Motor Objects

Introduction

A **Motor** object manages a single motor on a controller. It represents the physical connections between the motor, drive, and associated I/O. The Motor object contains encoder data, limit switch, home sensor, amp fault and amp enable states, DAC outputs, and other status information.

For simple systems, there is a one-to-one relationship between the Axis, Filter and Motor objects.

For information about using absolute encoders with the MPI click here.

Methods

Create, Delete, Validate Methods

mpiMotor**Create**	Create Motor object
mpiMotor**Delete**	Delete Motor object
mpiMotor**Validate**	Validate Motor object

Configuration and Information Methods

mpiMotor**AmpEnableGet**	Get state of amp enable output
mpiMotor**AmpEnableSet**	Set state of amp enable output
meiMotor**AmpFault**	Writes the amp fault buffer data of a motor.
meiMotor**AmpFaultClear**	Clears the amp fault buffer data of a motor.
meiMotor**AmpWarning**	Writes the amp warning buffer of the motor.
meiMotor**AmpWarningClear**	Clears the motor's amp wanring message buffer.
mpiMotor**AxisMapGet**	Get object map of axes
meiMotor**CommutationModeGet**	Gets the commutation mode of a motor.
meiMotor**CommutationModeSet**	Sets the commutation mode of a motor.
mpiMotor**ConfigGet**	Get motor configuration
mpiMotor**ConfigSet**	Set motor configuration
meiMotor**ConfigStepper**	Configures a motor for stepper mode.
meiMotor**DacConfigGet**	Get a Motor's (motor) Dac configuration
meiMotor**DacConfigSet**	Set a Motor's (motor) Dac configuration
meiMotor**DacFlashConfigGet**	
meiMotor**DacFlashConfigSet**	
mpiMotor**Feedback**	Get feedback position

mpiMotor**FlashConfigGet**	Get flash config of motor
mpiMotor**FlashConfigSet**	Set flash config of motor
meiMotor**Info**	Get information about the network, node, and drive interface
mpiMotor**IoGet**	Get dedicated I/O bits
mpiMotor**IoSet**	Set dedicated I/O bits
mpiMotor**Status** / meiMotor**Status**	Get motor status

Event Methods

mpiMotor**EventConfigGet**	Get motor's event configuration
mpiMotor**EventConfigSet**	Set motor's event configuration
mpiMotor**EventNotifyGet**	Get motor's event mask for host notification.
mpiMotor**EventNotifySet**	Set motor's event mask for host notification.
mpiMotor**EventReset**	Reset events specified in event mask

Memory Methods

mpiMotor**Memory**	Get address of motor memory
mpiMotor**MemoryGet**	Copy motor memory to application memory
mpiMotor**MemorySet**	Copy application memory to motor memory

Action Methods

meiMotor**EncoderReset**	Clears encoder faults.
meiMotor**MultiTurnReset**	Clears the SynqNet drive multi-turn data for absolute type encoders.

Relational Methods

mpiMotor**Control**	Get handle to associated Control object
mpiMotor**FilterMapGet**	Get object map of associated Filters
mpiMotor**FilterMapSet**	Set the Filters using object map
mpiMotor**Number**	Get index number of motor (in Control list)

Other Methods

meiMotor**CompareListGet**	
meiMotor**EncoderRatio**	Get encoder ratio from the XMP.

Data Types

- MEIMotor**AmpFaults**
- MEIMotor**AmpFaultMsg**
- MEIMotor**AmpWarnings**
- MEIMotor**AmpWarningMsg**

MPIMotor**Brake**

MPIMotor**BrakeMode**

MPIMotor**Config** / MEIMotor**Config**

MEIMotor**DacConfig**

MEIMotor**DacChannelConfig**

MEIMotor**DacChannelStatus**

MEIMotor**DacStatus**

MEIMotor**DedicatedIn**

MEIMotor**DedicatedOut**

MEIMotor**DisableAction**

MPIMotor**Encoder** / MEIMotor**Encoder**

MPIMotor**EncoderFault**

MPIMotor**EncoderFaultMask**

MEIMotor**EncoderRatio**

MEIMotor**EncoderReverseModulo**

MEIMotor**EncoderType**

MPIMotor**EventConfig** / MEIMotor**EventConfig**

MPIMotor**EventTrigger**

MEIMotor**FaultBit**

MEIMotor**FaultConfig**

MEIMotor**FaultMask**

MPIMotor**Feedback**

MEIMotor**Info**

MEIMotor**InfoNodeType**

MPIMotor**Io**

MEIMotor**IoConfig**

MEIMotor**IoConfigIndex**

MEIMotor**IoMask**

MEIMotor**IoType**

MPIMotor**Message** / MEIMotor**Message**

MEIMotor**Status**

MEIMotor**StatusOutput**

MEIMotor**Stepper**

MEIMotor**StepperPulse**

MEIMotor**StepperPulseType**

MEIMotor**StepperStatus**

MPIMotor**Type**

Macros

mpiMotor**EncoderFaultMaskBIT**

Constants

MEIMotor**AmpMsgMAX**

MEIMotor**AmpFaultsMAX**

MEIMotor**AmpWarningsMAX**

mpiMotorCreate

Declaration

```
const MPIMotor mpiMotorCreate(MPIControl control,
                              long       number)
```

Required Header stdmpi.h

Description **MotorCreate** creates a Motor object associated with the motor identified by ***number***, and located on the motion controller (***control***). MotorCreate is the equivalent of a C++ constructor.

Return Values	
handle	to a Motor object
MPIHandleVOID	if the Motor object could not be created

See Also mpiMotorDelete | mpiMotorValidate

mpiMotorDelete

Declaration

```
long mpiMotorDelete(MPIMotor motor)
```

Required Header

stdmpi.h

Description

MotorDelete deletes a Motor object and invalidates its handle (***motor***). *MotorDelete* is the equivalent of a C++ destructor.

Return Values

MPIMessageOK	if *MotorDelete* successfully deletes a Motor object and invalidates its handle

See Also

mpiMotorCreate | mpiMotorValidate

mpiMotorValidate

Declaration

```
long mpiMotorValidate(MPIMotor motor)
```

Required Header

stdmpi.h

Description

MotorValidate validates a Motor object and its handle (***motor***).

Return Values

MPIMessageOK	if Motor is a handle to a valid object.

See Also

mpiMotorCreate | mpiMotorDelete

mpiMotorAmpEnableGet

Declaration

```
long mpiMotorAmpEnableGet(MPIMotor  motor,
                          long      *ampEnable)
```

Required Header

stdmpi.h

Description

MotorAmpEnableGet gets the state of the amp enable output for a Motor (***motor***) and writes it in the location pointed to by ***ampEnable***. Note that the actual state of amp enable output also depends upon the actual wiring and the polarity chosen in the instance of the MPIMotorConfig structure.

If "ampEnable" is	*Then*
FALSE (**0**)	the amp is disabled
TRUE (**1**)	the amp is enabled

Return Values	
MPIMessageOK	if *MotorAmpEnableGet* successfully writes the Motor's amp enable output state to the location

See Also

MPIMotorConfig | mpiMotorAmpEnableSet

mpiMotorAmpEnableSet

Declaration

```
long mpiMotorAmpEnableSet(MPIMotor  motor,
                          long      ampEnable)
```

Required Header

stdmpi.h

Description

MotorAmpEnableSet sets the state of the amp enable output for a Motor (***motor***) to ***ampEnable***. Note that the actual state of amp enable output also depends upon the actual wiring and the polarity chosen in the instance of the MPIMotorConfig structure.

If "ampEnable" is	*Then*
FALSE (**0**)	the amp will be disabled
TRUE (**1**)	the amp will be enabled

Return Values	
MPIMessageOK	if *MotorAmpEnableSet* successfully sets the Motor's amp enable output state to ampEnable

See Also

MPIMotorConfig | mpiMotorAmpEnableGet

meiMotorAmpFault

Declaration

```
long meiMotorAmpFault(MPIMotor           motor,
                      MEIMotorAmpFaults  *fault);
```

Required Header stdmei.h

Description **MotorAmpFault** reads the amp fault buffer from a motor and writes the data into a structure pointed to by fault.

motor	a handle to the Motor object
***fault**	a pointer to a structure containing the number of amp faults, their coded values and message strings. See MEIMotorAmpFaults.

Return Values

MPIMessageOK	if *MotorAmpFault* writes the amp fault buffer data

See Also meiMotorAmpFaultClear | meiMotorAmpWarning | meiMotorWarningClear

meiMotorAmpFaultClear

Declaration

```
long meiMotorAmpFaultClear(MPIMotor    motor);
```

Required Header stdmei.h

Description **MotorAmpFaultClear** flushes the motor's amp fault message buffer. The number of amp faults is set to zero, the coded values are set to zero, and the messages are cleared.

motor	a handle to the Motor object

Return Values

MPIMessageOK	if *MotorAmpFaultClear* successfully flushes the Motor's amp fault message buffer.

See Also meiMotorAmpFault | meiMotorAmpWarning | meiMotorWarningClear

meiMotorAmpWarning

Declaration

```
long meiMotorAmpWarning(MPIMotor              motor,
                        MEIMotorAmpWarnings  *warning);
```

Required Header stdmei.h

Description **MotorAmpWarning** reads the amp warning buffer from a motor and writes the data into a structure pointed to by warning.

motor	a handle to the Motor object
***warning**	a pointer to a structure containing the number of amp warnings, their coded values and message strings. See MEIMotorAmpWarnings.

Return Values

MPIMessageOK	if *MotorAmpWarning* successfully writes the amp warning buffer data

See Also meiMotorAmpWarningClear | meiMotorAmpFault | meiMotorAmpFaultClear

meiMotorAmpWarningClear

Declaration `long meiMotorAmpWarningClear(MPIMotor motor);`

Required Header stdmei.h

Description **MotorAmpWarningClear** flushes the motor's amp warning message buffer. The number of amp warnings is set to zero, the coded values are set to zero, and the messages are cleared.

motor	a handle to the Motor object

Return Values

MPIMessageOK	if *MotorAmpWarningClear* successfully ...

See Also meiMotorAmpWarning | meiMotorAmpFault | meiMotorAmpFaultClear

mpiMotorAxisMapGet

Declaration

```
long mpiMotorAxisMapGet(MPIMotor      motor,
                        MPIObjectMap  *axismap)
```

Required Header

stdmpi.h

Description

MotorAxisMapGet gets the object map of the Axes associated with a Motor (**motor**) and writes it into the structure pointed to by **axismap**.

Return Values	
MPIMessageOK	if *MotorAxisMapGet* successfully writes the Motor's object map of Axes to the structure

See Also

meiMotorCommutationModeGet

Declaration

```
long meiMotorCommutationModeGet(MPIMotor       motor,
                                MEIXmpCommMode *mode)
```

Required Header

stdmei.h

Description

MotorCommutationModeGet gets the commutation mode of a Motor (***motor***) and writes it to the location pointed to by ***mode***.

Return Values	
MPIMessageOK	if *MotorCommutationModeGet* successfully gets the commutation mode of a Motor and writes it to the location

See Also

meiMotorCommutationModeSet

meiMotorCommutationModeSet

Declaration

```
long meiMotorCommutationModeSet(MPIMotor        motor,
                                MEIXmpCommMode  mode)
```

Required Header stdmei.h

Description **MotorCommutationModeSet** sets the commutation mode of a Motor (***motor***) to ***mode***.

Return Values	
MPIMessageOK	if *MotorCommutationModeSet* successfully sets the commutation mode of a Motor to ***mode***

See Also meiMotorCommutationModeGet

mpiMotorConfigGet

Declaration

```
long mpiMotorConfigGet(MPIMotor       motor,
                       MPIMotorConfig *config,
                       void           *external)
```

Required Header

stdmpi.h

Description

MotorConfigGet gets a Motor's (***motor***) configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The configuration information in ***external*** is in addition to the configuration information in ***config***, i.e, the configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

XMP Only

external either points to a structure of type MEIMotorConfig{} or is NULL.

Return Values

MPIMessageOK	if *MotorConfigGet* successfully writes the Motor's configuration to the structure(s)

See Also

MEIMotorConfig | mpiMotorConfigSet

mpiMotorConfigSet

Declaration

```
long mpiMotorConfigSet(MPIMotor       motor,
                       MPIMotorConfig *config,
                       void           *external)
```

Required Header

stdmpi.h

Description

MotorConfigSet sets a Motor's (***motor***) configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The configuration information in ***external*** is *in addition* to the configuration information in ***config***, i.e, the configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

XMP Only

external either points to a structure of type **MEIMotorConfig{}** or is NULL.

Return Values

MPIMessageOK	if *MotorConfigSet* successfully sets the Motor's configuration using data from the structure(s)

See Also

mpiMotorConfigGet | MEIMotorConfig
Special Note: Using mpiMotorConfigSet with Absolute Encoders

meiMotorConfigStepper

Declaration

```
long meiMotorConfigStepper(MPIMotor          motor,
                           MEIMotorConfig    *config,
                           long              stepperNumber)
```

Required Header

stdmei.h

Description

MotorConfigStepper modifies the motor configuration structure pointed to by config, to use a step engine (stepperNumber) from another motor. By default, each motor uses its own step engine. Do NOT use more than one motor per step engine. Use the methods mpiMotorConfigGet/Set(…) to read/write the motor configuration from/to the controller.

motor	a handle to the Motion object
***config**	a pointer to the motion frame buffer status structure returned by the method
stepperNumber	index to a step engine

Return Values

MPIMessageOK	if *MotorConfigStepper* successfully modifies the motor configuration structure pointed to by ***config***

See Also

mpiMotorConfigGet | mpiMotorConfigSet

meiMotorDacConfigGet

Declaration

```
long meiMotorDacConfigGet(MPIMotor           motor,
                          MEIMotorDacConfig  *config);
```

Required Header stdmei.h

Description **MotorDacConfigGet** gets a Motor's (***motor***) DAC configuration and writes it to the structure pointed to by ***config***.

motor	a handle to the Motor object.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values

MPIMessageOK	if *MotorDacConfigGet* successfully writes the Motor's Dac configuration to the ***config*** structure.

