2229     // For C++ non-firmware usage, include a constructor.
2230     #ifdef __cplusplus
2231         #ifndef __TI_COMPILER_VERSION__
2232
2234         _cpmTuneConfigReg() {
2235             bits = 0;
2236         }
2237
2238         #endif
2239     #endif
2240 };
2242 typedef union _cpmTuneConfigReg cpmTuneConfigReg;
2243
2244 //
    *
2245 //*****************************************************************
    *************
2246
2247 //=-=-=-
```

```
    BFld: 4;                    // 12-15 <-DO NOT combine these BFld entries!
    BFld: 3;                    // 16-18 <-DO NOT combine these BFld entries!
    BFld FWverMask : 1;         // 19: Inhibit display of minor FW version

    BFld: 1;                    // 20:
    BFld Vendor : 4;            // 21: Vendor code (LSB)
                                // 22:  0=TEKNIC
                                // 23:
                                // 24: Vendor code (MSB)
    BFld Debug : 3;             // 25: Debug Mode (LSB)
                                // 26:  0,7=off
                                // 27: Debug Mode (MSB)
    BFld SensorlessDelaySeed : 3;//  28-30
};
typedef struct _cpmOptionRegFlds cpmOptionsFlds;
//*
//*****************************************************************************


typedef struct _cpmSetFlagsFlds {
    BFld:1;
    BFld HardStop : 1;                  // 1  Simulate hitting a hard stop
    BFld HomingReq : 1;                 // 2  Request to initiate hom
    BFld HomingComplete : 1;            // 3  Homing completed manual
    BFld HomingInvalid : 1;             // 4  Force homing to be inva
} cpmSetFlagsFlds;

typedef union _cpmSetFlagsReg {
    Uint16 bits;
    cpmSetFlagsFlds fld;

#ifdef __cplusplus
#ifndef __TI_COMPILER_VERSION__

    _cpmSetFlagsReg() {
        bits = 0;
    }

#endif
#endif
} cpmSetFlagsReg;

//*****************************************************************************
// Restore byte packing
#ifdef _MSC_VER
#pragma pack(pop)
#endif
```

