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
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
mpiMotor**FeedbackGet**	Get feedback position
mpiMotor**FlashConfigGet**	Get flash config of motor
mpiMotor**FlashConfigSet**	Set flash config of motor
mpiMotor**IoGet**	Get dedicated I/O bits
mpiMotor**IoSet**	Set dedicated I/O bits
mpiMotor**Status** / meiMotor**Status**	Get motor status
mpiMotor**Type**	Get Motor type

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
mpiMotor**EventWait**	Set the contents of the structure pointed to by status.

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**EncoderInit**	Initializes an absolute encoder.
meiMotor**EncoderReset**	Resets an absolute encoder.

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**	
mpiMotor**DedicatedInAddrGet**	Get the address of the Dedicated IO for the Motor
meiMotor**DedicatedIoAddrDecode**	
meiMotor**DedicatedOutAddrGet**	
meiMotor**EncoderRatio**	Get encoder ratio from the XMP.
meiMotor**RelatedStepMotorGet**	

Data Types

MPIMotor**Brake**
MPIMotor**BrakeMode**
MPIMotor**Config** / MEIMotor**Config**
MEIMotor**DacConfig**
MEIMotor**DacChannelConfig**
MEIMotor**DacChannelStatus**
MEIMotor**DacStatus**
MPIMotor**EncoderFault**
MPIMotor**EncoderFaultMask**
MPIMotor**EventConfig** / MEIMotor**EventConfig**
MPIMotor**EventTrigger**
MEIMotor**Input**
MPIMotor**Io**
MPIMotor**Message** / MEIMotor**Message**
MEIMotor**Output**
MEIMotor**ResourceNumber**
MEIMotor**Status**
MEIMotor**Stepper**
MEIMotor**Transceiver**

MEIMotor**TransceiverConfig**
MEIMotor**TransceiverExtendedId**
MEIMotor**TransceiverExtendedMask**
MEIMotor**TransceiverId**
MEIMotor**TransceiverMask**
MPIMotor**Type**
MEIMotor**TypeInfo**

Macros

mpiMotor**EncoderFaultMaskBIT**

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.

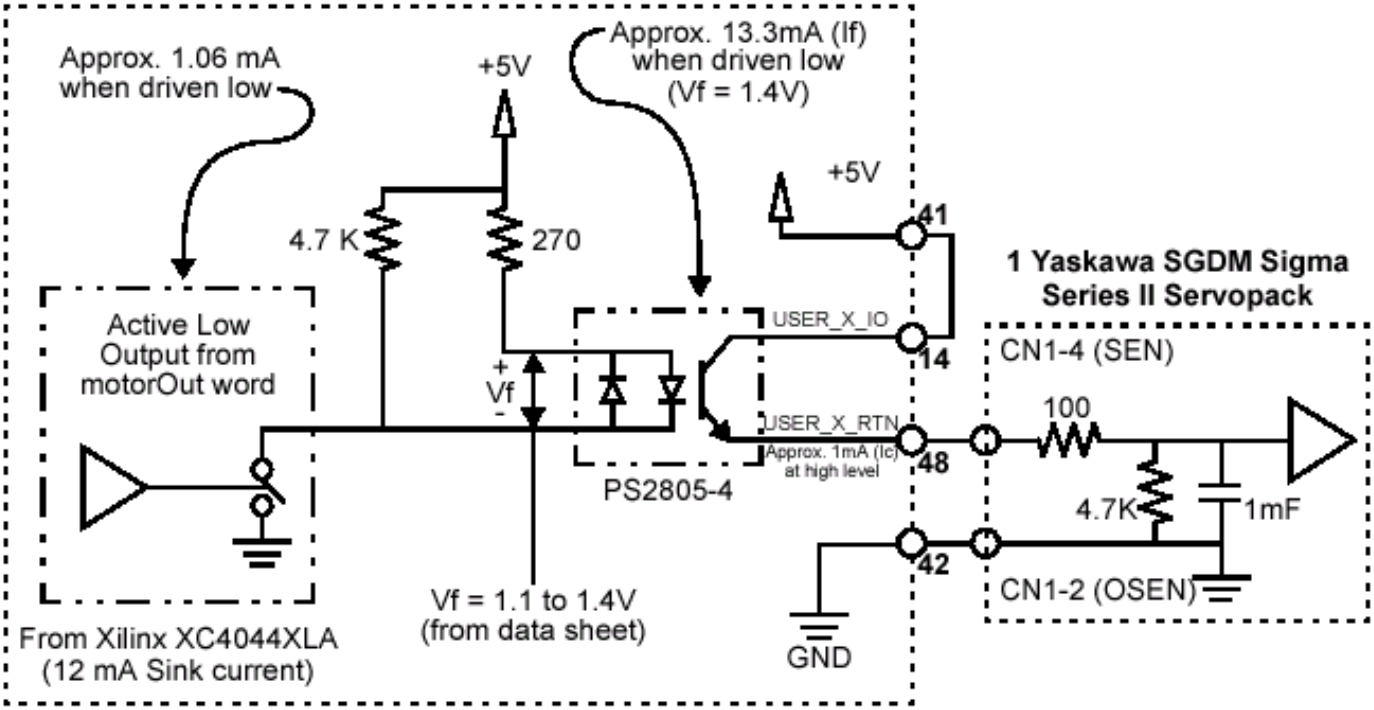
The encoders are automatically interrogated at power-up or after a controller reset. Axis Origin and Command Positions are set to correctly reflect the absolute position of the motor with no position error initially. The absolute position is the position within a single revolution.