See Also meiMotorDacConfigSet | MEIMotorDacConfig

meiMotorDacConfigSet

Declaration

```
long meiMotorDacConfigSet(MPIMotor          motor,
                          MEIMotorDacConfig *config);
```

Required Header stdmei.h

Description **MotorDacConfigSet** configures a Motor's (***motor***) DAC using data from the structure pointed to by ***config***.

motor	a handle to the Motor object.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values

MPIMessageOK	if *MotorDacConfigSet* successfully writes the Motor's Dac configuration to the controller.

See Also meiMotorDacConfigGet | MEIMotorDacConfig

meiMotorDacFlashConfigGet

Declaration

```
long meiMotorDacFlashConfigGet(MPIMotor           motor,
                               void               *flash,
                               MEIMotorDacConfig  *config);
```

Required Header stdmei.h

Description **MotorDacFlashConfigGet** gets a Motor's (***motor***) DAC configuration from flash memory and writes it to the structure pointed to by ***config***.

motor	a handle to the Motor object
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values

MPIMessageOK	if *MotorDacFlashConfigGet* successfully writes the DAC's configuration from flash memory to the structure.

See Also meiMotorDacFlashConfigSet | meiMotorDacConfigGet | MEIMotorDacConfig

meiMotorDacFlashConfigSet

Declaration

```
long meiMotorDacFlashConfigSet(MPIMotor           motor,
                               void               *flash,
                               MEIMotorDacConfig  *config);
```

Required Header stdmei.h

Description **MotorDacFlashConfigSet** sets a Motor's (***motor***) DAC configuration to flash memory using data from the structure pointed to by ***config***.

motor	a handle to the Motor object
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values

MPIMessageOK	if *MotorDacFlashConfigSet* successfully writes the DAC's configuration to flash memory using data from the ***config*** structure.

See Also meiMotorDacFlashConfigGet

mpiMotorFeedback

Declaration

```
long mpiMotorFeedback(MPIMotor          motor,
                      MPIMotorFeedback  *feedback)
```

Required Header stdmpi.h

Description **MotorFeedback** gets the feedback position of a Motor (***motor***) and writes it into the location pointed to by ***feedback***.

Return Values	
MPIMessageOK	if *MotorFeedback* successfully writes the feedback position into the location.

See Also mpiMotorFeedbackConfigSet

mpiMotorFlashConfigGet

Declaration

```
long mpiMotorFlashConfigGet(MPIMotor        motor,
                            void            *flash,
                            MPIMotorConfig  *config,
                            void            *external)
```

Required Header

stdmpi.h

Description

MotorFlashConfigGet gets a Motor's (***motor***) flash configuration and writes it in the structure pointed to by ***config***, and also writes it in the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Motor's flash configuration information in ***external*** is in addition to the Motor's flash configuration information in ***config***, i.e, the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

XMP Only	***external*** either points to a structure of type MEIMotorConfig{} or is NULL.

Return Values

MPIMessageOK	if *MotorFlashConfigGet* successfully writes the Motor's flash configuration to the structure(s) ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigSet

mpiMotorFlashConfigSet

Declaration

```
long mpiMotorFlashConfigSet(MPIMotor        motor,
                            void            *flash,
                            MPIMotorConfig  *config,
                            void            *external)
```

Required Header

stdmpi.h

Description

MotorFlashConfigSet sets a Motor's (***motor***) flash configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Motor's flash configuration information in ***external*** is in addition to the Motor's flash configuration information in ***config***, i.e., the flash configuration information in ***config*** and in external is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

XMP Only	***external*** either points to a structure of type MEIMotorConfig{} or is NULL.

Return Values

MPIMessageOK	if *MotorFlashConfigSet* successfully sets the Motor's flash configuration using data from the structure(s) ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	returned because the SynqNet topology has not yet been saved to flash using meiSynqNetFlashTopologySave(…).

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigGet | meiSynqNetFlashTopologySave

meiMotorInfo

Declaration

```
long meiMotorInfo(MPIMotor      motor,
                  MEIMotorInfo  *info);
```

Required Header

stdmei.h

Description

MotorInfo reads the static information about the network, node, and drive interface associated with the motor object, and writes it into the structure pointed to by ***info***.

motor	a handle to the Motor object
***info**	a pointer to a motor information structure

Return Values

MPIMessageOK	if *MotorInfo* successfully writes the network and node/drive interface information into the Motor structure.

See Also

meiMotorStatus

mpiMotorIoGet

Declaration

```
long mpiMotorIoGet(MPIMotor    motor,
                   MPIMotorIo  *io)
```

Required Header stdmpi.h

Description **MotorIoGet** gets a Motor's (***motor***) dedicated I/O bits and writes them into the structure pointed to by ***io***.

NOTE

When using I/O on SynqNet nodes, use the motor I/O masks in the drive's header file for clearer code. Most SynqNet nodes have a header file that defines things that are specific to that drive. The header files are found in C:\mei\XMP\sqNodeLib\include. An example is shown below that reads the hall sensors from the Trust TA802.

```
/*
        Shows the hall sensors.
        Make sure you include trust_ta800.h. Ex:
        #include "C:\mei\XMP\sqNodeLib\include\trust_ta800.h"

        The hall sensor masks are found in the enum TA800MotorIoMask.
*/

void showHalls(MPIMotor motor)
{
    MPIMotorIo io;
      long returnValue;
      long a, b, c;

      while (meiPlatformKey(MPIWaitPOLL) <= 0)
      {
         returnValue = mpiMotorIoGet(motor, &io);
         msgCHECK(returnValue);

         a = ((io.input & TA800MotorIoMaskHALL_A) == TA800MotorIoMaskHALL_A);
         b = ((io.input & TA800MotorIoMaskHALL_B) == TA800MotorIoMaskHALL_B);
         c = ((io.input & TA800MotorIoMaskHALL_C) == TA800MotorIoMaskHALL_C);

         /* Prints a 1 or 0 indicating the hall state */
         printf("Hall A %d B %d C %d\t\t\r", a, b, c);

         meiPlatformSleep(100);
      }
}
```

Sample Code

```
/* Poll io inputs for a motor and print to the screen */
void readIo(MPIMotor motor)
{
    MPIMotorIo io;
    long returnValue;

    while (meiPlatformKey(MPIWaitPOLL) <= 0)
    {
        returnValue = mpiMotorIoGet(motor, &io);
        msgCHECK(returnValue);

        printf("\rIO %x", io.input);

        meiPlatformSleep(100); /* Wait 100 mSec */
    }
}
```

Return Values

MPIMessageOK	if *MotorIoGet* successfully writes the Motor's dedicated I/O bits to the structure.

See Also

mpiMotorIoSet

mpiMotorIoSet

Declaration

```
long mpiMotorIoSet(MPIMotor    motor,
                   MPIMotorIo  *io)
```

Required Header stdmpi.h

Description **MotorIoSet** sets a Motor's (***motor***) dedicated I/O bits using data from the structure pointed to by ***io***.

NOTE

When using I/O on SynqNet nodes, use the motor I/O masks in the drive's header file for clearer code. Most SynqNet nodes have a header file that defines things that are specific to that drive. The header files are found in C:\mei\XMP\sqNodeLib\include. An example is shown below that reads the hall sensors from the Trust TA802.

```
/*
        Shows the hall sensors.
        Make sure you include trust_ta800.h. Ex:
        #include "C:\mei\XMP\sqNodeLib\include\trust_ta800.h"

        The hall sensor masks are found in the enum TA800MotorIoMask.
*/

void showHalls(MPIMotor motor)
{
    MPIMotorIo io;
      long returnValue;
      long a, b, c;

      while (meiPlatformKey(MPIWaitPOLL) <= 0)
      {
         returnValue = mpiMotorIoGet(motor, &io);
         msgCHECK(returnValue);

         a = ((io.input & TA800MotorIoMaskHALL_A) == TA800MotorIoMaskHALL_A);
         b = ((io.input & TA800MotorIoMaskHALL_B) == TA800MotorIoMaskHALL_B);
         c = ((io.input & TA800MotorIoMaskHALL_C) == TA800MotorIoMaskHALL_C);

         /* Prints a 1 or 0 indicating the hall state */
         printf("Hall A %d B %d C %d\t\t\r", a, b, c);

         meiPlatformSleep(100);
      }
}
```

Return Values	
MPIMessageOK	if *MotorIoSet* successfully sets the Motor's dedicated I/O bits using data from the structure

See Also

mpiMotorIoGet

mpiMotorStatus / meiMotorStatus

mpiMotorStatus

Declaration

```
long mpiMotorStatus(MPIMotor    motor,
                    MPIStatus   *status,
                    void        *external)
```

Required Header stdmpi.h

Description

MotorStatus writes a Motor's (***motor***) status into the structure pointed to by ***status***, and also into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The ***motor's*** status information in ***external*** is in addition to the motor's status information in ***status***, i.e, the status configuration information in ***status*** and in ***external*** is not the same information. Note that ***external*** can be NULL (but status must not be NULL).

motor	a handle to the Motor object
***status**	a pointer to the motor status structure returned by the method
***external**	a pointer to an implementation-specific structure

XMP Only ***external*** either points to a structure of type MEIMotorStatus{...} or is NULL.

Return Values	
MPIMessageOK	if *MotorStatus* successfully gets the status of a Motor object.
MPIMessageARG_INVALID	if the *status* pointer is NULL.

See Also MPIStatus

meiMotorStatus

Declaration

```
long meiMotorStatus(MPIControl  control,
                    long        motorNumber
                    MPIStatus   *status,
                    void        *external)
```

Required Header stdmei.h

Description

MotorStatus gets a Motor's status and writes it to the structure pointed to by ***status***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The ***motor's*** status information in ***external*** is in addition to the motor's status information in ***status***, i.e, the status configuration information in ***status*** and in ***external*** is not the same information. Note that ***external*** can be NULL (but status must not be NULL).

control	a handle to the Control object
motorNumber	index to the motor
***status**	a pointer to the motor status structure returned by the method
***external**	pointer to an implementation-specific structure

XMP Only	***external*** either points to a structure of type MEIMotorStatus{...} or is NULL.

Return Values

MPIMessageOK	if *MotorStatus* successfully gets the status of a Motor object.
MPIMessageARG_INVALID	if the *status* pointer is NULL.

See Also

MPIStatus | MEIMotorStatus

mpiMotorEventConfigGet

Declaration

```
long mpiMotorEventConfigGet(MPIMotor            motor,
                            MPIEventType        eventType,
                            MPIMotorEventConfig *eventConfig,
                            void                *external)
```

Required Header

stdmpi.h

Description

MotorEventConfigGet gets the Motor's (***motor***) configuration for the event specified by ***eventType*** and writes it into the structure pointed to by ***eventConfig***, and also writes it to the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The event configuration information in ***external*** is in addition to the event configuration information in ***eventConfig***, i.e, the event configuration information in ***eventConfig*** and in ***external*** is not the same information. Note: Set ***eventConfig*** or ***external*** to NULL. One must be NULL, the other must be passed a pointer.

XMP Only

external either points to a structure of type MEIMotorEventConfig{} or is NULL.

Sample Code

```
    for(index = 0; index < AXIS_COUNT; index++)
            { // turn off error limit and limit switch actions for
             motors 0 to AXIS_COUNT
                    returnValue = mpiMotorEventConfigGet(motor[index],
                            MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
                    msgCHECK(returnValue);

                    eventConfig.action = MPIActionNONE;

                    returnValue = mpiMotorEventConfigSet(motor[index],
                            MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
                    msgCHECK(returnValue);

                    returnValue = mpiMotorEventConfigGet(motor[index],
                            MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
                    msgCHECK(returnValue);

                    eventConfig.action = MPIActionNONE;

                    returnValue = mpiMotorEventConfigSet(motor[index],
                            MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
                    msgCHECK(returnValue);

                    returnValue = mpiMotorEventConfigGet(motor[index],
```

```
                        MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
                msgCHECK(returnValue);

                eventConfig.action = MPIActionNONE;

                returnValue = mpiMotorEventConfigSet(motor[index],
                        MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
                msgCHECK(returnValue);
        }
```

Return Values

MPIMessageOK	if *MotorEventConfigGet* successfully writes the Motor's configuration for the event to the structure(s)

See Also

MEIMotorEventConfig | mpiMotorEventConfigSet

mpiMotorEventConfigSet

Declaration

```
long mpiMotorEventConfigSet(MPIMotor            motor,
                            MPIEventType        eventType,
                            MPIMotorEventConfig *eventConfig,
                            void                *external)
```

Required Header

stdmpi.h

Description

MotorEventConfigSet reads the structure pointed to by ***eventConfig*** and sets the Motor's (***motor***) configuration for the event specified by ***eventType***.

The event configuration information in ***external*** is in addition to the event configuration information in ***eventConfig***, i.e, the event configuration information in ***eventConfig*** and in ***external*** is not the same information.

NOTE: Set ***eventConfig*** or ***external*** to NULL. One must be NULL, the other must be passed a pointer.

XMP Only

external either points to a structure of type MEIMotorEventConfig{} or is NULL.

Return Values

MPIMessageOK	if *MotorEventConfigGet* successfully writes the Motor's configuration for the event to the structure(s)

See Also

MEIMotorEventConfig | mpiMotorEventConfigGet

mpiMotorEventNotifyGet

Declaration

```
long mpiMotorEventNotifyGet(MPIMotor      motor,
                            MPIEventMask  *eventMask,
                            void          *external)
```

Required Header

stdmpi.h

Description

MotorEventNotifyGet writes the event mask (that specifies the event type(s) for which host notification has been requested) to the location pointed to by ***eventMask***, and also writes it into the implementation-specific location pointed to by ***external*** (if ***external*** is not NULL).