```
//*****************************************************************************
#endif // __PUBCPM_REGS_H__
//=============================================================================
//  END OF FILE pubCpmRegs.h
//=============================================================================
```

sFoundation 2.0

_cpmStatusRegFlds Member List

This is the complete list of members for **_cpmStatusRegFlds**, including all inherited members.

AbovePosn	_cpmStatusRegFlds
AFromStart	_cpmStatusRegFlds
AlertPresent	_cpmStatusRegFlds
AtTargetVel	_cpmStatusRegFlds
BFromEnd	_cpmStatusRegFlds
Disabled	_cpmStatusRegFlds
Enabled	_cpmStatusRegFlds
GoingDisabled	_cpmStatusRegFlds
Homing	_cpmStatusRegFlds
HwFailure	_cpmStatusRegFlds
InA	_cpmStatusRegFlds
InB	_cpmStatusRegFlds
InDisableStop	_cpmStatusRegFlds
InHardStop	_cpmStatusRegFlds
InMotion	_cpmStatusRegFlds
InNegLimit	_cpmStatusRegFlds
InPosLimit	_cpmStatusRegFlds
InvInA	_cpmStatusRegFlds
InvInB	_cpmStatusRegFlds
MotionBlocked	_cpmStatusRegFlds

MoveBufAvail	_cpmStatusRegFlds
MoveCanceled	_cpmStatusRegFlds
MoveCmdComplete	_cpmStatusRegFlds
MoveCmdNeg	_cpmStatusRegFlds
MoveDone	_cpmStatusRegFlds
NotReady	_cpmStatusRegFlds
OutOfRange	_cpmStatusRegFlds
PowerEvent	_cpmStatusRegFlds
Ready	_cpmStatusRegFlds
ShutdownState	_cpmStatusRegFlds
StatusEvent	_cpmStatusRegFlds
StepsActive	_cpmStatusRegFlds
TriggerArmed	_cpmStatusRegFlds
UserAlert	_cpmStatusRegFlds
VectorSearch	_cpmStatusRegFlds
WasHomed	_cpmStatusRegFlds

Homing

Homing Setup

To configure an SC Node for Homing, open ClearView and select Homing Setup from the Setup Menu. From this dialog box you can configure how a node homes.

- Homing Direction (Select direction of shaft rotation during homing)
- Homing Move Parameters (Set motor homing speed and acceleration.)
- Homing Target
    - Change of Input: If this mode is selected the node will home until the specified input changes state
    - HardStop: If this mode is selected, the node will home into a hard stop.
        - Homing Torque Limit (This will be the maximum force used to detect a hard stop during HardStop homing)
- Offset Move: This defines a post homing move that will automatically execute after home is located. After this move is executed the motor position will be set to zero.
- Homing Offset Direction (Select the direction of the offset move)

Homing is required to use soft limits and establish the node's number space for absolute type moves.

See also

sFnd::IHoming to initiate homing

ClearView V1.0.27 [Axis 0]: configuration file <Unloaded <CPM-SCHP-3432S-
File
Edit
Setup
Action
Details
Access [Full]
About
Input Actions...
Torque Limiting...
Homing Setup...
Software Limit Setup...
Auto-tune...
Fine Tuning...
Operating Mode...
Units
Disabl
Clear Ex
STOP
Torque
Override On
Stop
Enabled
RMS Max: 4%
4%
Bus Volts
74.6
Homing Setup
Homing Direction
CCW (+)
CW (-)
Homing Move Parameters
Max Velocity During Home Moves (RPM)
40.0
Max Accel. During Home Moves (RPM/s)
500
Homing Target
Change of Input
A Input Turns On
B Input Turns On
A Input Turns Off
B Input Turns Off
Hard Stop
Homing Torque Limit (% of max)
5.
Offset Move
Distance From Homing Target (cnts)
0
CCW (+)
Note: When homing is complete (after the offset move), the motor position will be = 0.
Homing Offset Direction
Reverse of Home
Same as Home
OK
Cancel

Move Done Feature

The *Move Done* feature provides a mechanism to specify the end of a move command, including any settling criteria. This feature uses the two specification parameters: *In-Range Position Window* and *Verify Time* for its operation.

The **_cpmStatusRegFlds::MoveDone** *Status Register* bit is cleared upon the start of the move command execution. When a move command has completed, a timer is started that is initialized to the *Move Done Time Constant* parameter. During this time, the *Position Tracking Error* is monitored. If the tracking error exceeds the *In Range Window*, the timer is reset back to the *Verify Time*. When the timer expires, the *Move Done* status bit is asserted and latches until the next move command is initiated, or if the tracking error exceeds the *In Range Window*.

To set the In-Range Winow, and Verify Time for you application, use the ClearView software to configure these parameters for each ClearPath-SC Node. These parameters can be found under the Details drop-down menu under "Move Status Indicators".

Configuring the In-Range Window, and Move Done Qualify Time

The following ClearView scope shot illustrates the effects of tracking and the generation of the *Move Done* event. In this example, the *In-Range Window* is set to 10 encoder counts and the *Verify Time* is set to 25 milliseconds. The bounce at the end of move (illustrated by the orange area in the move ribbon) resulted in an additional 25 milliseconds of delay before the *Move Done* event asserted at the end of the qualifying period (illustrated by the green area in the move ribbon). The pink marker shows the location of the *Move Done* event.

Move Done and Tracking Interactions

sFoundation 2.0

_mnStatusReg Member List

This is the complete list of members for **_mnStatusReg**, including all inherited members.

attnBits	**_mnStatusReg**	
bits	**_mnStatusReg**	
cpm	**_mnStatusReg**	
StateStr(char *buffer, size_t size)	**_mnStatusReg**	inline

_ShutdownInfo Member List

This is the complete list of members for **_ShutdownInfo**, including all inherited members.

enabled	**_ShutdownInfo**
statusMask	**_ShutdownInfo**
theStopType	**_ShutdownInfo**

sFoundation 2.0

commonAlertFlds Member List

This is the complete list of members for **commonAlertFlds**, including all inherited members.

ConfigOutOfDate	commonAlertFlds
EStopped	commonAlertFlds
FwError0	commonAlertFlds
FwError1	commonAlertFlds
FwInvalidConfiguration	commonAlertFlds
FwNetBufferOverrun	commonAlertFlds
FwSelfTest	commonAlertFlds
FwStackOverrun	commonAlertFlds
FwWatchdogRestarted	commonAlertFlds
HwADC	commonAlertFlds