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;
```

ABS Encoder Support



Return to Motor Objects page

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).

Return to mpiMotorConfigSet

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

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

mpiMotorAxisMapGet

Declaration

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

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 ***map***.

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
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 *dacConfig,
                          MEIFlash          flash);
```

Required Header stdmei.h

Description **MotorDacConfigGet** gets a Motor's (***motor***) Dac configuration and writes it to the structure pointed to by ***dacConfig***. The dac configuration located in ***flash*** is retrieved if the flash structure is not NULL.

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 *dacConfig,
                          MEIFlash          flash);
```

Required Header stdmei.h

Description **MotorDacConfigSet** configures a Motor's (***motor***) Dac using data from the structure pointed to by ***dacConfig***. The dac configuration located in ***flash*** is set if the flash structure is not NULL.

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

See Also meiMotorDacConfigGet | MEIMotorDacConfig

mpiMotorFeedbackGet

Declaration

```
long mpiMotorFeedbackGet(MPIMotor motor,
                         double   *feedback)
```

Required Header stdmpi.h

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

Return Values	
MPIMessageOK	if *MotorFeedbackGet* 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.

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigGet

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***.

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***.

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 ***status*** or ***external*** can be NULL (but not both NULL).

motor	a handle to the Motor object
***status**	a pointer to the motor status structure returned by the method
***external**	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.

See Also MPIStatus | MEIMotorStatus

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).

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**	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.

See Also

MPIStatus | MEIMotorStatus

mpiMotorType

Declaration

```
long mpiMotorType(MPIMotor      motor,
                  MPIMotorType  *type)
```

Required Header

stdmpi.h

Description

MotorType writes a Motor's (***motor***) type into the structure pointed to by ***type***.

Return Values	
MPIMessageOK	if *MotorType* successfully writes the Motor's type to the structure(s)

See Also

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.

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

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

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

mpiMotorEventWait

Declaration