The event notification information in ***external*** is in addition to the event notification information in ***eventmask***, i.e, the event notification information in ***eventmask*** and in ***external*** is not the same information. Note that ***eventmask*** or ***external*** can be NULL (but not both NULL).

Event notification is enabled for event types specified in ***eventmask***, which is a bit mask of MPIEventMask bits associated with the desired MPIEventType values. Event notification is disabled for event types not specified in ***eventmask***. The MPIEventMask bits must be set of cleared using the MPIEventMask macros.

XMP Only

external either points to a structure of type MEIEventNotifyData{} or is NULL. The MEIEventNotifyData{} structure is an array of firmware addresses, whose contents are placed into the MEIEventStatusInfo{} structure (of all events generated by this object).

Return Values

MPIMessageOK	if *MotorEventNotifyGet* successfully writes the event mask to the location(s)

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo
mpiMotorEventNotifySet

mpiMotorEventNotifySet

Declaration

```
long mpiMotorEventNotifySet(MPIMotor      motor,
                            MPIEventMask  eventMask,
                            void          *external)
```

Required Header

stdmpi.h

Description

MotorEventNotifySet requests host notification of the event(s) that are generated by ***motor*** and specified by ***eventMask***, and also specified by the implementation-specific location pointed to by ***external*** (if ***external*** is not NULL).

The event notification information in ***external*** is in addition to the event notification information in ***eventmask***, i.e, the event notification information in ***eventmask*** and in ***external*** is not the same information. Note that ***eventmask*** or ***external*** can be NULL (but not both NULL).

Event notification is enabled for event types specified in ***eventMask***, a bit mask of MPIEventMask bits associated with the desired MPIEventType values. Event notification is disabled for event types that are not specified in ***eventMask***. The MPIEventMask bits must be set of cleared using the MPIEventMask macros.

The mask of event types generated by a Motor object consists of bits from MPIEventMaskMOTION and MPIEventMaskAXIS.

XMP Only

external either points to a structure of type MEIEventNotifyData{} or is NULL. The MEIEventNotifyData{} structure is an array of firmware addresses, whose contents are placed into the MEIEventStatusInfo{} structure (of all events generated by this object).

To	Then
enable host notification of all events	set ***eventmask*** to MPIEventMaskALL
disable host notification of all events	set ***eventmask*** to MPIEventTypeNONE

Sample Code

```
for(index = 0; index < AXIS_COUNT; index++)
        { // turn off error limit and limit switch actions for
         motors 0 to AXIS_COUNT
                returnValue = mpiMotorEventConfigGet(motor[index],
                        MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
                msgCHECK(returnValue);

                eventConfig.action = MPIActionNONE;

                returnValue = mpiMotorEventConfigSet(motor[index],
                        MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
                msgCHECK(returnValue);

                returnValue = mpiMotorEventConfigGet(motor[index],
                        MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
                msgCHECK(returnValue);

                eventConfig.action = MPIActionNONE;

                returnValue = mpiMotorEventConfigSet(motor[index],
                        MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
                msgCHECK(returnValue);

                returnValue = mpiMotorEventConfigGet(motor[index],
                        MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
                msgCHECK(returnValue);

                eventConfig.action = MPIActionNONE;

                returnValue = mpiMotorEventConfigSet(motor[index],
                        MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
                msgCHECK(returnValue);
        }
```

Return Values

MPIMessageOK	if *MotorEventNotifySet* successfully requests host notification of the event(s) that are specified by ***eventMask*** and generated by ***motor***

See Also MPIEventType | MEIEventNotifyData | MEIEventStatusInfo
mpiMotorEventNotifyGet

mpiMotorEventReset

Declaration

```
long mpiMotorEventReset(MPIMotor      motor,
                        MPIEventMask  eventMask)
```

Required Header

stdmpi.h

Description

MotorEventReset resets the event(s) that are specified in ***eventMask*** and generated by ***motor***.
Your application must call *MotorEventReset* only after one or more latchable events have occurred.

Return Values

MPIMessageOK	if *MotorEventReset* successfully resets the event(s) that are specified in ***eventMask*** and generated by ***motor***

See Also

mpiMotorMemory

Declaration

```
long mpiMotorMemory(MPIMotor  motor,
                    void      **memory)
```

Required Header stdmpi.h

Description **MotorMemory** sets (writes) an address (used to access a Control object's memory) to the contents of ***memory***.

Return Values	
MPIMessageOK	if *MotorMemory* successfully writes the Motor's address to the contents of *memory*.

See Also mpiMotorMemoryGet | mpiMotorMemorySet

mpiMotorMemoryGet

Declaration

```
long mpiMotorMemoryGet(MPIMotor motor,
                       void     *dst,
                       void     *src,
                       long     count)
```

Required Header stdmpi.h

Description **MotorMemoryGet** copies ***count*** bytes of a Motor's (***motor***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *MotorMemoryGet* successfully copies data from Motor memory to application memory

See Also mpiMotorMemorySet | mpiMotorMemory

mpiMotorMemorySet

Declaration

```
long mpiMotorMemorySet(MPIMotor motor,
                       void     *dst,
                       void     *src,
                       long     count)
```

Required Header

stdmpi.h

Description

MotorMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Motor's (***motor***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *MotorMemorySet* successfully copies data from application memory to Motor memory

See Also

mpiMotorMemoryGet | mpiMotorMemory

meiMotorEncoderReset

Declaration

```
long meiMotorEncoderReset(MPIMotor motor)
```

Required Header

stdmei.h

Description

MotorEncoderReset clears the encoder fault status registers for the primary and secondary encoder associated with the motor object.

Return Values

MPIMessageOK	if *MotorEncoderReset(...)* successfully resets the motor's encoder value.

See Also

meiMotorMultiTurnReset

Declaration

```
long meiMotorMultiTurnReset(MPIMotor    motor);
```

Required Header

stdmei.h

Description

MotorMultiTurnReset clears the SynqNet drive multi-turn data for absolute type encoders. This is only needed when configuring the zero location for an absolute encoder or when the absolute encoder's battery is replaced. meiMotorMultiTurnReset is an offline operation. Make sure all motors (amp enables) are disabled before executing a multi-turn reset. The SynqNet network may be shutdown (dropped from SYNQ mode) due to specific drive limitations.

Not all SynqNet drives support or require this feature. Please see the drive manufacturer's documentation for details.

motor	a handle to the Motor object

Return Values

MPIMessageOK	if *MotorMultiTurnReset* successfully ...

See Also

mpiMotorControl

Declaration

```
const MPIControl mpiMotorControl(MPIMotor motor)
```

Required Header

stdmpi.h

Description

MotorControl returns a handle to the Control object with which the motor is associated.

motor	a handle to the Motor object

Return Values

MPIControl	handle to a Control object
MPIHandleVOID	if motor is invalid

See Also

mpiMotorCreate | mpiControlCreate

mpiMotorFilterMapGet

Declaration

```
long mpiMotorFilterMapGet(MPIMotor     motor,
                          MPIObjectMap *filtermap)
```

Required Header stdmpi.h

Description **MotorFilterMapGet** gets the object map of the Filters [that are associated with a Motor (***motor***)] and writes it into the structure pointed to by ***filtermap.***

Return Values	
MPIMessageOK	if *MotorFilterMapGet* successfully writes the Filter's object map (associated with a Motor) to the structure

See Also mpiMotorFilterMapSet

mpiMotorFilterMapSet

Declaration

```
long mpiMotorFilterMapSet(MPIMotor      motor,
                          MPIObjectMap  filtermap)
```

Required Header stdmpi.h

Description **MotorFilterMapSet** sets the Filters [that are associated with a Motor (***motor***)], using data from the object map specified by ***filtermap***.

Return Values	
MPIMessageOK	if *MotorFilterMapSet* successfully sets the Filters using data from the object map

See Also mpiMotorFilterMapGet

mpiMotorNumber

Declaration

```
long mpiMotorNumber(MPIMotor motor,
                    long     *number)
```

Required Header

stdmpi.h

Description

MotorNumber writes the index of a Motor (***motor***, on the motion controller that ***motor*** is associated with) to the contents of ***number***.

Return Values	
MPIMessageOK	if *MotorNumber* successfully writes the Motor's index to the contents of ***number***

See Also

meiMotorCompareListGet

Declaration

```
long meiMotorCompareListGet(MPIMotor motor,
                            long          *compareCount,
                            long          *compareList);
```

Required Header stdmei.h

Description **MotorCompareListGet** sets ***compareCount*** to the number of compares that ***motor*** has. It sets ***compareList*** to the number of the compare object for each compare.

Return Values	
MPIMessageOK	if *motor* is a valid MPIMotor object.

See Also

meiMotorEncoderRatio

Declaration

```
long meiMotorEncoderRatio(MPIControl          control,
                          long                motorNumber,
                          long                encoderNumber,
                          MEIMotorEncoderRatio *ratio)
```

Required Header

stdmei.h

Description

MotorEncoderRatio gets encoder ratio from the XMP.

WARNING: This is a customer-specific method that is only supported with custom firmware. To inquire about using this method, please contact an MEI Applications Engineer.

Return Values

MPIMessageOK	if *meiMotorEncoderRatio* successfully gets encoder ration from the XMP.
MPIMessageARG_INVALID	if arguments are not valid

See Also

MEIMotorAmpFaults

MEIMotorAmpFaults

```
typedef struct MEIMotorAmpFaults {
    long                 count;
    long                 code[MEIMotorAmpFaultsMAX];
    MEIMotorAmpFaultMsg  message[MEIMotorAmpFaultsMAX];
} MEIMotorAmpFaults;
```

Description

The **MotorAmpFaults** structure contains the amp fault information from a SynqNet drive. The amp fault messages are drive specific. Not all drives support amp fault messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

count	The number of amp faults in the buffer.
code	An array of drive specific amp fault coded values.
message	An array of drive specific amp fault message strings.

See Also

meiMotorAmpFault | meiMotorAmpFaultClear

MEIMotorAmpFaultMsg

MEIMotorAmpFaultMsg

```
typedef char   MEIMotorAmpFaultMsg[MEIMotorAmpMsgMAX];
```

Description

MotorAmpFaultMsg defines the amp fault message string definition.

See Also

MEIMotorAmpMsgMAX

MEIMotorAmpWarnings

MEIMotorAmpWarnings

```
typedef struct MEIMotorAmpWarnings {
    long                    count;
    long                    code[MEIMotorAmpWarningsMAX];
    MEIMotorAmpWarningMsg   message[MEIMotorAmpWarningsMAX];
} MEIMotorAmpWarnings;
```

Description

The **MotorAmpWarnings** structure contains the amp warning information from a SynqNet drive. The amp warning messages are drive specific. Not all drives support amp warning messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

count	The number of amp faults in the buffer.
code	An array of drive specific amp fault coded values.
message	An array of drive specific amp fault message strings.

See Also

meiMotorAmpWarning | meiMotorAmpWarningClear

MEIMotorAmpWarningsMsg

MEIMotorAmpWarningsMsg

```
typedef  char  MEIMotorAmpWarningsMsg[MEIMotorAmpMsgMAX];
```

Description

MotorAmpWarningsMsg defines the maximum number of amp warning messages per motor.

See Also

MPIMotorBrake

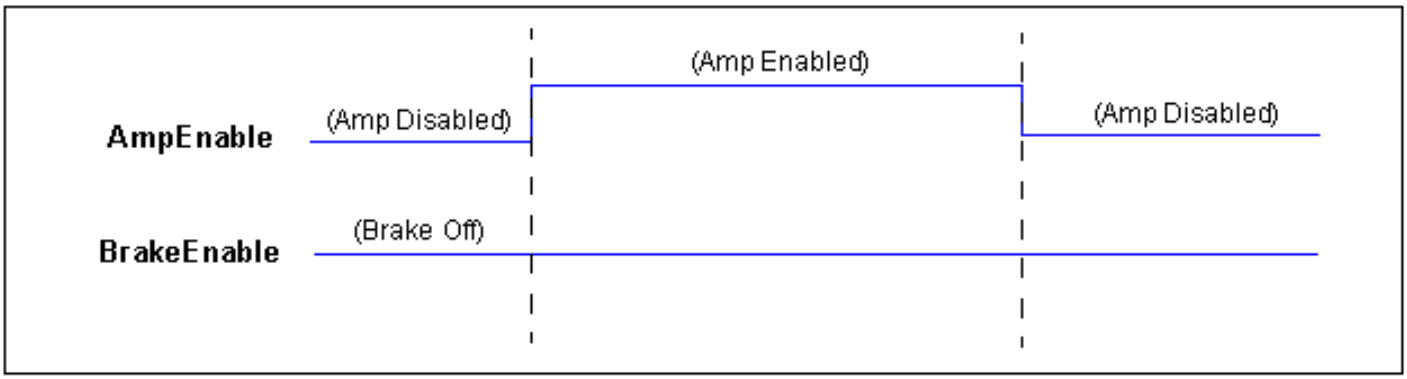
MPIMotorBrake

```
typedef struct MPIMotorBrake {
    MPIMotorBrakeMode   mode;
    float               applyDelay;
    float               releaseDelay;
} MPIMotorBrake;
```

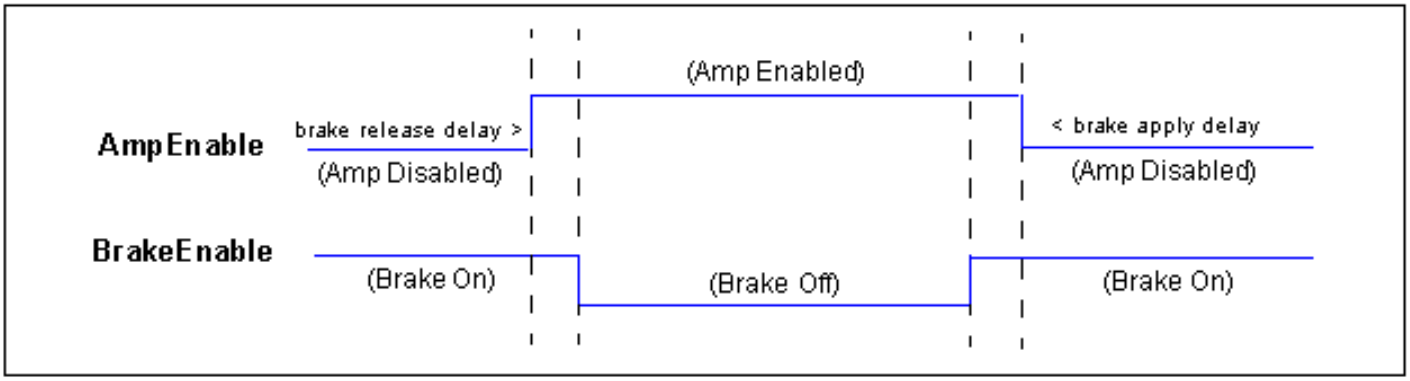
Description

The **MotorBrake** structure specifies the configuration for a motor's dedicated brake logic. Each motor object has a dedicated brake output. The controller enables/disables the brake depending on the amp enable state and the brake configuration. When the amp enable is disabled, the brake is set to an active state. When the amp enable is enabled, the brake is set to an inactive state.