HwADCsat	commonAlertFlds
HwClockProblem	commonAlertFlds
HwEEPROMdead	commonAlertFlds
HwFlashChanged	commonAlertFlds
HwFlashCorrupt	commonAlertFlds
HwPowerProblem	commonAlertFlds
HwRAM	commonAlertFlds
NetVoltageLow	commonAlertFlds
NetWatchdog	commonAlertFlds
RunTimeErr	commonAlertFlds

sFoundation 2.0

sdk Examples

Included in the ClearPath-SC software development kit(sdk) are several Microsoft Visual Studio project examples. These examples range from a simple hello world comunication example, to multithreaded dynamic application examples with error handling, and automatic node identification. New users just starting out, are advised to follow the examples in the loose sequence below.

HelloWorld

This example program will initiate communication with a network of ClearPath SC nodes.

Files

- **HelloWorld.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and Port objects. The main program executes, then waits for the user to press a key in order to end the program.

Example-StatusAlerts

This is an example for a network of ClearPath SC nodes. This example will discover all the nodes on the given port. Each Node is checked for alerts, and any found alerts are attempted to be cleared.

Files

- **Example-StatusAlerts.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and Port objects. The main program executes, then waits for the user to press a key in order to end the program.

Example-Motion

This is an example for a network of ClearPath SC nodes. This example will discover all the nodes on the given port, and execute sequential moves on each Node. The program will wait for each move to be finished before executing the next move on the next Node so only one node will be moving at a time.

Files

- **Example-Motion.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and Port objects. The main program executes a series of moves, then waits for the user to press a key in order to end the program.

Example-Homing

This is an example for a network of ClearPath SC nodes. This example will identify the first Node on the port and execute the homing sequence on that Node.

Files

- **Example-Homing.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and Port objects. The main program executes a series of moves, then waits for the user to press a key in order to end the program.

Example-GPIO

This is an example for a network of ClearPath SC nodes connected to an SC-HUB. This program will assert the General purpose Brake output on the SC-HUB board, and discover all the nodes on the given port. Finnally the program will record the current state of inputs A, and B on each node before shutting down.

Files

- **Example-GPIO.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and Port objects. The main program then waits for the user to press a key in order to end the program.

Example-SingleThreaded

This is an example for a network of ClearPath SC nodes. This program will discover all the nodes on the given port number and squentially perform a series of moves on each Node meant to exercise the basic features of a ClearPath SC.

Files

- **Example-SingleThreaded.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and Axes objects. The main program then waits for the user to press a key in order to end the program.
- **Axis Class** - This class talks to the nodes directly; there is one Axis object per node on the network. Each Axis runs in its own operations before calling the next Axis on the network. The Axis class basically starts up a node, issues a series of move and other operations, then closes down.

Example-MultiThreaded

This is a multi-Threaded example for a network of ClearPath SC nodes. This program will discover all the nodes on the given port number and synchronize moves among those nodes. The synchronization is fairly simple - the moves are all started at the same time (using a trigger) and the new moves are not sent until all the moves have completed. Each node has a different length of move, so they all finish at different times. The Axis class handles the interaction with a node on the network. The Supervisor class interacts with the Axis objects, telling them when to send their moves to the node. After each node has sent its move, the Supervisor starts all the moves with a Trigger command. The Axis then waits for MoveDone, and reports back to the Supervisor. The Supervisor waits for all the moves to finish, and starts the cycle again.

Files

- **Example-MultiThreaded.cpp** - This is the main function for the program. It holds a reference to the system manager of the network and also creates the Supervisor object. The main program then waits for the user to press a key in order to end the program.
- **Axis Class** - This class talks to the nodes directly; there is one Axis object per node on the network. Each Axis runs in its own thread, and is synchronized with the other nodes through the Supervisor class. The Axis class basically waits for the Supervisor to tell it to send its move, then waits for MoveDone, reporting back to the Supervisor when it is received. It also reports to the Supervisor if an error was encountered with the node.
- **Supervisor Class** - This class holds the references to all the nodes in the network, and synchronizes the moves among those nodes. This class runs its own thread to handle the synchronization without having to wait for the user input to

end the program. The Supervisor thread is a state machine that waits for the Axes to be idle, kicks the Axes to tell them to send their move to their node, waits for all the moves to be sent, sends a Trigger to start the moves, waits for all the Axes to report MoveDone, then starts all over again.

---

sFoundation 2.0

Example-MultiThreaded/Supervisor.h

Return to Example-MultiThreaded

Related Files:

- Main Function File: Example-MultiThreaded.cpp
- Axis Class Header File: Axis.h
- Axis Class Implementaion: Axis.cpp
- Supervisor Class Implementaion: Axis.cpp