```
long mpiNotifyEventWait(MPINotify       notify,
                        MPIEventStatus  *status,
                        MPIWait         timeout)
```

Required Header

stdmpi.h

Description

NotifyEventWait sets the contents of the structure pointed to by ***status***, using the status of the first event in the internal FIFO event queue (maintained by a Notify object (***notify***)), and then removes the first event from the queue. If no event is available in the internal FIFO event queue, NotifyEventWait will wait for ***timeout*** milliseconds.

If "timeout" is	*Then*
MPIWaitPOLL (**0**)	*NotifyEventWait* will not wait for an event to arrive
MPIWaitFOREVER (**-1**)	*NotifyEventWait* will wait forever for an event to arrive

Return Values	
MPIMessageOK	if *NotifyEventWait* successfully sets the contents of the structure pointed to by ***status***, and then removes the first event from the queue
MPIMessageTIMEOUT	if no event is present and the contents of ***status*** are undefined

See Also

mpiMotorMemory

Declaration

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

Required Header stdmpi.h

Description **MotorMemory** writes an address [which can be used to access a Motor's (***motor***) memory] to the contents of ***memory***. This address, or an address calculated from it, can be passed as the ***src*** parameter to MPIMotorMemoryGet(...) and as the ***dst*** parameter to MPIMotorMemorySet(...).

Return Values	
MPIMessageOK	*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

meiMotorEndcoderInit

Declaration

```
long meiMotorEncoderInit(MPIMotor        motor,
                         MEIMotorOutput  output,
                         MPIObjectMap    map)
```

Required Header stdmei.h

Description **MotorEncoderInit** inititalizes one or more absolute encoders associated with the specified MPIObjectMap map, using an output signal associated with the motor.

Return Values	
MPIMessageOK	if meiMotorEncoderInit(...) successfully initializes absolute encoder(s).
MEIMotorMessage ABS_ENCODER_FAULT	Returned when a protocol error occurs during the serial data transmission from the drive to controller.
MEIMotorMessage ABS_ENCODER_TIMEOUT	Returned when the controller does not receive serial data from the drive.

See Also MPIObjectMap | meiMotorEncoderInit

meiMotorEncoderReset

Declaration `long meiMotorEncoderReset(MPIMotor motor)`

Required Header stdmei.h

Description **MotorEncoderReset** clears the broken wire and illegal state detection latches for the encoder associated with the specified motor.

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

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  *map)
```

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 ***map.***

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  map)
```

Required Header stdmpi.h

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

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

mpiMotorDedicatedInAddrGet

Declaration

```
long mpiMotorDedicatedInAddrGet(MPIMotor    motor,
                                long        motorNumber)
```

Required Header stdmpi.h

Description **MotorDedicatedInAddrGet** gets the address of the Dedicated IO for the Motor that is specified by ***motorNumber***.

Return Values	
Address	The address of where the Dedicated IO is located in XMP memory.

See Also

meiMotorDedicatedIoAddrDecode

Declaration

```
long meiMotorDedicatedIOAddrDecode(MPIMotor    motor,
                                   long        addr,
                                   long*       motorNumber);
```

Required Header

stdmei.h

Description

MotorDedicatedIoAddrDecode takes an addr and decodes which motor's dedicated IO this ***addr*** points to, and then sets ***motorNumber*** to that motor's motorNumber.

Return Values

MPIMessageOK	if *motor* is a valid handle and *addr* is a valid address (eg. it is located in the motor memory space on the controller).

See Also

meiMotorDedicatedOutAddrGet

Declaration

```
long meiMotorDedicatedOutAddrGet(MPIMotor   motor,
                                 long       motorNumber);
```

Required Header

stdmei.h

Description

MotorDedicatedOutAddrGet gets the address of the Dedicated IO for the Motor that is specified by ***motorNumber***.

Return Values

Address	The address of where the Dedicated IO is located in XMP memory.

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.

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

See Also

meiMotorRelatedStepMotorGet

Declaration

```
long meiMotorRelatedStepMotorGet(MPIMotor    motor,
                                 long        *motorNumber)
```

Required Header stdmei.h

Description

MotorRelatedStepMotorGet searches for another motor that shares the resources of the motor provided. It then sets ***motorNumber*** to that motor's number. This scenario is only possible among Stepper Motors on a XMP Pulse controller. If there are no related motors found, ***motorNumber*** is set to MEIMotorResourceNumberINVALID.

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

See Also See the Software section of the XMP Pulse App Note 218.

MPIMotorBrake

MPIMotorBrake