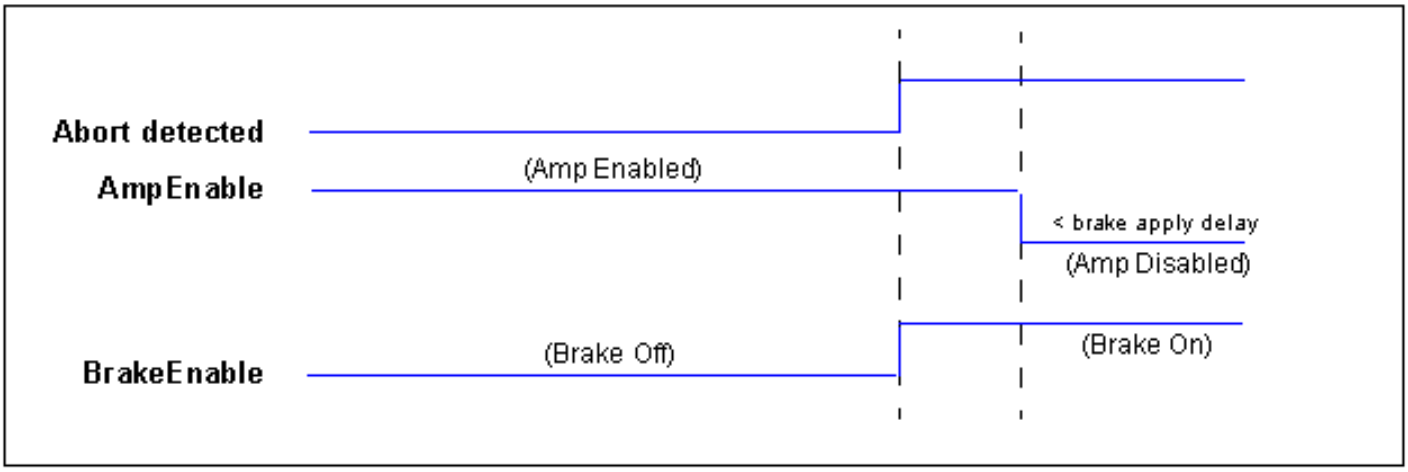
See the diagrams below for the brake logic details:



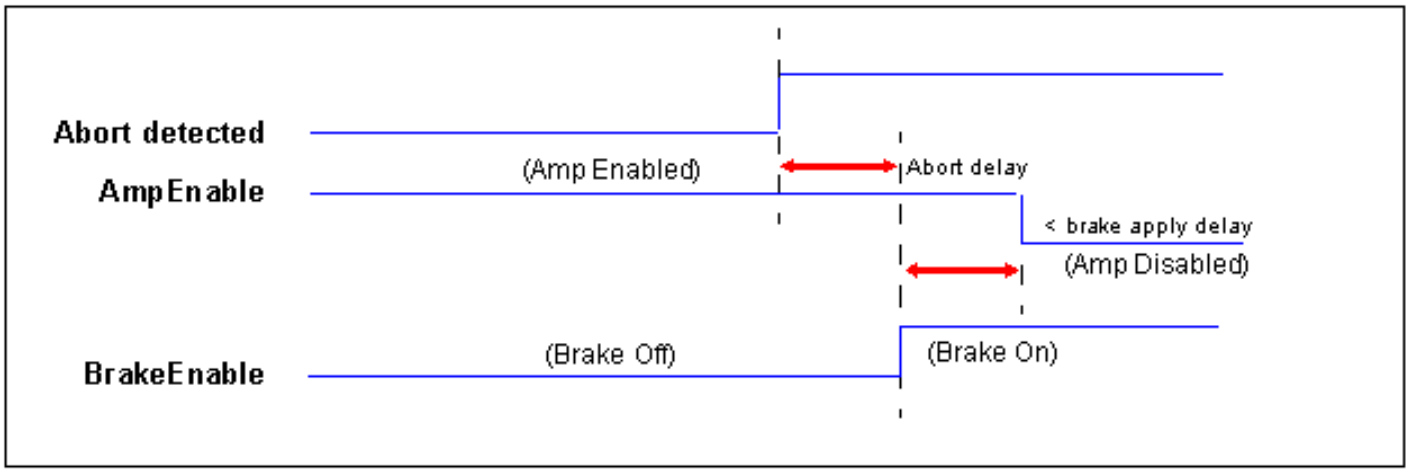
Case 1. Amp On/Off with NO Brake



Case 2. Amp On/Off with Brake



Case 3. Amp Off after ABORT Detection (in system with NO abort delay)



Case 4. Amp Off after ABORT Detection (in system WITH abort delay)

mode	An enumerated brake mode. See MPIMotorBrakeMode.
applyDelay	The time between when the brake is active and the amp enable is disabled. The units are in seconds.
releaseDelay	The time between when the amp enable is enabled and the brake is inactive. The units are in seconds.

See Also

mpiMotorConfigGet | mpiMotorConfigSet | MPIMotorConfig | MEIMotorDedicatedOut

MPIMotorBrakeMode

MPIMotorBrakeMode

```
typedef enum{
    MPIMotorBrakeModeNONE,
    MPIMotorBrakeModeDELAY,

} MPIMotorBrakeMode;
```

Description

MotorBrakeMode is an enumeration of modes for the dedicated brake signal.

MPIMotorBrakeModeNONE	Brake feature is disabled.
MPIMotorBrakeModeDELAY	Brake is enabled/disabled with specified delays.

See Also

MPIMotorBrake | MPIMotorConfig

MPIMotorConfig / MEIMotorConfig

MPIMotorConfig

```
typedef struct MPIMotorConfig {
    MPIMotorType      type;

    /* Event configuration, ordered by MPIEventType */
    MPIMotorEventConfig    event[MPIEventTypeMOTOR_LAST];

    float           abortDelay;
    float           enableDelay;
    MPIMotorBrake   brake;

    MPIObjectMap    filterMap;

} MPIMotorConfig;
```

Description

event	Structure to configure various Motor Events. See MPIMotorEventConfig description.
abortDelay	Sets time value, in seconds, to delay Abort action after Event has occured.
enableDelay	Sets time value, in seconds, to delay Enabling of the amplifier after commanded.
brake	Configures the dedicated brake logic. See MPIMotorBrake.
filterMap	Get/Set a map of Filter Objects to which the Motor is mapped. Default mapping is Filter 0 to Motor 0, Fitler 1 to Motor 1, etc. See also MPIObjectMap description in Object section.

MEIMotorConfig

```
typedef struct MEIMotorConfig {
    MEIMotorEncoder          Encoder[MEIXmpMotorEncoders];
    MEIMotorStatusOutput     StatusOutput;

    MEIMotorIoConfig         Io[MEIMotorIoConfigIndexLAST];

    MEIMotorFaultConfig      faultConfig;

    MEIMotorStepper          Stepper;
    MEIMotorDacConfig        Dac;

    MEIXmpCommutationBlock   Commutation;
                               /* read-only from field Theta to end */
```

MEIMotorDacConfig

MEIMotorDacConfig

```
typedef struct MEIMotorDacConfig {
    MEIXmpDACPhase              Phase;
    MEIMotorDacChannelConfig    Cmd;
    MEIMotorDacChannelConfig    Aux;
} MEIMotorDacConfig;
```

Description

MotorDacConfig is a structure that includes Command and Auxiliary DAC configuration for each motor.

See Also

meiMotorDacConfigGet | meiMotorDacConfigSet

```
    MEIXmpLimitData         Limit[MEIXmpLimitLAST];

    MPIAction               nodeFailureAction;
    MPIAction               userFaultAction;
                              /* see MEISqNodeConfigUserFault{} structure */
    MEIMotorDisableAction   disableAction;

} MEIMotorConfig;
```

Description

Encoder	Structure to configure Motor Encoder type and parameters
StatusOutput	A structure to configure a motor's digital outputs to monitor axis status bits. Requires custom firmware.
Io	An array of motor I/O configuration structures.
faultConfig	A structure to configure a motor's fault bits. Support for motor fault bits is node/drive specific.
Stepper	Structure to configure Motor Stepper parameters. See MEIMotorStepper description.
Dac	Structure that includes Command and Auxiliary DAC configuration for each motor. See MEIMotorDacConfig description.
Commutation	A structure to configure controller sinusoidal commutation. This structure is controller specific. Please see the Sinusoidal Commutation Application Note for more details.
limit	Structure used to configure custom motor limits and events.
nodeFailureAction	Action applied to the motor when a Node failure occurs.
userFaultAction	Action applied to the motor when a User Fault occurs.
disableAction	Used to configure the controller to set the command position equal to the actual position while the motor is disabled.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorDacChannelConfig

MEIMotorDacChannelConfig

```
typedef struct MEIMotorDacChannelConfig {
    float               Offset; /* volts */
    float               Scale;
    MEIXmpDACInputType  InputType;
    MEIXmpGenericValue  *Input;
} MEIMotorDacChannelConfig;
```

Description

MotorDacChannelConfig is a structure used to configure the DAC settings.

Offset	Set DAC Offset value. Valid values range from -10 Volts to +10 Volts.
Scale	Multiplier for the Dac channel. Default value is 1.0.
InputType	An enumerated value to define whether the Dac channel input is a float or a long. The input type is reserved for special or custom configurations.
Input	A pointer to a MEIXmpGenericValue, which is a union of a long and float value. The input pointer is reserved for special or custom configurations.

See Also

MEIMotorDacChannelStatus

MEIMotorDacChannelStatus

```
typedef struct MEIMotorDacChannelStatus {
    float       level;  /* volts */
} MEIMotorDacChannelStatus;
```

Description

MotorDacChannelStatus is a structure that returns the DAC output value.

level	*level* reflects the DAC output value. Valid values range from -10 Volts to +10 Volts.

See Also

MEIMotorDacStatus

MEIMotorDacStatus

```
typedef struct MEIMotorDacStatus {
    MEIMotorDacChannelStatus    cmd;
    MEIMotorDacChannelStatus    aux;
} MEIMotorDacStatus;
```

Description

MotorDacStatus is a structure that returns the Status for both Command and Auxiliary DACs. It is used to read the ***cmd*** and ***aux*** DAC level (in volts) from the controller.

See Also

MEIMotorDedicatedIn

MEIMotorDedicatedIn

```
typedef enum {
    MEIMotorDedicatedInAMP_FAULT       =  MEIXmpMotorDedicatedFlagsMaskAMP_FAULT,  /*
bit 0  */
    MEIMotorDedicatedInBRAKE_APPLIED   =  MEIXmpMotorDedicatedFlagsMaskBRAKE_ON,   /*
bit 1  */
    MEIMotorDedicatedInHOME            =  MEIXmpMotorDedicatedFlagsMaskHOME,       /*
bit 2  */
    MEIMotorDedicatedInLIMIT_HW_POS    =  MEIXmpMotorDedicatedFlagsMaskPOS_LIMIT,  /*
bit 3  */
    MEIMotorDedicatedInLIMIT_HW_NEG    =  MEIXmpMotorDedicatedFlagsMaskNEG_LIMIT,  /*
bit 4  */
    MEIMotorDedicatedInINDEX_PRIMARY   =  MEIXmpMotorDedicatedFlagsMaskENC_INDEX0, /*
bit 5  */
    MEIMotorDedicatedInFEEDBACK_FAULT  =  MEIXmpMotorDedicatedFlagsMaskENC_FAULT,  /*
bit 6  */
    MEIMotorDedicatedInCAPTURED        =  MEIXmpMotorDedicatedFlagsMaskCAPTURED,   /*
bit 7  */
    MEIMotorDedicatedInHALL_A          =  MEIXmpMotorDedicatedFlagsMaskHALL_A,     /*
bit 8  */
    MEIMotorDedicatedInHALL_B          =  MEIXmpMotorDedicatedFlagsMaskHALL_B,     /*
bit 9  */
    MEIMotorDedicatedInHALL_C          =  MEIXmpMotorDedicatedFlagsMaskHALL_C,     /*
bit 10 */
    MEIMotorDedicatedInAMP_ACTIVE      =  MEIXmpMotorDedicatedFlagsMaskAMP_ACTIVE, /*
bit 11 */
    MEIMotorDedicatedInINDEX_SECONDARY =  MEIXmpMotorDedicatedFlagsMaskENC_INDEX1,
/* bit 12 */
    MEIMotorDedicatedInWARNING         =  MEIXmpMotorDedicatedFlagsMaskWARNING,
/* bit 13 */
    MEIMotorDedicatedInDRIVE_STATUS_9  =  MEIXmpMotorDedicatedFlagsMaskDRIVE_STATUS9,
/* bit 14 */
    MEIMotorDedicatedInDRIVE_STATUS_10 =
MEIXmpMotorDedicatedFlagsMaskDRIVE_STATUS10, /* bit 15 */
} MEIMotorDedicatedIn;
```

Description

MotorDedicatedIn is an enumeration of bit masks for the motor's dedicated inputs. The support for dedicated inputs is node/drive specific. See the node/drive manufacturer's documentation for details.