```
/*************************************************************************
    ****
 * $Source: $
 * $Revision: 7 $ $State: $ $Date: 12/09/16 13:13 $
 *
 * !NAME!
 *      Supervisor
 *
 * DESCRIPTION:
 *      This class handles the synchronization between all the
 *      nodes in the system
 *
 * !end!
    *
 ***************************************************************************

/*************************************************************************
    ****
 * !NAME!
    *
 *  Supervisor.h headers
 *
    !0! */
#include "Axis.h"
#include "pubSysCls.h"
/*                                                                    !end!
 ***************************************************************************

//***********************************************************************
    *******
// NAME
    *
//  Supervisor class
//
// DESCRIPTION
//  The supervisory thread that sets up and handles the synchronization
//  between all the nodes.
```

```
//
class Supervisor {
private:
    thread m_thread;        // Supervisor's thread
    SysManager &m_sysMgr;   // sFoundation manager

    vector<Axis *> m_axes;      // the list of nodes

    bool m_quitting;        // are we quitting?

    // synchronization variables
    mutex m_mutex;
    condition_variable m_cond;
    // Flags to indicate that all nodes have reported back
    // for various conditions
    bool m_nodesIdle, m_nodesMoved, m_nodesDone, m_nodesKicked, m_nodesReady
    // Flags to keep track of which nodes have reported back
    // for various conditions
    vector<bool> m_axisReady;
    vector<bool> m_axisIdle;
    vector<bool> m_axisSentMove;
    vector<bool> m_axisDone;

    // State machine information
    enum SuperStateEnum {
        SUPER_IDLE,
        SUPER_WAIT_FOR_IDLE,
        SUPER_WAIT_FOR_MOVES_SENT,
        SUPER_WAIT_FOR_MOVES_DONE,
        SUPER_EXITED
    };

    SuperStateEnum m_state;
    SuperStateEnum m_lastState;

public:
    // Constructor
    Supervisor(vector<Axis*> theAxes, SysManager &myMgr);

    void CreateThread(){
        m_thread = thread(&Supervisor::SuperMain, this);
    }

    // Print stats for all the nodes
    void PrintStats(){
        printf("\nStats:\t\tnumMoves\t\tPosition\n");
        for (Uint16 iAxis = 0; iAxis < m_axes.size(); iAxis++){
            m_axes.at(iAxis)->PrintStats();
        }
    }

    // Signals and waits by the nodes
    void SignalIdle(Uint16 nodeNum);        // Nodes tell the supervisor
     when they are idle
```

```
    void WaitForKick();                     // Nodes need to wait for the
     supervisor to kick them
    void SignalMoveSent(Uint16 nodeNum);    // Nodes tell the supervisor
     when they've sent their move
    void SignalMoveDone(Uint16 nodeNum);    // Nodes tell the supervisor
     whne their move is done
    void SignalReady(Uint16 nodeNum);       // Nodes signal when they are
     initialized

    void SuperMain();       // the work for this thread

    // sFoundation supervisor for access to foundation support
    SysManager &Mgr() { return m_sysMgr; }