```
typedef struct MPIMotorBrake {
    MPIMotorBrakeMode   mode;
    float               enableDelay;
    float               disableDelay;

} MPIMotorBrake;
```

Description

MotorBrake allows the ability to enable and disable a brake when the motor's amp enable output is enabled or disabled.

The reason for this is easiest to imagine on a vertical axis of motion: If you release the brake before enabling servo control on a vertical axis, the axis will not be controlled and will fall under the influence of gravity. Likewise, when setting a brake on a vertical axis, you want to set the brake before turning off the amplifier so that no motion occurs when disabling the servo control.

NOTE: the only output capable of being used for a brake is the USER I/O 0 output of the motor.

mode	Specifies whether a brake is to be tied to the motor's amp enable output.
enableDelay	The amount of time the brake is enabled before servo control is disabled. **NOTE:** the brake is enabled immediately, the delay is for servo control.
disableDelay	The amount of time servo control is turned on before the brake is disabled.

See Also

MPIMotorBrakeMode

MPIMotorBrakeMode

```
typedef enum{
     MPIMotorBrakeModeINVALID = -1,
     MPIMotorBrakeModeNONE,
     MPIMotorBrakeModeDELAY,

} MPIMotorBrakeMode;
```

Description

MotorBrakeMode specifies whether a brake enable is tied to the amp enable output.

MPIMotorBrakeModeNONE	No brake enable output is tied to the amp enable output.
MPIMotorBrakeModeDELAY	Ties a brake enable output to the amp enable output.

See Also

MPIMotorConfig / MEIMotorConfig

MPIMotorConfig

```
typedef struct MPIMotorConfig {
    MPIMotorType    type;

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

    long    ampEnablePolarity;  /* FALSE => active lo, else active hi */
    long    encoderPhase;       /* 0 => normal, else reversed */
    long    captureOnChange;    /* 0 => normal, else enabled */

    float           abortDelay;
    float           enableDelay;
    MPIMotorBrake   brake;

    MPIObjectMap    filterMap;

    MPIMotorIo       io;
} MPIMotorConfig;
```

Description

event	Structure to configure various Motor Events. See MPIMotorEventConfig description.
ampEnablePolarity	Configures Amplifier Enable Output polarity. For active low signal = FALSE, for Active High = TRUE
encoderPhase	Configures encoder phasing. Normal (A rising, B rising, A falling, etc.) = 0, Reversed = non-zero.
captureOnChange	Configures Captures objects for this motor to Capture on Rising and Falling transitions.
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	Settings for tying a brake enable output to the amp enable output.
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.
io	Structure to read the Motor input and set Motor output values.. See MPIMotorIo description.

MEIMotorConfig

```
typedef struct MEIMotorConfig {
    MEIMotorEncoder       Encoder[MEIXmpMotorEncoders];
    MEIXmpIO              StatusOutput[MEIXmpMotorStatusOutputs];
    MEIMotorTransceiver   Transceiver[MEIXmpMotorTransceivers];
    MEIMotorTransceiver   TransceiverExtended[MEIXmpMotorTransceiversExtended];
    long                  UserOutInvert;    /* Opto Polarity */
    MEIMotorStepper       Stepper;
    long                  EncoderTermination;
    long                  SIM4;
    MEIMotorDacConfig     Dac;
    long                  pulseEnable;     /* 0 => normal, else pulse output */
    long                  pulseWidth;      /* 0.1 to 25.5 microseconds */

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

    MEIXmpLimitData     Limit[MEIXmpLimitLAST];

