Axis Objects

Introduction

An **Axis** object manages a single physical axis on a motion controller. It represents a reference line in a coordinate system. The controller calculates an axis's command position every sample based on the motion commanded by the Motion Supervisor. The Axis object contains command, actual, and error position data, plus status.

An Axis can have one or more Filters associated with it and each Filter can have one or more Motors associated with it. The Filter and Motor objects ensure the Axis command path is followed and that the control signals get to the correct motor. Complex mechanical systems with two (or more) motors can be mapped to a single axis of motion, abstracting the details of the physical hardware and making motion software much easier to develop.

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

Methods

Create, Delete, Validate Methods

mpiAxis**Create**	Create Axis object
mpiAxis**Delete**	Delete Axis object
mpiAxis**Validate**	Validate Axis object

Configuration and Information Methods

mpiAxis**ActualPositionGet**	Get actual position
mpiAxis**ActualPositionSet**	Set actual position
mpiAxis**ActualVelocity**	Get actual velocity
mpiAxis**ConfigGet**	Get Axis configuration
mpiAxis**ConfigSet**	Set Axis configuration
mpiAxis**CommandPositionGet**	Get command position
mpiAxis**CommandPositionSet**	Set command position
mpiAxis**FlashConfigGet**	Get Axis flash config
mpiAxis**FlashConfigSet**	Set Axis flash config
mpiAxis**OriginGet**	Get Axis origin
mpiAxis**OriginSet**	Set Axis origin
mpiAxis**PositionError**	Get position error of an Axis
mpiAxis**Status**	Get Axis status

mpiAxis**Trajectory**	Get Axis trajectory

Event Methods

mpiAxis**EventNotifyGet**	Get event mask
mpiAxis**EventNotifySet**	Set event mask
mpiAxis**EventReset**	

Memory Methods

mpiAxis**Memory**	Set Axis memory address
mpiAxis**MemoryGet**	Copy bytes of Axis memory to application memory
mpiAxis**MemorySet**	Copy bytes of application memory to Axis memory

Relational Methods

mpiAxis**Control**	Return handle of Control associated with Axis
mpiAxis**FilterMapGet**	Get object map of Filters
mpiAxis**FilterMapSet**	Set object map of Filters
mpiAxis**MotorMapGet**	Get object map of Motors
mpiAxis**Number**	Get index of Axis

Data Types

MPIAxis**Config** / MEIAxis**Config**

MPIAxis**InPosition**

MPIAxis**Master**

MPIAxis**MasterType**

MPIAxis**Message**

mpiAxisCreate

Declaration

```
MPIAxis MPIAxis mpiAxisCreate(MPIControl control,
                              long       number)
```

Required Header

stdmpi.h

Description

AxisCreate creates an axis object associated with the axis identified by ***number*** located on motion controller ***control***. AxisCreate is the equivalent of a C++ constructor.

control	a handle to Axis object.
number	the number specifies which Axis object is being created. The number corresponds to an Axis object in XMP memory.

Remarks

An **Axis** represents a physical axis in space such as X, Y, Z, Theta, or other axes. An Axis may be comprised of one or more motors, such as with a gantry system.

Return Values

handle	to an Axis object
MPIHandleVOID	if the object could not be created

See Also

mpiAxisDelete | mpiAxisValidate

mpiAxisDelete

Declaration

```
long mpiAxisDelete(MPIAxis  axis)
```

Required Header

stdmpi.h

Description

AxisDelete deletes an Axis object and invalidates its handle (**axis**).
AxisDelete is the equivalent of a C++ destructor.

axis	the Axis handle to be deleted

Remarks

All objects that are created in an application should be deleted in reverse order at the end of the code.

Return Values

MPIMessageOK	if *AxisDelete* successfully deletes an Axis object and invalidates its handle

See Also

mpiAxisCreate | mpiAxisValidate

mpiAxisValidate

Declaration

```
long mpiAxisValidate(MPIAxis   axis)
```

Required Header

stdmpi.h

Description

AxisValidate validates the Axis object and its handle (***axis***). AxisValidate should be called immediately after an object is created.

axis	a handle to the Axis object to be validated

Return Values

MPIMessageOK	if Axis is a handle to a valid object.

See Also

mpiAxisCreate | mpiAxisDelete

mpiAxisActualPositionGet

Declaration

```
long mpiAxisActualPositionGet(MPIAxis  axis,
                              double   *actual)
```

Required Header

stdmpi.h

Description

axis	a handle to an Axis object
***actual**	a pointer to the Axis actual position returned by the method.