    // Quit and tell the nodes to quit, too
    void Quit(){
        printf("  Supervisor quitting\n");
        m_quitting = true;
        for (Uint16 iAxis = 0; iAxis < m_axes.size(); iAxis++)
            m_axes.at(iAxis)->Quit();
    }

    // Wait for the thread to finish
    void Terminate(){
        m_thread.join();
    }

private:
    // Mark that a particular node has signalled back for the given conditic
    void SignalPerNode(vector<bool> &nodeFlags,
                       bool &condition,
                       Uint16 nodeNum){
        unique_lock<mutex> lock(m_mutex);
        nodeFlags.at(nodeNum) = true;
        bool gotAll = true;
        for (Uint16 iAxis = 0; iAxis < nodeFlags.size(); iAxis++)
            gotAll = gotAll && nodeFlags.at(iAxis);
        if (gotAll){
            condition = true;
            m_cond.notify_all();
        }
    }

    // Wait for a given condition
    void WaitForCondition(bool &condition){
        unique_lock<mutex> lock(m_mutex);
        while (!condition && !m_quitting)
            m_cond.wait(lock);
    }

    // Clear the indicators for a particular condition
    void ResetCondition(vector<bool> &nodeFlags, bool &condition){
        unique_lock<mutex> lock(m_mutex);
        for (Uint16 iAxis = 0; iAxis < nodeFlags.size(); iAxis++)
```

```
            nodeFlags.at(iAxis) = false;
        condition = false;
    }

    // Set a condition that doesn't rely on all the nodes to do something
    void SetCondition(bool &condition){
        unique_lock<mutex> lock(m_mutex);
        condition = true;
        m_cond.notify_all();
    }