MEIMotorDedicatedInAMP_FAULT	Generated by the masked motor fault bits. Active when one or more masked motor faults bits are active. See MEIMotorFaultConfig.
MEIMotorDedicatedInBRAKE_APPLIED	Mechanical brake state.
MEIMotorDedicatedInHOME	Position calibration sensor.
MEIMotorDedicatedInLIMIT_HW_POS	Hardware limit for the positive direction.
MEIMotorDedicatedInLIMIT_HW_NEG	Hardware limit for the negative direction.
MEIMotorDedicatedInINDEX_PRIMARY	Primary encoder index input signal.
MEIMotorDedicatedInFEEDBACK_FAULT	Position feedback status. TRUE when position feedback fails, FALSE when operating properly.
MEIMotorDedicatedInCAPTURED	Currently not supported.
MEIMotorDedicatedInHALL_A	Reflects the state of Hall Sensor A
MEIMotorDedicatedInHALL_B	Reflects the state of Hall Sensor B
MEIMotorDedicatedInHALL_C	Reflects the state of Hall Sensor C
MEIMotorDedicatedInAMP_ACTIVE	A bit set by the drive that indicates the amplifier's state. 1 = Amplifier is closing the current loop and the motor winding are energized. 0 = Amplifier is not closing the current loop and the motor windings are not energized. Support for this bit varies depending on the drive type.
MEIMotorDedicatedInINDEX_SECONDARY	Secondary encoder index input signal.
MEIMotorDedicatedInWARNING	Drive warning state. 1 = drive warning status bit is active and warning message is available 0 = drive warning status bit is not active. Support for this bit varies depending on the drive type.
MEIMotorDedicatedInDRIVE_STATUS_9	State of bit 9 in the SynqNet drive specific status register.
MEIMotorDedicatedInDRIVE_STATUS_10	State of bit 10 in the SynqNet drive specific status register.

See Also

mpiMotorIoGet

MEIMotorDedicatedOut

MEIMotorDedicatedOut

```
typedef enum {
    MEIMotorDedicatedOutAMP_ENABLE     = MEIXmpMotorDedicatedFlagsMaskAMP_ENABLE,
/* bit 0 */
    MEIMotorDedicatedOutBRAKE_RELEASE  = MEIXmpMotorDedicatedFlagsMaskBRAKE_RELEASE,
/* bit 1 */
} MEIMotorDedicatedOut;
```

Description

MotorDedicatedOut is an enumeration of bit masks for the motor's dedicated outputs. The support for dedicated outputs is node/drive specific. See the node/drive manufacturer's documentation for details.

MEIMotorDedicatedOutAMP_ENABLE	Enable/disable drive or amplifier. Drive is enabled when TRUE, disabled when FALSE.
MEIMotorDedicatedOutBRAKE_RELEASE	Enable/disable mechanical brake. Brake is released (motor shaft is free) when TRUE, engaged when FALSE.

See Also

mpiMotorIoGet | mpiMotorIoSet

MEIMotorDisableAction

MEIMotorDisableAction

```
typedef enum MEIMotorDisableAction {
    MEIMotorDisableActionNONE,
    MEIMotorDisableActionCMD_EQ_ACT,
} MEIMotorDisableAction;
```

Description

MotorDisableAction is an enumeration of controller actions to be applied when the Amp Enable is disabled. This feature can be configured by calling mpiMotorConfigSet(...) with the disableAction element in the MEIMotorConfig structure set to one of the values defined in the MEIMotorDisableAction enumeration.

The default configuration is MEIMotorDisableActionCMD_EQ_ACT. This configuration applies some safety features which eliminate motor jumps when the Amp Enable is enabled. This is the recommended configuration for all servo motor types.

The CMD_EQ_ACT action does not operate with stepper motors. If the motor type is a stepper, make sure to set the disable action to NONE. The pulse output is based on the command position, so if the controller sets the command position equal to the actual position during motion, it will cause very unusual motion profiles.

MEIMotorDisableActionNONE	No action. When the Amp Enable is disabled (or enabled), the controller continues to calculate and apply the torque demand output value.
MEIMotorDisableActionCMD_EQ_ACT	Command position equals actual position action (default). When the Amp Enable is disabled, the controller will: 1) Disable the servo loop output (except for the offset). 2) Set the command position equal to the actual position every sample. 3) Clear the integrator error. When the Amp Enable is enabled, the controller will operate the servo loop normally.

See Also

mpiMotorConfigSet | MEIMotorConfig | MEIMotorDisableAction

MPIMotorEncoder / MEIMotorEncoder

MPIMotorEncoder

```
typedef enum {
    MPIMotorEncoderPRIMARY,
    MPIMotorEncoderSECONDARY,
} MPIMotorEncoder;
```

Description

MotorEncoder is an enumeration of encoder feedback inputs for a motor.

MPIMotorEncoderPRIMARY	The first encoder feedback for a motor.
MPIMotorEncoderSECONDARY	The second encoder feedback for a motor.

See Also

MEIMotorEncoder

```
typedef struct MEIMotorEncoder {
    MEIMotorEncoderType             type;
    long                            encoderPhase;
                                    /* 0 => normal, else reversed */
    long                            filterDisable;
                                    /* 0 => quad filter enabled,
                                    else not enabled */
    long                            countsPerRev;
    MEIMotorEncoderRatio            ratio;
    MEIMotorEncoderReverseModulo    reverseModulo;
} MEIMotorEncoder;
```

Description

type	The type of feedback being used by the motor.
encoderPhase	Controls the direction of the encoder counts (only applicable with quadrature encoder feedback).
filterDisable	Enables/Disables the use of filters on the encoder signal.

countsPerRev	The number of encoder counts per one revolution of the motor's shaft.
ratio	Custom firmware required. Encoder feedback ratio for scaling.
reverseModulo	Custom firmware required. Reverse modulo value for the encoder to handle rollover.

See Also

mpiMotorCreate

MPIMotorEncoderFault

MPIMotorEncoderFault

```
typedef enum {
    MPIMotorEncoderFaultPRIMARY,
    MPIMotorEncoderFaultSECONDARY,
    MPIMotorEncoderFaultPRIMARY_OR_SECONDARY,
} MPIMotorEncoderFault;
```

Description

MotorEncoderFault is an enumeration of encoder fault sources for motor encoder fault events. Each motor object supports a primary and secondary encoder. The hardware may or may not support a secondary encoder.

MPIMotorEncoderFaultPRIMARY	Sets the Motor Event (Encoder Fault) to trigger from the primary encoder.
MPIMotorEncoderFaultSECONDARY	Sets the Motor Event (Encoder Fault) to trigger from the secondary encoder.
MPIMotorEncoderFaultPRIMARY_OR_SECONDARY	Sets the Motor Event (Encoder Fault) to trigger from either the primary or secondary encoder.

See Also

MPIMotorEventConfig | MPIMotorEventTrigger

MPIMotorEncoderFaultMask

MPIMotorEncoderFaultMask

```
typedef enum {
     MPIMotorEncoderFaultMaskNONE,
     MPIMotorEncoderFaultMaskBW_DET,
     MPIMotorEncoderFaultMaskILL_DET,
     MPIMotorEncoderFaultMaskABS_ERR,
     MPIMotorEncoderFaultMaskALL
} MPIMotorEncoderFaultMask;
```

Description

MotorEncoderFaultMask is an enumeration to mask bits from MPIMotorEncoderFault register.

MPIMotorEncoderFaultMaskBW_DET	Mask for Broken Wire detection
MPIMotorEncoderFaultMaskILL_DET	Mask for Illegal State detection
MPIMotorEncoderFaultMaskABS_ERR	Mask for Absolute Encoder Error

See Also

MEIMotorEncoderRatio

MEIMotorEncoderRatio

```
typedef struct MEIMotorEncoderRatio {
    long    A;    /* denominator */
    long    B;    /* numerator */
} MEIMotorEncoderRatio;
```

Description

MotorEncoderRatio is used to set the Encoder ratio. Ratio is programmed into the firmware via a denominator (A) and a numerator (B). B/A = ratio.

NOTE: Custom Firmware is required to support this data type.

A	the denominator.
B	the numerator.

See Also

MEIMotorEncoderReverseModulo

MEIMotorEncoderReverseModulo

```
typedef struct MEIMotorEncoderReverseModulo {
    long    *Ptr;   /* XMP Address */
    long    Rollover;
} MEIMotorEncoderReverseModulo;
```

Description

MotorEncoderReverseModulo is used to program the firmware to calculate a 32-bit encoder position based off of the data that is pointed to by ***Ptr***. Each sample, the firmware calculates a delta from the data that is pointed to by ***Ptr*** and the ***Rollover*** value. This delta is added to a 32-bit counter to create a 32-bit position.

NOTE: Custom Firmware is required to support this data type.

***Ptr**	a pointer to the XMP address.
Rollover	the value is the resolution of the incoming data pointed to by *Ptr.

See Also

MEIMotorEncoderType

MEIMotorEncoderType

```
typedef enum MEIMotorEncoderType{
    MEIMotorEncoderTypeQUAD_AB,
    MEIMotorEncoderTypeDRIVE,
    MEIMotorEncoderTypeSSI,
} MEIMotorEncoderType;
```

Description

MotorEncoderType is the type of encoder being used for feedback. The encoder type is drive specific. Therefore, an appropriate default value should be set by the MPI.

MEIMotorEncoderTypeQUAD_AB	Quadrature encoder feedback.
MEIMotorEncoderTypeDRIVE	Drive memory interface feedback. This includes all feedback types supported by the drive.
MEIMotorEncoderTypeSSI	Synchronous Serial Interface (SSI) absolute encoder feedback.

See Also

MPIMotorEventConfig / MEIMotorEventConfig

MPIMotorEventConfig

```
typedef struct MPIMotorEventConfig {
    MPIAction            action;
    MPIMotorEventTrigger trigger;
    long                 direction;
    float                duration;     /* seconds */
} MPIMotorEventConfig;
```

Description

MotorEventConfig is a structure used to configure Motor Events.

direction	for Hardware and Software limits, enabling "direction" requires the direction of motion to be in direction of Limit.
duration	time that Limit (e.g. Home, Pos. and Neg. Limits, User Limit) must be asserted before Event is generated. Value in seconds. **NOTE**: The duration parameter for the home limit should be zero for standard configurations. A non-zero value will result in the home limit being missed.

MEIMotorEventConfig

```
typedef MEIXmpLimitData MEIMotorEventConfig;
typedef struct {
    MEIXmpLimitCondition    Condition[MEIXmpLimitConditions];
    MEIXmpStatus            Status;
    MEIXmpLogic             Logic;
    MEIXmpLimitOutput       Output;
    long                    Count;
    long                    State;
} MEIXmpLimitData;
```

Description

condition	is a structure that configures the conditioanl statements evaluated to generate a Limit Event. Each limit may have up to two conditions (MEIXmpLimitConditions = 2). This structure is described in further detail in App Note 215.
status	an enum that defines what actions the XMP will take when a user limit evaluates TRUE. Always set Status to at least MEIXmpStatusLIMIT to notify the motor object that a limit has occurred. Valid **Status** values are listed in the Values of Status table below.
logic	an enum that sets the logic applied between the two condition block outputs, Condition[0] and Condition[1]. Valid **Logic** values are listed in the Values of Logic table below.

output	is a structure that allows specific action to be taken when the Limit Event is generated. In addition to generatign a Motion Action, a Limit can write to any other valid XMP Firmware register defined in the *OutputPtr with value described by AndMask and OrMask. This structure is described in further detail in App Note 215.
count	For internal use only. The MPI method, mpiMotorEventConfigSet(...) will not write these values.
state	For internal use only. The MPI method, mpiMotorEventConfigSet(...) will not write these values.

Values of Status	Action to be taken
MEIXmpStatusLIMIT	None
MEIXmpStatusLIMIT | MEIXmpStatusPAUSE	Axes attached to the motor will be Paused
MEIXmpStatusLIMIT | MEIXmpStatusSTOP	Axes attached to the motor will be Stopped
MEIXmpStatusLIMIT | MEIXmpStatusABORT	Axes attached to the motor will be Aborted
MEIXmpStatusLIMIT | MEIXmpStatusESTOP	Axes attached to the motor will be E-Stopped
MEIXmpStatusLIMIT | MEIXmpStatusESTOP_ABORT	Axes attached to the motor will be E-Stopped and Aborted

Values of Logic	Evaluates	Motor object notified that a limit has occurred if...
MEIXmpLogicNEVER	Nothing	No event is generated
MEIXmpLogicSINGLE	Condition[0]	Condition[0] = = TRUE
MEIXmpLogicOR	Condition[0], Condition[1]	(Condition[0] || Condition[1]) = = TRUE
MEIXmpLogicAND	Condition[0], Condition[1]	(Condition[0] && Condition[1]) = = TRUE
other **MEIXmpLogic** enums	For internal use only.	