Return Values

MPIMessageOK	if *AxisActualPositionGet* successfully gets and writes the value of the actual postion of ***axis*** to the location.

See Also

mpiAxisActualPositionSet | Using the Origin Variable

mpiAxisActualPositionSet

Declaration

```
long mpiAxisActualPositionSet(MPIAxis  axis,
                              double   actual)
```

Required Header

stdmpi.h

Description

AxisActualPositionSet sets the value of the actual position of an Axis (***axis***) to ***actual***.

axis	a handle to the Axis object
actual	value to which the Axis actual position will be set

Return Values

MPIMessageOK	if *AxisActualPositionSet* successfully sets the value of the actual postion of an Axis to ***actual***

See Also

AxisCommandPositionSet | Using the Origin Variable

mpiAxisCommandPositionSet

Declaration

```
long mpiAxisCommandPositionSet(MPIAxis  axis,
                               double   command)
```

Required Header

stdmpi.h

Description

AxisCommandPositionSet sets the value of the command position of an Axis (**axis**) from ***command***.

mpiAxisCommandPositionSet(...) Error Check
The mpiAxisCommandPositionSet(...) error check has been extended. If the controller is updating the axis's command position when mpiAxisCommandPositionSet(...) is called, MPIAxisMessageCOMMAND_NOT_SET will be returned. mpiAxisCommandPositionSet(...) checks for the following conditions:

- Axis is in a STOPPING, STOPPED, or MOVING state.
- Any motor associated with the axis has the disableAction configuration set to MEIMotorDisableActionCMD_EQ_ACT and the motor's Amp Enable is disabled.
- If the command position read from the controller does not match the requested position.

axis	a handle to the Axis object
command	value to which the Actual command position will be set

Remarks

Setting the Axis Command Position may cause the axis to jump. See the discussion of the Axis Origin before using the AxisActualPositionSet and AxisCommandPositionSet methods.

Return Values

MPIMessageOK	if *AxisCommandPositionSet* successfully sets the value of the command position of ***axis*** from ***command***

Axis: unable to set command position	Indicates that the motor associated to this axis is configured to mode MEIMotorDisableActionCMD_EQ_ACT and the motor is disabled. When the motor is disabled and in the MEIMotorDisableActionCMD_EQ_ACT mode, the command position is continually set to the actual position. To set the command position, enable the motor or take the motor out of the MEIMotorDisableActionCMD_EQ_ACT mode. Alternatively, setting the origin for the motor can often perform an equivalent result in this situation, as the command position will be set to the actual position the next sample.

See Also

MEIMotorDisableAction | AxisActualPositionSet | AxisCommandPositionSet | MPIAxisMessage

mpiAxisActualVelocity

Declaration

```
long mpiAxisActualVelocity(MPIAxis   axis,
                           double    *actual)
```

Required Header

stdmpi.h

Description

AxisActualVelocity reads the value of the actual velocity (in counts per servo sample) on an Axis (***axis***) and writes it in the location pointed to by ***actual***.

Return Values	
MPIMessageOK	if *AxisActualVelocity* succesfully gets and writes the value of the actual velocity of ***axis*** to the location.

See Also

mpiAxisConfigGet

Declaration

```
long mpiAxisConfigGet(MPIAxis       axis,
                      MPIAxisConfig *config,
                      void          *external)
```

Required Header

stdmpi.h

Description

AxisConfigGet gets the configuration of an Axis (**axis**) 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).

axis	a handle to the Axis object
***config**	pointer to the MPIAxisConfig structure
***external**	pointer to an external. See remarks below

Remarks

For XMP controllers, ***external*** either points to a structure of type **MEIAxisConfig{}** or is NULL.

Sample Code

```
/* Change axis encoder scaling.
   limit scale to +/- 2.0 */
void axisScale(MPIAxis axis, float scale)
{
    MPIAxisConfig config;
    MEIAxisConfig xmpConfig;

    mpiAxisConfigGet(axis, &config, &xmpConfig);
    xmpConfig.APos[0].Coeff = (long)(scale * MEIXmpFRACTIONAL_UNITY);
    mpiAxisConfigSet(axis, &config, &xmpConfig);
}
```

Return Values

MPIMessageOK	if *AxisConfigGet* successfully gets the Axis configuration and writes it into the structure(s)

See Also

MPIAxisConfig | mpiAxisConfigSet | MEIAxisConfig

mpiAxisConfigSet

Declaration

```
long mpiAxisConfigSet(MPIAxis       axis,
                      MPIAxisConfig *config,
                      void          *external)
```

Required Header

stdmpi.h

Description

AxisConfigSet sets the configuration of an Axis (***axis***) 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).