    // Clear a condition that doesn't relay on all the nodes
    void ResetCondition(bool &condition){
        unique_lock<mutex> lock(m_mutex);
        condition = false;
    }
};
```

pubSysCls.h File Reference

Go to the source code of this file.

Classes

class	**sFnd::IObj** Feature Availability Interface. More...
class	**sFnd::IObjWithNode** Feature Availability And Root Node Access Interface. More...
class	**sFnd::IObjWithPort** Feature Availability And Port Access Interface. More...
class	**sFnd::ValueBase** Node Parameter Container Base Class. More...
class	**sFnd::ValueDouble** This class object implements access to floating point parameters from a node. This class object implements access to floating point parameters within the node. More...
class	**sFnd::ValueSigned** Signed 32-bit Integer Parameter Object. More...

class	**sFnd::ValueUnsigned** Unsigned 32-bit Integer Parameter Object. More...
class	**sFnd::ValueString** String Parameter Object. More...
class	**sFnd::ValueStatus** Status Type Register Node Parameter Access Object. More...
class	**sFnd::ValueAlert** Alert Type Register Node Parameter Access Object. More...
class	**sFnd::ValuePowerReg** Power Status Register Node Parameter Access Object. More...
class	**sFnd::IBrakeControl** Brake Control Feature Interface. More...
class	**sFnd::IGrpShutdown** Group Shutdown Feature Interface. More...
class	**sFnd::IPortAdv** Advanced Port Features Interface. More...
class	**sFnd::IPort** Serial Port Interface. More...
class	**sFnd::IAttnNode** *Node Attention* Feature Interface. More...
class	**sFnd::IAttnPort**

	Serial Port Attention Feature Interface. More...
class	**sFnd::INodeAdv** Advanced Node Features Interface. More...
class	**sFnd::IHoming** Homing Feature Interface. More...
class	**sFnd::IInfoEx** Expert Information Interface. More...
class	**sFnd::IInfo** Node Information Interface. More...
class	**sFnd::ILimits** Torque and Position Limits Feature Interface. More...
class	**sFnd::IMotionAdv** Advanced Motion Control Interface. More...
class	**sFnd::IMotion** Motion Control Feature Interface. More...
class	**sFnd::ISetupEx** Expert Setup Interface. More...
class	**sFnd::ISetup** Node Setup Features Interface. More...
class	**sFnd::IStatusAdv** Advanced Status Features Interface. More...
class	**sFnd::IStatus** Node Status Features Interface. More...

class	**sFnd::INode** Virtual Node Interface. More...
class	**sFnd::INode::UseMutex** A Simple and Safe Mutex mechanism. More...
class	**sFnd::SysManager** sFoundation Top Level System Object. More...
class	**sFnd::SysManager::ThreadLock** A safe thread locking mechanism. More...

Namespaces

sFnd

Detailed Description

sFoundation 2.0 Software Framework Class Library.

sFoundation 2.0 software framework for host programming a system of ClearPath-SC nodes. This framework implements a rich class library for interacting with the nodes via C++.ZSFzsdfzdzd

sFoundation 2.0

inc Directory Reference

Directories

Generated by doxygen 1.8.11

inc-pub Directory Reference

Files

file **pubCoreRegs.h** [code]
Universal Register Definitions.

file **pubCpmRegs.h** [code]
S-Foundation register definitions for programming ClearPath-SC nodes.

file **pubNetAPI.h** [code]
Network Constants and types for interacting with nodes.

file **pubSysCIs.h** [code]

sFoundation 2.0

Advanced Features

Advanced Features

This section lists features only available in Advanced SC Nodes. For a system with both basic and advanced nodes you can use the **sFnd::IInfo::NodeType** function to determine whether the Node is Advanced or not.

- **Advanced Move Types**
- **Triggered Moves**
- **Position Capture**
- **Conditional Torque Limiting**
- **Attentions/Interupts for Event Driven Applications**

Move Types

Position Moves

These moves will cause the node to move from the current position to the specified destination via simple trapezoidal style move profiles. They all end with the node stopped and a Move Done event being issued when the end position has been reached within the settling criteria.

Upon reception of a positional move the current set of motion constraints(velocity/acceleration limits etc.) are copied and utilized by this move segment. The motion constraints may be changed after the move is sent (even if the move has not started yet) and will apply to the next move issued by the host.

The destination may be specified as relative to the start of move or to an absolute location in the commanded position space.

If a positional move is issued during a velocity move, it will cause the motor to accelerate to the velocity specified by the **sFnd::IMotion::VelLimit** until the specified location from the start of the move has occurred.

Motion Constraint Parameters

The following table provides a list of constraint parameters related to the moves:

Code	Description	Parameter
V-Lim	Velocity Limit constraint.	**sFnd::IMotion::VelLimit**
A-Lim	Acceleration Limit constraint.	**sFnd::IMotion::AccLimit**

The following diagram shows a typical velocity verses time profiles.

Trapezoidal Style

See also

sFnd::IMotion::MovePosnStart Function for Initiating Positional Moves

Move Done Feature for determining when a move has completed

CPM NodeStops and E-Stops to Cancel Motion For cancelling commanded moves

Advanced Move Types for additional move types available on Advanced ClearPath-SC motors

Velocity Moves