See Also

Special Note: MPIMotorEventConfig and Motor Limit Configuration
See App Note 215 for a more in-depth breakdown of this structure.
mpiMotorEventConfigGet | mpiMotorEventConfigSet | MPIEncoderFault

MPIMotorEventTrigger

MPIMotorEventTrigger

```
typedef union {
    long                 polarity; /* 0 => active low, else active high */
    long                 position; /* MPIEventTypeLIMIT_SW_[POS|NEG] */
    float                error;    /* MPIEventTypeLIMIT_ERROR */
    MPIMotorEncoderFault encoder;  /* MPIEventTypeENCODER_FAULT */
} MPIMotorEventTrigger;
```

Description

polarity	Configures the polarity for the motor event trigger. If polarity = 0 (FALSE), the event will trigger on an active low signal. If polarity = 1 (TRUE), the event will trigger on an active high signal.
position	Configures the positive and negative software position limits. The controller monitors the actual position and compares it to the positive and negative software position limits. If the positive limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_POS event. If the negative limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_NEG event.
error	Configures the position error limit. The controller calculates the position error each sample period: (command position - actual position) = position error. If the position error exceeds the range of the specified error limit, the controller will generate a MPIEventTypeLIMIT_ERROR.
encoder	Configures the controller to monitor the primary or secondary for faults. The encoder should be set to one of the values defined by the MPIMotorEncoderFault{...} enumeration. When configured, if the primary, secondary, or either encoder generates a fault, the controller will generate the MPIEventTypeENCODER_FAULT.

See Also

MPIMotorEventConfig | MPIMotorEncoderFault | MPIEventType | MEIEventType

MEIMotorFaultBit

MEIMotorFaultBit

```
typedef enum MEIMotorFaultBit {
    MEIMotorFaultBitINVALID,
    MEIMotorFaultBitAMP_FAULT,
    MEIMotorFaultBitDRIVE_FAULT,
    MEIMotorFaultBitWATCHDOG_FAULT,
    MEIMotorFaultBitCHECKSUM_ERROR,
    MEIMotorFaultBitFEEDBACK_FAULT,
    MEIMotorFaultBitAMP_NOT_POWERED,
    MEIMotorFaultBitDRIVE_NOT_READY,
    MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY,
} MEIMotorFaultBit;
```

Description

MotorFaultBit is an enumeration of motor fault bits for the SynqNet node drive interface. The support for motor fault bits is node/drive specific. Please see the node\drive manufacturer's documentation for details.

MEIMotorFaultBitAMP_FAULT

External amp fault input pin. Indicates a fault condition in the amplifier.

MEIMotorFaultBitDRIVE_FAULT

Bit in the drive interface status register. Indicates a fault condition in the drive.

MEIMotorFaultBitWATCHDOG_FAULT

Bit in the drive interface status register. Indicates a drive failure due to a watchdog timeout. The node alternately sets/clears the watchdog each sample, the drive's processor then clears/sets the watchdog. If the drive's processor does not respond, the watchdog fault bit is set.

MEIMotorFaultBitCHECKSUM_ERROR

Bit in the drive interface status register. Indicates the data transfer (via serial or parallel memory interface) between the failed SynqNet node FPGA and drive with a checksum error.

MEIMotorFaultBitFEEDBACK_FAULT

Bit in the drive interface status register. Active when one or more of the feedback status bits is triggered. Indicates that the drive/motor position feedback system has failed.

MEIMotorFaultBitAMP_NOT_POWERED

Bit in the drive interface status register. The SynqNet drive amplifier power stage does not have sufficient voltage. The motor windings cannot be energized properly until this bit is clear.

MEIMotorFaultBitDRIVE_NOT_READY

Bit in the drive interface status register. The SynqNet drive is not ready to receive commands or servo motors. This fault indicates the drive has not completed its processor initialization or there is some other serious drive processor problem.

MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY

Bit in the drive interface status register. Active when one or more of the secondary feedback status bits is triggered. Indicates that the drive/motor auxilury position feedback system has failed.

See Also

MEIMotorFaultMask | MEIMotorFaultConfig | meiMotorStatus

MEIMotorFaultConfig

MEIMotorFaultConfig

```
typedef struct MEIMotorFaultConfig {
    MEIMotorFaultMask   faultMask; /* sets the
                                      MEIMotorDedicatedInAMP_FAULT state */
} MEIMotorFaultConfig;
```

Description

The **MotorFaultConfig** structure specifies the motor fault bit configuration.

faultMask	A mask of motor fault bits. The masked motor fault bits will be monitored by the controller's motor object. If any masked motor fault bit is active (TRUE), the motor's dedicated amp fault input (MEIMotorDedicatedInAMP_FAULT) is set active (TRUE). The support for motor fault bits is node/drive specific. During SynqNet network initialization, the SqNodeLib automatically configures the faultMask based on the node type. See the node/drive manufacturer's documentation for details.

See Also

MEIMotorFaultBit | MEIMotorFaultMask | meiMotorStatus

MEIMotorFaultMaskDRIVE_NOT_READY	Bit in the drive interface status register. The SynqNet drive is not ready to receive commands or servo motors. This fault indicates the drive has not completed its processor initialization or there is some other serious drive processor problem.
MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY	Bit in the drive interface status register. Active when one or more of the secondary feedback status bits is triggered. Indicates that the drive/motor auxilury position feedback system has failed.

See Also

MEIMotorFaultBit | MEIMotorFaultConfig | meiMotorStatus

MEIMotorFaultMask

MEIMotorFaultMask

```
  typedef enum MEIMotorFaultMask {
      MEIMotorFaultMaskAMP              = (1<< MEIMotorFaultBitAMP_FAULT),
      MEIMotorFaultMaskDRIVE            = (1<< MEIMotorFaultBitDRIVE_FAULT),
      MEIMotorFaultMaskWATCHDOG         = (1<< MEIMotorFaultBitWATCHDOG_FAULT),
      MEIMotorFaultMaskCHECKSUM         = (1<< MEIMotorFaultBitCHECKSUM_ERROR),
      MEIMotorFaultMaskFEEDBACK         = (1<< MEIMotorFaultBitFEEDBACK_FAULT),
      MEIMotorFaultMaskAMP_NOT_POWERED = (1<< MEIMotorFaultBitAMP_NOT_POWERED),
      MEIMotorFaultMaskDRIVE_NOT_READY = (1<< MEIMotorFaultBitDRIVE_NOT_READY),
      MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY = (1<<
MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY),
  } MEIMotorFaultMask;
```

Description

MotorFaultMask is an enumeration of motor fault bit masks for the SynqNet node drive interface. The support for motor fault bits is node/drive specific. Please see the node\drive manufacturer's documentation for details.

MEIMotorFaultMaskAMP	External amp fault input pin. Indicates a fault condition in the amplifier.
MEIMotorFaultMaskDRIVE	Bit in the drive interface status register. Indicates a fault condition in the drive.
MEIMotorFaultMaskWATCHDOG	Bit in the drive interface status register. Indicates a drive failure due to a watchdog timeout. The node alternately sets/clears the watchdog each sample, the drive's processor then clears/sets the watchdog. If the drive's processor does not respond, the watchdog fault bit is set.
MEIMotorFaultMaskCHECKSUM	Bit in the drive interface status register. Indicates the data transfer (via serial or parallel memory interface) between the failed SynqNet node FPGA and drive with a checksum error.
MEIMotorFaultMaskFEEDBACK	Bit in the drive interface status register. Active when one or more of the feedback status bits is triggered. Indicates that the drive/motor position feedback system has failed.
MEIMotorFaultMaskAMP_NOT_POWERED	Bit in the drive interface status register. The SynqNet drive amplifier power stage does not have sufficient voltage. The motor windings cannot be energized properly until this bit is clear.

MPIMotorFeedback

MPIMotorFeedback

```
typedef double MPIMotorFeedback[MPIMotorEncoderLAST];
```

Description

MotorFeedback is an array of doubles used to store motor feedback (both primary and secondary).

See Also

MPIMotorEncoder | mpiMotorFeedback

MEIMotorInfo

MEIMotorInfo

```
typedef struct MEIMotorInfo {
    MEIMotorInfoNodeType   nodeType;
    struct {
        long    networkNumber;
        long    nodeNumber;
        long    driveIndex;
    } sqNode;

    long    captureCount;
    long    encoderCount;
    long    probeCount;

} MEIMotorInfo;
```

Description

The **MotorInfo** structure contains static data determined during network initialization. It identifies the network, node, and drive interface associated with the motor object.

nodeType	Identifies the type of node. See MEIMotorInfoNodeType.
networkNumber	An index to the SynqNet network (0, 1, 2 etc.).
nodeNumber	An index to the node (0, 1, 2, etc.).
driveIndex	An index to the drive interface (0, 1, 2, etc.).
captureCount	Counts the number of captures for the motor.
encoderCount	Counts the number of encoders for the motor.
probeCount	Counts the number of hardware Probe engines for the motor.

See Also

meiMotorInfo

MEIMotorInfoNodeType

MEIMotorInfoNodeType

```
typedef enum MEIMotorInfoNodeType {
    MEIMotorInfoNodeTypeNONE,
    MEIMotorInfoNodeTypeSQNODE,
} MEIMotorInfoNodeType;
```

Description

MotorInfoNodeType is an enumeration of node types. It specifies the node type associated with the motor.

MEIMotorInfoNodeTypeNONE	Not a network node. The node is local to the controller.
MEIMotorInfoNodeTypeSQNODE	A SynqNet node type.

See Also

MEIMotorInfo

MPIMotorIo

MPIMotorIo

```
typedef struct MPIMotorIo {
    unsigned long       input;
    unsigned long       output;
} MPIMotorIo;
```

Description

MotorIo is a structure used to read the Motor input and set Motor output values.

input	The **input** value reflects the 32 bits of general and dedicated input values. The dedicated input values (bits 0-15) are specified by MEIMotorDedicatedIn. The general purpose input values (bits 16-31) can be specified by MEIMotorIoMask.
output	The **output** value reflects the 32 bits of general and dedicated output values. The dedicated output values (bits 0-15) are specified by MEIMotorDedicatedOut. The general purpose output values (bits 16-31) can be specified by MEIMotorIoMask.

See Also

MEIMotorDedicatedIn | MEIMotorDedicatedOut | MEIMotorIoMask

MEIMotorIoConfig

MEIMotorIoConfig

```
typedef struct MEIMotorIoConfig {
    MEIMotorIoType     Type;
} MEIMotorIoConfig;
```

Description

The **MotorIoConfig** structure specifies the configuration for the motor's digital I/O.

Type	The I/O bit function. See MEIMotorIoType.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorIoConfigIndex

MEIMotorIoConfigIndex

```
typedef enum MEIMotorIoConfigIndex {
    MEIMotorIoConfigIndex0,
    MEIMotorIoConfigIndex1,
    MEIMotorIoConfigIndex2,
    MEIMotorIoConfigIndex3,
    MEIMotorIoConfigIndex4,
    MEIMotorIoConfigIndex5,
    MEIMotorIoConfigIndex6,
    MEIMotorIoConfigIndex7,
    MEIMotorIoConfigIndex8,
    MEIMotorIoConfigIndex9,
    MEIMotorIoConfigIndex10,
    MEIMotorIoConfigIndex11,
    MEIMotorIoConfigIndex12,
    MEIMotorIoConfigIndex13,
    MEIMotorIoConfigIndex14,
    MEIMotorIoConfigIndex15,
} MEIMotorIoConfigIndex;
```

Description

MotorIoConfigIndex is an enumeration of configurable motor I/O, indexed by a number.

MEIMotorIoConfigIndex0	motor I/O number 0
MEIMotorIoConfigIndex1	motor I/O number 1
MEIMotorIoConfigIndex2	motor I/O number 2
MEIMotorIoConfigIndex3	motor I/O number 3
MEIMotorIoConfigIndex4	motor I/O number 4
MEIMotorIoConfigIndex5	motor I/O number 5
MEIMotorIoConfigIndex6	motor I/O number 6
MEIMotorIoConfigIndex7	motor I/O number 7
MEIMotorIoConfigIndex8	motor I/O number 8
MEIMotorIoConfigIndex9	motor I/O number 9
MEIMotorIoConfigIndex10	motor I/O number 10
MEIMotorIoConfigIndex11	motor I/O number 11
MEIMotorIoConfigIndex12	motor I/O number 12
MEIMotorIoConfigIndex13	motor I/O number 13

MEIMotorIoConfigIndex14	motor I/O number 14
MEIMotorIoConfigIndex15	motor I/O number 15

See Also

MEIMotorIoConfig

MEIMotorIoMask

MEIMotorIoMask

```
typedef enum {
    MEIMotorIoMask0  =  MEIXmpMotorIOMaskConfigurable0,  /* bit 16 */
    MEIMotorIoMask1  =  MEIXmpMotorIOMaskConfigurable1,  /* bit 17 */
    MEIMotorIoMask2  =  MEIXmpMotorIOMaskConfigurable2,  /* bit 18 */
    MEIMotorIoMask3  =  MEIXmpMotorIOMaskConfigurable3,  /* bit 19 */
    MEIMotorIoMask4  =  MEIXmpMotorIOMaskConfigurable4,  /* bit 20 */
    MEIMotorIoMask5  =  MEIXmpMotorIOMaskConfigurable5,  /* bit 21 */
    MEIMotorIoMask6  =  MEIXmpMotorIOMaskConfigurable6,  /* bit 22 */
    MEIMotorIoMask7  =  MEIXmpMotorIOMaskConfigurable7,  /* bit 23 */
    MEIMotorIoMask8  =  MEIXmpMotorIOMaskConfigurable8,  /* bit 24 */
    MEIMotorIoMask9  =  MEIXmpMotorIOMaskConfigurable9,  /* bit 25 */
    MEIMotorIoMask10 =  MEIXmpMotorIOMaskConfigurable10, /* bit 26 */
    MEIMotorIoMask11 =  MEIXmpMotorIOMaskConfigurable11, /* bit 27 */
    MEIMotorIoMask12 =  MEIXmpMotorIOMaskConfigurable12, /* bit 28 */
    MEIMotorIoMask13 =  MEIXmpMotorIOMaskConfigurable13, /* bit 29 */
    MEIMotorIoMask14 =  MEIXmpMotorIOMaskConfigurable14, /* bit 30 */
    MEIMotorIoMask15 =  MEIXmpMotorIOMaskConfigurable15, /* bit 31 */
} MEIMotorIoMask;
```

Description

MotorIoMask is an enumeration of bit masks for the motor's configurable I/O. The support for configurable I/O is node/drive specific. See the node/drive manufacturer's documentation for details.