The MEIXmpAxisGear firmware feature only supports servo motor types. The axis gear feature does not support step motor types.

axis	a handle to the Axis object
***config**	pointer to an MPIAxisConfig structure.
***external**	pointer to an external. See remarks below.

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

Sample Code

```
/* Change axis encoder scaling.
   limit scale to +/- 2.0 */
void axisScale(MPIAxis axis, float scale)
{
    MPIAxisConfig config;
    MEIAxisConfig xmpConfig;

    mpiAxisConfigGet(axis, &config, &xmpConfig);
    xmpConfig.APos[0].Coeff = (long)(scale * MEIXmpFRACTIONAL_UNITY);
    mpiAxisConfigSet(axis, &config, &xmpConfig);
}
```

Return Values

MPIMessageOK	if *AxisConfigSet* successfully sets the Axis configuration.

See Also

mpiAxisConfigGet | MEIAdcConfig | MEIAxisConfig

mpiAxisCommandPositionGet

Declaration

```
long mpiAxisCommandPositionGet(MPIAxis axis,
                               double  *command)
```

Required Header

stdmpi.h

Description

AxisCommandPositionGet gets the value of the command position of an Axis (**axis**) and puts it in the location pointed to by ***command***.

axis	a handle to the Axis object
***command**	a pointer to the Axis command position returned by the method

Return Values

MPIMessageOK	if *AxisCommandPositionGet* successfully gets the value of the command position of ***axis*** and puts it in the location.

See Also

mpiAxisCommandPositionSet

mpiAxisFlashConfigGet

Declaration

```
long mpiAxisFlashConfigGet(MPIAxis       axis,
                           void          *flash,
                           MPIAxisConfig *config,
                           void          *external)
```

Required Header

stdmpi.h

Description

AxisFlashConfigGet gets the flash configuration for an Axis (***axis***) and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***sexternal*** (if ***external*** is not NULL).

The Axis flash configuration information in ***external*** is in addition to the Axis 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).

axis	a handle to the Axis object
***flash**	
***config**	pointer to an MPIAxisConfig structure
***external**	pointer to an external. See remarks below.

Remarks

For XMP controllers, ***external*** either points to a structure of type **MEIAxisConfig{}** or is NULL. ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values

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

See Also

MEIFlash | mpiAxisFlashConfigSet |MEIAxisConfig

mpiAxisFlashConfigSet

Declaration

```
long mpiAxisFlashConfigSet(MPIAxis          axis,
                           void             *flash,
                           MPIAxisConfig    *config,
                           void             *external)
```

Required Header

stdmpi.h

Description

AxisFlashConfigSet sets the flash configuration for for an Axis (***axis***) 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 Axis flash configuration information in ***external*** is *in addition* to the Axis 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).

axis	a handle to the Axis object
***flash**	
***config**	pointer to an MPIAxisConfig structure
***external**	pointer to an external. See remarks below.

XMP Only

external either points to a structure of type **MEIAxisConfig{}** or is NULL. ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values

MPIMessageOK	if *AxisFlashConfigSet* successfully sets the Axis flash configuration using data from the structure(s)

See Also

MEIFlash | mpiAxisFlashConfigGet | MEIAxisConfig

mpiAxisOriginGet

Declaration

```
long mpiAxisOriginGet(MPIAxis  axis,
                      double   *origin)
```

Required Header

stdmpi.h

Description

AxisOriginGet gets the value of the origin of an Axis (***axis***) and writes it into the location pointed to by ***origin***.

axis	a handle to the Axis object.
***origin**	pointer to the Origin value returned by the method

Return Values

MPIMessageOK	if *AxisOriginGet* successfully gets the value of the origin of the Axis and writes it to the location

See Also

mpiAxisOriginSet | Using the Origin Variable

mpiAxisOriginSet

Declaration

```
long mpiAxisOriginSet(MPIAxis  axis,
                      double   origin)
```

Required Header

stdmpi.h

Description

AxisOriginSet sets the value of the origin of an Axis (***axis***) to ***origin***.