This type of move causes the velocity to ramp up or down at the **sFnd::IMotion::AccLimit** rate until the specified velocity is reached. Once this has occurred the move is considered finished (even though the node is still moving at the commanded velocity). The **sFnd::IMotion::VelocityReachedTarget** and **sFnd::IMotion::VelocityAtTarget** functions can be used to detect if the status register's **_cpmStatusRegFlds::AtTargetVel** bit is asserted to signal reaching the commanded velocity.

After the specified velocity is reached, the move is complete and will continue to run at this speed until another move is received or a cancelling **Nodestop** is performed.

Motion Constraint Parameters

The following table provides a list of constraint parameters related to the moves:

Code	Description	Parameter
A-Lim	Acceleration Limit constraint.	**sFnd::IMotion::AccLimit**

The following diagram show a typical velocity verses time profile.

Velocity Style

See also

sFnd::IMotion::MoveVelStart Function for Initiating Velocity Moves

CPM NodeStops and E-Stops to Cancel Motion For cancelling commanded moves

Advanced Move Types for additional move types available on Advanced ClearPath-SC motors

Motion Related Features

Motion Related Features

This section deals with motion related features including monitering status of move, making moves etc.

- **Move Types**
- **Homing**
- **Reading and Modifying Positions**
- **Move Done Feature**
- **Monitoring Moves and Move Status**
- **Motion Limits**

Conditional Torque Limiting

Conditional Torque Limiting

ClearPath-SC offers different torque/force foldback options that are useful for various scenarios. The torque foldback can be described as dynamic torque reduction applied during the ClearPath's operations. The foldback feature allows specialized actions for several different tasks including clamping operations and mechanically guiding parts to fit varied slot alignments.

Properly tuned servo systems apply full torque/force when tracking errors occur. This is especially true for errors that persist due to the integrator action of the servo compensator. To achieve a clamping force, the torque/force limit must be reduced to a level that will prevent crushing forces from being applied to the load and eventual motor overheating.

Each of the following foldback conditions run independently of each other. Each condition is compared to the others and, at each sample-time, the lowest torque limit value amongst all foldback features enabled is used for the following sample time. Each foldback mode will start folding back from the current torque/force level.

For example, consider when a motor's current holding torque is at 55% of maximum and the torque limit is set to 100%. Assume a foldback mode is engaged with a foldback limit of 10%. The torque will be instantly reduced from 55% to 10%. If a smooth transistion is desired, ClearPath-SC allows the specification of a relaxation time where the torque will asymptotically approach the new lower limit. The restoration of the foldback limit to a higher value occurs without any delay or shaping.

Configuring Torque Limiting through ClearView

For most applications the torque foldback settings/conditions will not change during operation, so it is easiest to configure these settings through the ClearView user interface. Once set, a user will be able to save a configuration file which will contain these settings to be loaded into any number of motors. The dialog box to configure Torque Limiting can be found under the setup dropdown menu in Clearview(see below).

Configuring Torque Limiting Through Clearview

Limit Conditions

- Move Done: After an axis makes a move it is sometimes desirable to reduce the holding torque. This is especially true in mechancs which are not back driveable. For these types of machines there can be a signifigant amount of torque used to hold the load in position at the end of a move due to mechanical wind-up and friction. Using Move Done Torque Foldback can signifigantly reduce the continous torque requirements of these types of systems. In this mode, torque is limited based on the Status register's **Move Done field.**
- A After Start/B Before End: In this mode, torque is limited based on the Status register's **A After Start** or **B Before End** fields.
- Absolute Position: In this mode, torque is limited based on the Node's absolute position. This mode requires **Homing**.
- Node Input A/B: In this mode, alternate torque limits can be triggered by Node input A or B

Reading and Modifying Positions

Overview

The CPM tracks the measured and commanded position of the motor and allows the application to adjust the position origin to create a user-based number space. The distances are always in integer units and can span a thirty-two bit number space. The positions wrap the number space in a two's complement manner. In other words, the next positive step from position 2,147,483,647 is -2,147,483,648.

Commanded Position

The commanded position is the current position the motor is servoing to. This can always be read from the **Commanded Position** parameter. If this position is read during a move, its value will be changing in real time. This value will not change if there is no commanded motion.

Measured Position

Once a move has been issued, the servo algorithms are used to drive the motor such that it corrects for differences between the move command and the actual motor position. Minimizing error between the motion command and motor load position is always the goal of a closed loop servo system. The measured position can be read from the **Measured Position** parameter.