MEIMotorIoMask0	motor I/O mask for bit number 0
MEIMotorIoMask1	motor I/O mask for bit number 1
MEIMotorIoMask2	motor I/O mask for bit number 2
MEIMotorIoMask3	motor I/O mask for bit number 3
MEIMotorIoMask4	motor I/O mask for bit number 4
MEIMotorIoMask5	motor I/O mask for bit number 5
MEIMotorIoMask6	motor I/O mask for bit number 6
MEIMotorIoMask7	motor I/O mask for bit number 7
MEIMotorIoMask8	motor I/O mask for bit number 8
MEIMotorIoMask9	motor I/O mask for bit number 9
MEIMotorIoMask10	motor I/O mask for bit number 10
MEIMotorIoMask11	motor I/O mask for bit number 11
MEIMotorIoMask12	motor I/O mask for bit number 12
MEIMotorIoMask13	motor I/O mask for bit number 13

MEIMotorIoMask14	motor I/O mask for bit number 14
MEIMotorIoMask15	motor I/O mask for bit number 15

See Also

mpiMotorIoGet | mpiMotorIoSet | MEIMotorIoType

MEIMotorIoType

MEIMotorIoType

```
typedef enum MEIMotorIoType {
    MEIMotorIoTypeOUTPUT,
    MEIMotorIoTypeSOURCE1,
    MEIMotorIoTypeSOURCE2,
    MEIMotorIoTypeSOURCE3,
    MEIMotorIoTypeSOURCE4,
    MEIMotorIoTypeSOURCE5,
    MEIMotorIoTypeSOURCE6,
    MEIMotorIoTypeSOURCE7,
    MEIMotorIoTypeSOURCE8,
    MEIMotorIoTypeSOURCE9,
    MEIMotorIoTypeSOURCE10,
    MEIMotorIoTypeSOURCE11,
    MEIMotorIoTypeSOURCE12,
    MEIMotorIoTypeSOURCE13,
    MEIMotorIoTypeSOURCE14,
    MEIMotorIoTypeSOURCE15,

    MEIMotorIoTypeINPUT =  0x10,
} MEIMotorIoType;
```

Description

MotorIoType is an enumeration of motor I/O functions. A SynqNet node's motor I/O may support one or more features depending on the node hardware and FPGA. MotorIoType contains the generic enumerations for all nodes. For the node specific enumerations, please see the individual SqNode modules for details.

MEIMotorIoTypeOUTPUT	discrete digital output
MEIMotorIoTypeSOURCE1	motor I/O source number 1 for node-specific feature.
MEIMotorIoTypeSOURCE2	motor I/O source number 2 for node-specific feature.
MEIMotorIoTypeSOURCE3	motor I/O source number 3 for node-specific feature.
MEIMotorIoTypeSOURCE4	motor I/O source number 4 for node-specific feature.
MEIMotorIoTypeSOURCE5	motor I/O source number 5 for node-specific feature.
MEIMotorIoTypeSOURCE6	motor I/O source number 6 for node-specific feature.
MEIMotorIoTypeSOURCE7	motor I/O source number 7 for node-specific feature.
MEIMotorIoTypeSOURCE8	motor I/O source number 8 for node-specific feature.
MEIMotorIoTypeSOURCE9	motor I/O source number 9 for node-specific feature.
MEIMotorIoTypeSOURCE10	motor I/O source number 10 for node-specific feature.
MEIMotorIoTypeSOURCE11	motor I/O source number 11 for node-specific feature.

MEIMotorIoTypeSOURCE12	motor I/O source number 12 for node-specific feature.
MEIMotorIoTypeSOURCE13	motor I/O source number 13 for node-specific feature.
MEIMotorIoTypeSOURCE14	motor I/O source number 14 for node-specific feature.
MEIMotorIoTypeSOURCE15	motor I/O source number 15 for node-specific feature.
MEIMotorIoTypeINPUT	discrete digital input

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MPIMotorMessage / MEIMotorMessage

MPIMotorMessage

```
typedef enum {
     MPIMotorMessageMOTOR_INVALID,
} MPIMotorMessage;
```

Description

MotorMessage is an enumeration of Motor error messages that can be returned by the MPI library.

MPIMotorMessageMOTOR_INVALID	The motor number is out of range. This message code is returned by mpiMotorCreate(…) if the motor number is less than zero or greater than or equal to MEIXmpMAX_Motors.

MEIMotorMessage

```
typedef enum {
    MEIMotorMessageMOTOR_NOT_ENABLED,
    MEIMotorMessageSECONDARY_ENCODER_NA,
    MEIMotorMessageHARDWARE_NOT_FOUND,
    MEIMotorMessageSTEPPER_INVALID,
    MEIMotorMessageDISABLE_ACTION_INVALID,
    MEIMotorMessagePULSE_WIDTH_INVALID,
    MEIMotorMessageFEEDBACK_REVERSAL_NA,
    MEIMotorMessageFILTER_DISABLE_NA,
} MEIMotorMessage;
```

Description

MotorMessage is an enumeration of Motor error messages that can be returned by the MPI library.

MEIMotorMessageMOTOR_NOT_ENABLED

The motor number is not active in the controller. This message code is returned by mpiMotorEventConfig(…) if the specified motor is not enabled in the controller. To correct the problem, use mpiControlConfigSet(…) to enable the motor object, by setting the motorCount to greater than the motor number. For example, to enable motor 0 to 3, set motorCount to 4.

MEIMotorMessageSECONDARY_ENCODER_NA

The motor's secondary encoder is not available. This message code is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the encoder fault trigger is configured for a secondary encoder when the hardware does not support a secondary encoder. To correct the problem, do not select the secondary encoder when configuring the encoder fault conditions.

MEIMotorMessageHARDWARE_NOT_FOUND

The motor object's hardware resource is not available. This message code is returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the node hardware for the motor is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor objects. An application should not configure a motor object if there is no mapped hardware to receive the service commands. To correct this problem, verify that all expected nodes were found. Use meiSqNodeInfo(…) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEIMotorMessageSTEPPER_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessageDISABLE_ACTION_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessagePULSE_WIDTH_INVALID

The motor object stepper pulse width is not valid. The pulse width must be no greater than 1 millisecond and no less than 100 nanoseconds.

MEIMotorMessageFEEDBACK_REVERSAL_NA

Feedback reversal is not applicable or is not supported for the feedback type specified.

MEIMotorMessageFILTER_DISABLE_NA

Disabling of filters is not applicable or is not supported for the feedback type specified.

See Also

mpiMotorCreate

MEIMotorStatus

MEIMotorStatus

```
typedef struct MEIMotorStatus {
    MEIMotorDacStatus       dac;
    MEIMotorFaultMask       faultMask;
    MEIMotorStepperStatus   stepper;
} MEIMotorStatus;
```

Description

The **MotorStatus** structure that returns the specific Motor Status registers of the controller.

dac	The status for both Command and Auxiliary DACs.
faultMask	The motor fault bit masks for the SynqNet node drive interface.
stepper	Shows the status of the pulse module. It is used for checking network synchronization with the pulse module. It can also be used to check if a pulse velocity/width error occurred.

See Also

MEIMotorStepperStatus

MEIMotorStatusOutput

MEIMotorStatusOutput

```
typedef struct MEIMotorStatusOutput {
    long           *outPtr;
    MEIXmpIOMask   ioMask[MEIXmpMotorStatusOutputs];
} MEIMotorStatusOutput;
```

Description

The **MotorStatusOutput** structure specifies which motor outputs to follow the motor status bits. This feature requires custom controller firmware.

***outPtr**	a pointer to controller memory.
ioMask	a bit mask to select the motor output signals.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorStepper

MEIMotorStepper

```
typedef struct MEIMotorStepper {
    long                 loopback;   /* TRUE = FPGA pulse feedback,
                                        FALSE = encoder feedback */
    MEIMotorStepperPulse pulseA;
    MEIMotorStepperPulse pulseB;
    double               pulseWidth; /* output pulse width  (sec),
                                        10^(-7) < pulseWidth < 10^(-3) */
} MEIMotorStepper;
```

Description

MotorStepper is a structure used to configure Stepper Motor parameters.

To use pulse outputs, you will need to:

1. Configure the motor type for STEPPER.
2. See the motor's stepper pulse width. Make sure it meets the drive's requirments and is not smaller than 2x the minimum pulse period.
3. Enable the motor's stepper loopback if there is no encoder feedback.
4. Configure the motor's stepper pulseA and pulseB types for STEP, DIR, CW, CCW, QUADA, or QUADB.
5. Select the motor I/O's config type for pulseA and pulseB. This will route the pulseA/B signals to the node's digital outputs.

loopback	enables Step Loopback feature. When enabled, step output pulses counted to generate Actual Position. When disabled, external feedback device is required to generate actual position.
pulseA	Structure used to configure Step Pulse A.
pulseB	Structure used to configure Step Pulse B.
pulseWidth	sets width of Step pulse. Valid values range from a minimum width of 0.1 usec to maximum width of 25.5 usec.

See Also

MEIMotorStepperPulse | MEIMotorStepperPulseType

MEIMotorStepperPulse

MEIMotorStepperPulse

```
typedef struct MEIMotorStepperPulse {
    MEIMotorStepperPulseType    type;
    long                        invert;
} MEIMotorStepperPulse;
```

Description

MotorStepperPulse is a

To use pulse outputs, you will need to:

1. Configure the motor type for STEPPER.
2. See the motor's stepper pulse width. Make sure it meets the drive's requirments and is not smaller than 2x the minimum pulse period.
3. Enable the motor's stepper loopback if there is no encoder feedback.
4. Configure the motor's stepper pulseA and pulseB types for STEP, DIR, CW, CCW, QUADA, or QUADB.
5. Select the motor I/O's config type for pulseA and pulseB. This will route the pulseA/B signals to the node's digital outputs.

type	Configures the stepper motor's pulse type.
invert	If set to TRUE the actual Pulse output will be inverted by the FPGA.

See Also

MEIMotorStepper

MEIMotorStepperPulseType

MEIMotorStepperPulseType

```
typedef enum MEIMotorStepperPulseType {
    MEIMotorStepperPulseTypeSTEP,
    MEIMotorStepperPulseTypeDIR,

    MEIMotorStepperPulseTypeCW,
    MEIMotorStepperPulseTypeCCW,

    MEIMotorStepperPulseTypeQUADA,
    MEIMotorStepperPulseTypeQUADB,

} MEIMotorStepperPulseType;
```

Description

MotorStepperPulseType is an enumeration used to specify the pulse output signal type.

MEIMotorStepperPulseTypeSTEP	This will enable the pulse output (either A or B) to generate a step output. Use it together with MEIMotorStepperPulseTypeDIR to provide a complete step interface.
MEIMotorStepperPulseTypeDIR	This will enable the pulse output (either A or B) to output the direction of the move. Use it together with MEIMotorStepperPulseTypeSTEP to provide a complete step interface.
MEIMotorStepperPulseTypeCW	This will enable the pulse output (either A or B) to output a clockwise pulse train. Used together with MEIMotorStepperPulseTypeCCW to provide a complete step interface.
MEIMotorStepperPulseTypeCCW	This will enable the pulse output (either A or B) to output a counter-clockwise pulse train. Use it together with MEIMotorStepperPulseTypeCW to provide a complete step interface.

MEIMotorStepperPulseTypeQUADA	This will enable the pulse output (either A or B) to output a quadrature signal (90 degree phase difference to MEIMotorStepperPulseTypeQUADB). Use it together with MEIMotorStepperPulseTypeQUADB to provide a complete quadrature interface.
MEIMotorStepperPulseTypeQUADB	This will enable the pulse output (either A or B) to output a quadrature signal (90 degree phase difference to MEIMotorStepperPulseTypeQUADA). Use it together with MEIMotorStepperPulseTypeQUADA to provide a complete quadrature interface.

See Also

MEIMotorStepperStatus

MEIMotorStepperStatus

```
typedef struct MEIMotorStepperStatus {
    long    pulseLockLost; /* TRUE if Pulse jitter logic has lost lock,
                              otherwise FALSE */
    long    pulseStatus;    /* TRUE if a Pulse was generated incorrectly,
                              otherwise FALSE */
} MEIMotorStepperStatus;
```

Description

MotorStepperStatus is a structure used to check for error conditions relating to the pulse module.

pulseLockLost	Indicates that the pulse module lost timing lock with the SynqNet network.
pulseStatus	Indicates that the pulse module generated a new pulse when the previous pulse was not finished. It usually indicates that the pulse width is incorrect for the desired velocity.

See Also

MEIMotorStatus | MEIMotorStepper

MPIMotorType

MPIMotorType

```
typedef enum {
    MPIMotorTypeINVALID,

    MPIMotorTypeSERVO,
    MPIMotorTypeSTEPPER,

} MPIMotorType;
```

Description

MotorType is an enumeration of valid Motor Types.