    MEIXmpMotorTorqueLimitConfig TorqueLimitConfig;

    long AmpDisableWithLSR;     /* TRUE => XMP disables amp when LSR is active */

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

Description

	Encoder	Structure to configure Motor Encoder type and parameters
	Transceiver	Structure to configure Motor Transceivers. See MEIMotorTransceiver description.
	TransceiverExtended	Structure to configure Motor Extended Transceivers. See MEIMotorTransceiver description.
	UserOutInvert	Inverts the User Opto Polarity when wired as an Output.
	Stepper	Structure to configure Motor Stepper parameters. See MEIMotorStepper description.
	EncoderTermination	Enables encoder termination when set to 1. Enables 100 ohm resistor between complementary encoder channels. Set to 0 (zero) to disable.
	SIM4	Enables use of SIM4 scale interpolation module when set to 1. Set to 0 (zero) to disable.
	Dac	Structure that includes Command and Auxiliary DAC configuration for each motor. See MEIMotorDacConfig description.
	pulseEnable	Enables the Divide-by-N output pulse. See MEICompareConfig description.
	pulseWidth	Sets width of Step pulse. Valid values range from a minimum width of 0.1usec to a maximum width of 25.5usec.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

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

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.

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

MPIMotorEncoderFault

MPIMotorEncoderFault

```
typedef enum {
    MPIMotorEncoderFaultINVALID,
    MPIMotorEncoderFaultBW_DET,
    MPIMotorEncoderFaultILL_DET,
    MPIMotorEncoderFaultABS_ERR,

} MPIMotorEncoderFault;
```

Description

MotorEncoderFault is an enumeration used to get/set Encoder Fault.

MPIMotorEncoderFaultBW_DET	Encoder Broken Wire detection enabled. Error returned if complementary signals are in the same state. (e.g. A+ and A- return 5V.)
MPIMotorEncoderFaultILL_DET	Encoder Illegal State detection enabled.
MPIMotorEncoderFaultABS_ERR	Absolute encoder error detection enabled.

See Also

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

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.

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 Status" 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.

Value 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

Value 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

MPIMotorEventTrigger

MPIMotorEventTrigger

```
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;
```

Description

polarity	configures the polarity for Motor Event. Active Low = 0, Active High = non-zero.
position	configures position at which Positive and Negative Software limits generate Events.

See Also

MEIMotorInput

MEIMotorInput

```
typedef enum {
    MEIMotorInputSIM4_INDEX,
    MEIMotorInputSIM4_ENCB,
    MEIMotorInputSIM4_ENCA,
    MEIMotorInputXCVR_A,
    MEIMotorInputXCVR_B,
    MEIMotorInputXCVR_C,

    MEIMotorInputBROKEN_WIRE,
    MEIMotorInputILLEGAL_STATE,

    MEIMotorInputOVERTRAVEL_POS,
    MEIMotorInputOVERTRAVEL_NEG,
    MEIMotorInputHOME,
    MEIMotorInputAMP_FAULT,
    MEIMotorInputINDEX,

    MEIMotorInputUSER,
    MEIMotorInputCAPTURE,
} MEIMotorInput;
```

Description

MotorInput is an enumeration of the Motor Input register.

See Also

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	for XMP users, 'input' value reflects MEIMotorInput. See MEIMotorInput description.
output	for XMP users, 'output' value reflects MEIMotorOutput. See MEIMotorOutput description.

See Also

mpiMotorIoGet | mpiMotorIoSet

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.

MEIMotorMessage

```
typedef enum {
    MEIMotorMessageABS_ENCODER_FAULT,
    MEIMotorMessageABS_ENCODER_TIMEOUT,
    MEIMotorMessageMOTOR_NOT_ENABLED,

} MEIMotorMessage;
```

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

See Also

MEIMotorOutput

MEIMotorOutput

```
typedef enum {
    MEIMotorOutputXCVR_A,
    MEIMotorOutputXCVR_B,
    MEIMotorOutputXCVR_C,
    MEIMotorOutputBRAKE_ENABLE,
    MEIMotorOutputUSER,
    MEIMotorOutputCOMPARE,
} MEIMotorOutput;
```

Description

MotorOuput lists the Motor Output enumeration values.