The current tracking error is defined as: Commanded Position - Measured Position

The current tracking error can be read from the **sFnd::IMotion::PosnTracking** parameter.

Adjusting Positions

After an axis has successfully homed, or after the reception of a captured known position, it is often desirable to have the node's numeric origin match the application's origin. After homing the position is automatically set to zero at the home location (either a limit switch, or a hardstop depending on how homing is **configured**).

If an alternate number space is needed, the **sFnd::IMotion::AddToPosition** function can be used to simultaneously adjust the measured and commanded number space. For example, to "zero" the current position, one would add the negation of the current commanded position.

Note

If the **sFnd::IMotion::AddToPosition** function is called, soft limits will be ignored until the absolute position space is declared valid via the **sFnd::IHoming::SignalComplete** function.

Monitoring Moves and Move Status

Anatomy of a Move

Described below is the sequence of events that the host and motor execute when performing a positional move. This example assumes the motion constraints are set, the CPM is properly configured with a tuning file, and it is enabled.

1. Host issues a move command, using the **sFnd::IMotion::MovePosnStart** function.
2. The motor receives the move command from the host. The current motion constraints at the time of move command are captured and will apply to this move. Changes to any constraints after the move command is sent will only apply to the next move.
3. The motor checks the command for proper format and operational mode to allow its execution. If there is an issue with the command formatting, or operational mode settings in the motor, a **sFnd::_mnErr** will be thrown from the **sFnd::IMotion::MovePosnStart** function describing the nature of the problem.
4. The motor schedules the move to be executed:
    - If the previously issued move is not yet complete, the move is stored in a queue. Up to 16 Moves can be stored in each motor's move buffer, the *Status Register's* field **_cpmStatusRegFlds::MoveBufAvail** will be deasserted when this buffer is full. The host can query this register via the **sFnd::ValueStatus** object.
    - The motor will execute the next move in the queue as soon as the previous move is complete, unless the move is a triggered move.
    - If the move command is a **Triggered move**, the move will stay in the move queue buffer until a "Trigger" event is created by the motor or sent by the host, at which time the move will immediately execute. A move awaiting a trigger can be tested by checking the *Status Register's* **_cpmStatusRegFlds::TriggerArmed** bit. To cancel a

waiting move, use one of the procedures from the **CPM NodeStops and E-Stops to Cancel Motion** page.
    - When the queue has more space available, the status register's bit **_cpmStatusRegFlds::MoveBufAvail** will be asserted.
5. The motor executes the queued move per its arguments and the state of motion constraints when the move command was received. During the move, the following status events are set enroute.
    - **_cpmStatusRegFlds::AFromStart** *A From Start* of move.
    - **_cpmStatusRegFlds::BFromEnd** *B Counts from End* of move.
    - **_cpmStatusRegFlds::MoveDone** Move completed and settled.

Motion Limits

Tracking Error Limit

During normal operation a servo is always attempting to minimize **tracking error**, that is a servo will always attempt to exactly follow the commanded motion. Sometimes, it can become impossible for a servo to follow the commanded motion, typically this is due to either, a command that is too aggressive for the mechanics, or high friction/binding preventing the mechanics from moving. In these cases the servo motor will use full torque to attempt to maintain position, if this is not enough **tracking error** will increase. Eventually, if the **tracking error** exceeds the **tracking error limit**, the motor will shutdown. Some error is expected of all moves depending on the aggressiveness of the moves, and the loading of the mechanics, if the **tracking error limit** is set too low, you may get nuisance shutdowns. Typically this limit is set once through ClearView, and not changed during the application.

Access this dialog window through the details dropdown Menu

Global Torque Limit/RMS Torque Limit

ClearPath motors uses a settable **global torque limit** to prevent damage to both the motor, as well as your mechanics. There is also a built in **RMS torque level** to show how close you are to the continuous torque limit of the motor. This RMS level is based of the actual torque used over the past several minutes.

Software Limits

This feature lets you define travel limits in terms of encoder position, with no sensors or switches required. To use this feature, Check "Enable Software Limits" and set the two encoder positions as desired using ClearView.

The motor must have a sense of absolute position before Software Limits activate. Typically this is accomplished through homing and SW Limits are ignored until the motor is homed. Once Software Limits are activated, the motor will not execute a move that would violate the Software Limit settings.

See also

sFnd::IHoming::SignalComplete for applications that do not home, or adjust the absolute numberspace using the **sFnd::IMotion::AddToPosition** function