MPIMotorTypeSERVO	Motor configured as Servo
MPIMotorTypeSTEPPER	Motor configured as Stepper

See Also

mpiMotorEncoderFaultMaskBIT

Declaration

```
#define mpiMotorEncoderFaultMaskBIT(fault)      (0x1 << (fault))
```

Required Header

stdmpi.h

Description

MotorEncoderFaultMaskBIT converts the motor encoder fault into the motor encoder fault mask.

See Also

MPIMotorEncoderFault | MPIMotorEncoderFaultMask

MEIMotorAmpMsgMAX

MEIMotorAmpMsgMAX

```
#define MEIMotorAmpMsgMAX    (200) /* max chars per motor */
```

Description

MotorAmpMsgMAX defines the maximum number of characters per amp fault message.

See Also

MEIMotorAmpFaultsMAX

MEIMotorAmpFaultsMAX

```
#define MEIMotorAmpFaultsMAX    (32) /* max faults per motor */
```

Description

MotorAmpFaultsMAX defines the maximum number of amp fault messages per motor.

See Also

MEIMotorAmpWarningsMAX

MEIMotorAmpWarningsMAX

```
#define MEIMotorAmpWarningsMAX    (32) /* max Warnings per motor */
```

Description

MotorAmpFaultsMAX defines the maximum number of amp warning messages per motor.

See Also

Special Note: *MPIMotorEventConfig and Motor Limit Configuration*

Two fields, directionFlag and duration, are built into the MPIMotorEventConfig{} structure. From motor.h:

```
typedef enum {
    MPIMotorEncoderFaultMaskNONE     = 0x0,

    MPIMotorEncoderFaultMaskBW_DET   =
mpiMotorEncoderFaultMaskBIT(MPIMotorEncoderFaultBW_DET),
    MPIMotorEncoderFaultMaskILL_DET  =
mpiMotorEncoderFaultMaskBIT(MPIMotorEncoderFaultILL_DET),
    MPIMotorEncoderFaultMaskABS_ERR  =
mpiMotorEncoderFaultMaskBIT(MPIMotorEncoderFaultABS_ERR),

    MPIMotorEncoderFaultMaskALL      =
(mpiMotorEncoderFaultMaskBIT(MPIMotorEncoderFaultLAST) - 1)
} MPIMotorEncoderFaultMask;

typedef union {
    long        polarity;       /* 0 => active low, else active high */
    long        position;       /* MPIEventTypeLIMIT_SW_[POS|NEG] */
    float       error;          /* MPIEventTypeLIMIT_ERROR */
    long        mask;           /* MPIEventTypeENCODER_FAULT */
} MPIMotorEventTrigger;

typedef struct MPIMotorEventConfig {
    MPIAction               action;
    MPIMotorEventTrigger    trigger;
    long                    direction;
    float                   duration;     /* seconds */
} MPIMotorEventConfig;
```

The directionFlag field is used to configure MPIEventTypeLIMIT_HW_NEG, MPIEventTypeLIMIT_HW_POS, MPIEventTypeLIMIT_SW_NEG, and MPIEventTypeLIMIT_SW_POS. A value of TRUE for this field will force the command direction for motion to be used by the Xmp controller to qualify these limit events. A value of FALSE for this field will cause the limit event to depend on the state of the limit. If the limit has been exceeded (actual position > software positive limit, actual position < software negative limit, positive or negative hardware overtravel is TRUE) the limit event will be based direction of commanded motion, in exactly the same way as for directionFlag = TRUE. If the limit has not been exceeded the limit event and status will be based solely on the limit input (hardware limits) or actual position (software limits), ignoring the direction of commanded motion. The default status of durationFlag is FALSE (ignore direction). The directionFlag is ignored (returned FALSE from “get” methods) for case MPIEventTypeAMP_FAULT, MPIEventTypeHOME, and MPIEventTypeLIMIT_ERROR.

The duration field may used in the configuration of all MPI Motor Events. A positive value for this field will require the limit condition to exist for duration seconds before an event will occur. This field is useful in

overriding noisy limit inputs. For example, an overtravel limit with infrequent short (< 1msec) noise spikes on the limit input will ignore the noise of the limit is configured with a duration of 0.05. A spike whose duration was at least 0.05 seconds (50 milliseconds) would be required before an overtravel event would occur. The default value for duration is 0.0.

Return to MPIMotorEventConfig

Special Note: *Using mpiMotorConfigSet with Absolute Encoders*

All absolute encoder configuration through the MPI is made using mpiMotorConfigSet() calls. The below sample code demonstrates the correct way to configure the XMP for Yaskawa absolute encoders.

When using a motor's User Output:

```
returnValue =
        mpiMotorFlashConfigGet(motor,
                               NULL,
                               &motorConfig,
                               &motorConfigMEI);

        motorConfig.encoderPhase = TRUE; /* Reverse */

        /* Config User Output for SEN line */
        motorConfigMEI.UserOutInvert = TRUE;

        motorConfigMEI.Encoder[0].type = MEIXmpEncoderTypeABS_0; /* Yaskawa encoder
*/
        motorConfigMEI.Encoder[0].countsPerRev = 65536;          /* 65536 for 16-bit
                                                        encoders,131072 for 17-bit */

        returnValue =
                mpiMotorFlashConfigSet(motor,
                                       NULL,
                                       &motorConfig,
                                       &motorConfigMEI);
        msgCHECK(returnValue);

        returnValue =
                mpiMotorConfigSet(motor,
                                  &motorConfig,
                                  &motorConfigMEI);
        msgCHECK(returnValue);
```

When using a motor's Transceiver Output:

```
returnValue =
                mpiMotorFlashConfigGet(motor,
                                       NULL,
                                       &motorConfig,
                                       &motorConfigMEI);

        motorConfig.encoderPhase = TRUE; /* Reverse */

        /* Config transceiver for SEN line */
        motorConfigMEI.Transceiver[0].Config = MEIMotorTransceiverConfigOUTPUT;
        motorConfigMEI.Transceiver[0].Invert = TRUE;

        motorConfigMEI.Encoder[0].type = MEIXmpEncoderTypeABS_0; /* Yaskawa encoder
*/
```

```
        motorConfigMEI.Encoder[0].countsPerRev = 65536;          /* 65536 for 16-bit
                                                                 encoders,  131072 for 17-bit
*/
returnValue =
                mpiMotorFlashConfigSet(motor,
                                       NULL,
                                       &motorConfig,
                                       &motorConfigMEI);
        msgCHECK(returnValue);

        returnValue =
                mpiMotorConfigSet(motor,
                                          &motorConfig,
                                          &motorConfigMEI);
```

In the above sample code, the steps for configuration are:

1. Choose a transceiver, or User Opto, to be used for the encoders SEN line. The only restriction is that this transceiver must be on the same controller as the absolute encoder (not necessarily the same Motion Block).
2. Get the current motor configuration from flash memory.
3. Configure the encoder phase for the absolute encoder to Reverse.
4. For a User Opto, configure UserOutInvert to be TRUE. When using a transceiver, configure for Output, and Inverted.
5. Configure the encoder type and counts per revolution.
6. Save the current motor configuration from flash memory.

Once configured, the initialization of all axes associated with the motors having absolute encoders is automatic at power up or reset. The SEN line is toggled and the origin and command position are calculated and set from the absolute data sent by the drive.

IMPORTANT NOTE:

The drive must be powered but should not be enabled.

Determining the countPerRev Parameter

The **magnitude** countsPerRev parameter is detemined by the number of encoder counts (after quadrature) for one revolution of the motor. The **sign** of the countsPerRev is derimined by the direction for positive rotation for the motor. For Yaskawa drives this is determined by the drive parameter P000.0. P000.0 = 0 (“Standard Rotation”, factory default setting) will cause the motor to move in a counter-clockwise (CCW) direction for positive increases in encoder counts. For Standard Rotation (Pn000.0 = 0) the countPerRev parameter should be positive.

P000.0 = 1 (“Reverse Rotation”) will cause the motor to move in a clockwise (CCW) direction for positive increases in encoder counts. For Reverse Rotation the countsPerRev parameter should be negative.

For example the following code would be used for a drive congfigured for Standard Rotation where the number of counts for one revolution of the motor shaft is 8,192:

motorConfigMEI.Encoder[0].countsPerRev = 8192;

If the same drive were configured for Reverse Rotation the code would be:

motorConfigMEI.Encoder[0].countsPerRev = -8192;

For both Standard and Reverse Rotation the encoderPhase parameter should be TRUE (encoder reversed).

Working with Absolute Encoders

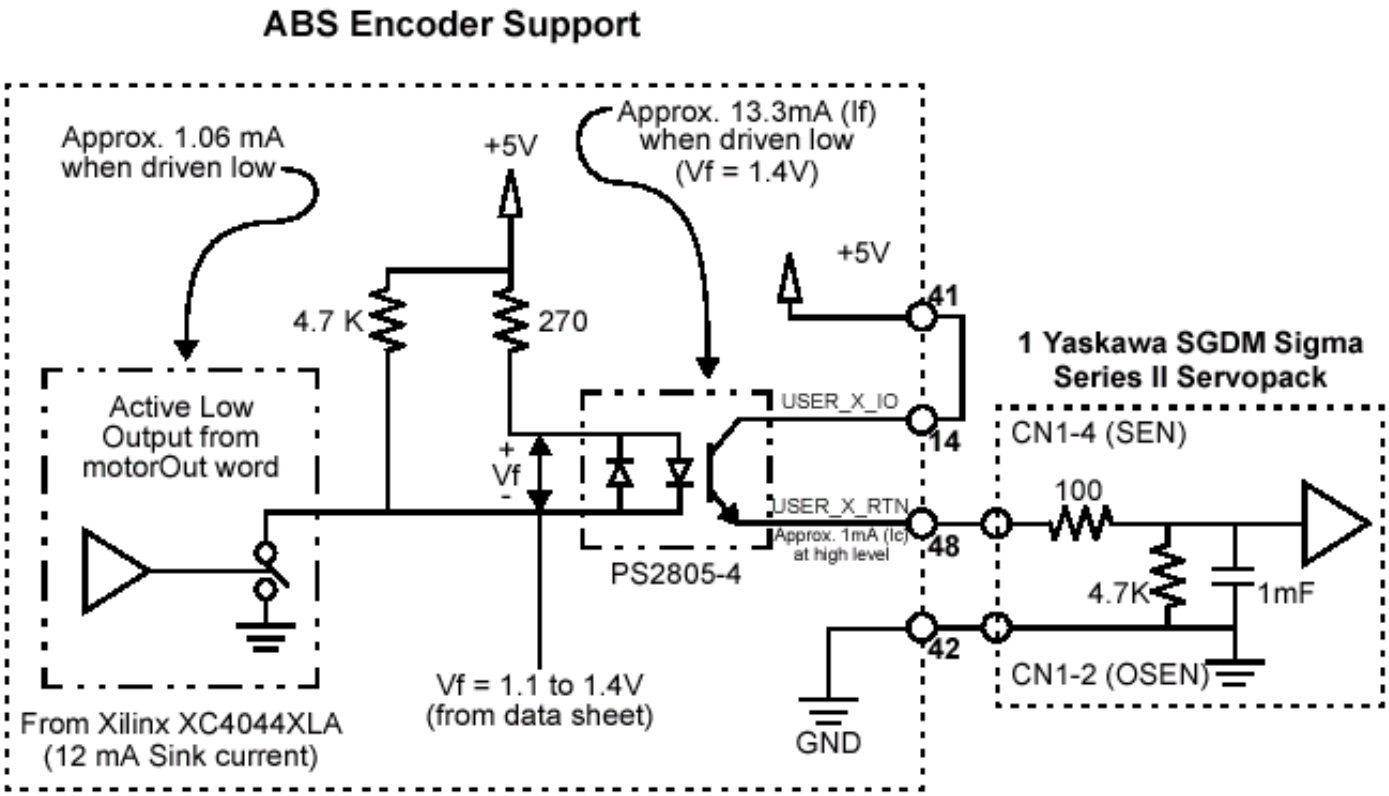
Currently, only absolute motor encoders of Yaskawa SGDM Sigma Series II Servopacks are supported by the MPI. Custom firmware is needed in order to use these absolute encoders. Contact MEI for more information.

The encoders are automatically interrogated at power-up or after a controller reset. Axis Origin and Command Positions are set to correctly reflect the absolute position of the motor with no position error initially. The absolute position is the position within a single revolution.

The encoder interrogation is controlled by a SEN signal (to the drive) which must be connected to a configured XMP Transceiver or User Out signal. There are no restrictions as to which XMP signal is used except that current drive limitations may limit the number of drives connected to the same XMP signal.

The MEIMotorEncoder{} structure has been added to the MEIMotorConfig{} object:

```
typedef struct MEIMotorEncoder {
    MEIXmpEncoderType     type;
    long                  countsPerRev;
} MEIMotorEncoder;

typedef struct MEIMotorConfig {
    MEIMotorEncoder    Encoder[MEIXmpMotorEncoders];
    MEIXmpIO           StatusOutput[MEIXmpMotorStatusOutputs];

    MEIMotorTransceiver     Transceiver[MEIXmpMotorTransceivers];
    long                    UserOutInvert;     /* Opto Polarity */
    MEIMotorStepper         Stepper;
    long                    EncoderTermination;

    /* Commutation is read-only from field Theta to end*/
    MEIXmpCommutationBlock     Commutation;

    MEIXmpLimitData     Limit[MEIXmpLimitLAST];

    MEIMotorFilterInput     FilterInput[MEIXmpMotorFilterInputs];
} MEIMotorConfig;
```



Return to Motor Objects