See Also

MEIMotorResourceNumber

MEIMotorResourceNumber

```
typedef enum {
    MEIMotorResourceNumberINVALID,
    MEIMotorResourceNumber0,
    MEIMotorResourceNumber1,
    MEIMotorResourceNumber2,
    MEIMotorResourceNumber3,
    MEIMotorResourceNumber4,
    MEIMotorResourceNumber5,
    MEIMotorResourceNumber6,
    MEIMotorResourceNumber7,
    MEIMotorResourceNumber8,
    MEIMotorResourceNumber9,
    MEIMotorResourceNumber10,
    MEIMotorResourceNumber11,
    MEIMotorResourceNumber12,
    MEIMotorResourceNumber13,
    MEIMotorResourceNumber14,
    MEIMotorResourceNumber15,
    MEIMotorResourceNumber16,
    MEIMotorResourceNumber17,
    MEIMotorResourceNumber18,
    MEIMotorResourceNumber19,
    MEIMotorResourceNumber20,
    MEIMotorResourceNumber21,
    MEIMotorResourceNumber22,
    MEIMotorResourceNumber23,
    MEIMotorResourceNumber24,
    MEIMotorResourceNumber25,
    MEIMotorResourceNumber26,
    MEIMotorResourceNumber27,
    MEIMotorResourceNumber28,
    MEIMotorResourceNumber29,
    MEIMotorResourceNumber30,
    MEIMotorResourceNumber31,

}MEIMotorResourceNumber;
```

Description

MotorResourceNumber is an enumeration value used only for XMP-Pulse controllers. All other XMP controllers should have the Step Resource Number set equal to the Motor number. XMP-Pulse users can set the Resource Number to allow sharing of a single motor block's resources between two stepper motors. e.g. Motors 0 and 16 could both be configured to use the output signals from Motor Block 0 by each having their Motor Resource Number set to 0.

See Also

MEIMotorStatus

MEIMotorStatus

```
typedef struct MEIMotorStatus {
    MEIMotorDacStatus   dac;
} MEIMotorStatus;
```

Description

MotorStatus is a structure that returns XMP specific Motor Status registers.

See Also

MEIMotorStepper

MEIMotorStepper

```
typedef struct MEIMotorStepper {
    float       PulseWidth;     /* output pulse width  (sec) */
    long        Loopback;       /* TRUE = count step pulses in encoder reg. */
    MEIMotorResourceNumber      ResourceNumber;
} MEIMotorStepper;
```

Description

MotorStepper is a structure used to configure Stepper Motor parameters.

PulseWidth	sets width of Step pulse. Valid values range from a minimum width of 0.1 usec to maximum width of 25.5 usec.
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.

See Also

MEIMotorTransceiver

MEIMotorTransceiver

```
typedef struct MEIMotorTransceiver {
        long                          Invert;
                  /* TRUE = invert (not valid for INPUT) */
        MEIMotorTransceiverConfig     Config;
} MEIMotorTransceiver;
```

Description

MotorTransceiver is a structure used to configure Transceiver parameters.

Invert	Inverts polarity of Transceiver Output. Not valid when the Transeiver is configured as an Input.
Config	Enumeration to configure Transceiver usage. See MEIMotorTransceiverConfig description.

See Also

MEIMotorTransceiverConfig

MEIMotorTransceiverConfig

```
typedef enum {
    MEIMotorTransceiverConfigINVALID,
    MEIMotorTransceiverConfigINPUT,                /* 0 */
    MEIMotorTransceiverConfigOUTPUT,               /* 1 */
    MEIMotorTransceiverConfigSTEP,                 /* 2 */
    MEIMotorTransceiverConfigDIR,                  /* 3 */
    MEIMotorTransceiverConfigCW,                   /* 4 */
    MEIMotorTransceiverConfigCCW,                  /* 5 */
    MEIMotorTransceiverConfigQUAD_A,               /* 6 */
    MEIMotorTransceiverConfigQUAD_B,               /* 7 */
    MEIMotorTransceiverConfigCOMPARE,              /* 8 */
    MEIMotorTransceiverConfigDIAG,                 /* 9 */
    MEIMotorTransceiverConfigNOT_AVAILABLE,

} MEIMotorTransceiverConfig;
```

Description

MotorTranceiverConfig is a structure used to configure various Motor Events.

MEIMotorTransceiverConfigINPUT	Transceiver configured as Input
MEIMotorTransceiverConfigOUTPUT	Transceiver configured as Output
MEIMotorTransceiverConfigSTEP	Transceiver configured as Step pulse output
MEIMotorTransceiverConfigDIR	Transceiver configured as Direction signal output
MEIMotorTransceiverConfigCW	Transceiver configured as Clockwise Step Output
MEIMotorTransceiverConfigCCW	Transceiver configured as Counterclockwise Step Output
MEIMotorTransceiverConfigQUAD_A	Transceiver configured as Quadrature A Output
MEIMotorTransceiverConfigQUAD_B	Transceiver configured as Quadrature B Output
MEIMotorTransceiverConfigCOMPARE	Transceiver configured for use as Compare Output

See Also

MEIMotorTransceiverExtendedId

MEIMotorTransceiverExtendedId

```
typedef enum {
    MEIMotorTransceiverExtendedIdINVALID,

    MEIMotorTransceiverExtendedIdD,
    MEIMotorTransceiverExtendedIdE,
    MEIMotorTransceiverExtendedIdF,

} MEIMotorTransceiverExtendedId;
```

Description

MotorTranceiverExtendedId is an enumeration of Extended Transceiver Identification. Not valid on all XMP controllers.

See Also

MEIMotorTransceiverExtendedMask

MEIMotorTransceiverExtendedMask

```
typedef enum {
    MEIMotorTransceiverExtendedMaskD,
    MEIMotorTransceiverExtendedMaskE,
    MEIMotorTransceiverExtendedMaskF,
} MEIMotorTransceiverExtendedMask;
```

Description

MotorTransceiverExtendedMask is an enumeration of Extended Transceiver I/O masks. Not valid on all XMP controllers.

See Also

MEIMotorTransceiverId

MEIMotorTransceiverId

```
typedef enum {
    MEIMotorTransceiverIdINVALID,

    MEIMotorTransceiverIdA,
    MEIMotorTransceiverIdB,
    MEIMotorTransceiverIdC,

} MEIMotorTransceiverId
```

Description

MotorTraceiverId is an enumeration of Transceiver Identification.

See Also

MEIMotorTransceiverMask

MEIMotorTransceiverMask

```
typedef enum {
    MEIMotorTransceiverMaskA,
    MEIMotorTransceiverMaskB,
    MEIMotorTransceiverMaskC,
} MEIMotorTransceiverMask;
```

Description

MotorTranceiverMask is an enumeration of Extended Transceiver I/O masks. Not valid on all XMP controllers.

See Also

MPIMotorType

MPIMotorType

```
typedef enum {
    MPIMotorTypeINVALID,

    MPIMotorTypeSERVO,
    MPIMotorTypeSTEPPER,
    MPIMotorTypeSERCOS_DRIVE,

} MPIMotorType;
```

Description

MotorType is an enumeration of valid Motor Types.

MPIMotorTypeSERVO	Motor configured as Servo
MPIMotorTypeSTEPPER	Motor configured as Stepper
MPIMotorTypeSERCOS_DRIVE	Motor configured as SERCOS drive

See Also

MEIMotorTypeInfo

MEIMotorTypeInfo

```
typedef union {
    struct {
        long    sercosNumber;
        long    nodeNumber;
    } sercos;
} MEIMotorTypeInfo;
```

Description

MotorTypeInfo is a union with one member (a sercos structure) that contains the sercosNumber and nodeNumber for that motor.

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