axis	a handle to the Axis object
origin	Value to which the Axis Origin will be set

Return Values

MPIMessageOK	if *AxisOriginSet* successfully sets the origin of an Axis to ***origin***

See Also

mpiAxisOriginGet | Using the Origin Variable

mpiAxisPositionError

Declaration

```
long mpiAxisPositionError(MPIAxis  axis,
                          double  *error)
```

Required Header

stdmpi.h

Description

AxisPositionError gets the value of the position error of an Axis (***axis***) and puts it in the location pointed to by ***error***. The position error is equal to (command position - actual position).

axis	a handle to the Axis object
***error**	a pointer to the Axis position error returned by the method

Return Values

MPIMessageOK	if *AxisPositionError* successfully gets and writes the value of the position error into ****error***

See Also

mpiAxisCommandPositionGet | mpiAxisActualPositionGet

mpiAxisStatus

Declaration

```
long mpiAxisStatus(MPIAxis    axis,
                   MPIStatus  *status,
                   void       *external)
```

Required Header

stdmpi.h

Description

AxisStatus gets the status of an Axis (***axis***) and writes it into the structure pointed to by ***status*** and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

axis	a handle to the Axis object
***status**	pointer to MPIStatus structure.
***external**	pointer to an implementation-specific structure.

XMP Only

external should always be set to NULL.

Return Values

MPIMessageOK	if *AxisStatus* successfully gets the Axis status and writes it into the structure(s)
MPIMessageARG_INVALID	if the *status* pointer is NULL.

See Also

mpiAxisTrajectory

Declaration

```
long mpiAxisTrajectory(MPIAxis       axis,
                       MPITrajectory *trajectory)
```

Required Header

stdmpi.h

Description

AxisTrajectory reads the current velocity and acceleration of ***axis*** and writes it into the structure pointed to by ***trajectory***.

NOTE: deceleration, jerkPercent, accelerationJerk, and decelerationJerk fields of ***trajectory*** cannot be read from the controller and consequently are set to zero.

axis	a handle to the Axis object.
***trajectory**	pointer to the MPITrajectory structure

Remarks

The default MPITrajectory structure can be used by the mpiMotionStart(...) and mpiMotionModify() methods.

Sample Code

```
MPITrajectory trajectory;

    mpiAxisTrajectory(axis, &trajectory);

    printf("Velocity %.3f\n"
           "Acceleration %.3f\n",
           trajectory.velocity,
           trajectory.acceleration);
```

Return Values

MPIMessageOK	if *AxisTrajectory* successfully gets the Axis trajectory and writes it into the structure

See Also

mpiMotionStart | mpiMotionModify | MPITrajectory

mpiAxisEventNotifyGet

Declaration

```
long mpiAxisEventNotifyGet(MPIAxis          axis,
                           MPIEventMask     *eventMask,
                           void             *external)
```

Required Header

stdmpi.h

Description

AxisEventNotifyGet 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).

axis	a handle to the Axis object
***eventMask**	pointer to an MPIEventMask
***external**	pointer to an external. See remarks below

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 *AxisEventNotifyGet* successfully writes the event mask to the location(s)

See Also

MEIEventNotifyData | MEIEventStatusInfo | mpiAxisEventNotifySet

mpiAxisEventNotifySet

Declaration

```
long mpiAxisEventNotifySet(MPIAxis        axis,
                           MPIEventMask   eventMask,
                           void           *external)
```

Required Header

stdmpi.h

Description

AxisEventNotifySet requests host notification of the event(s) that are generated by ***axis*** 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).

axis	a handle to the Axis object
eventMask	pointer to an MPIEventMask
***external**	pointer to an external

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	configure ***eventmask*** with mpiEventMaskALL(eventMask)
disable host notification of all events	configure ***eventmask*** with mpiEventMaskCLEAR(eventMask)

Return Values

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

See Also

MEIEventNotifyData | MEIEventStatusInfo | MPIEventMask | MPIEventType | mpiEventMaskALL | mpiEventMaskCLEAR | mpiAxisEventNotifyGet | MEIEventNotifyData

mpiAxisEventReset

Declaration

```
long mpiAxisEventReset(MPIAxis      axis,
                       MPIEventMask eventMask)
```

Required Header

stdmpi.h

Description

To...	Then...
enable host notification of all events	configure ***eventmask*** with mpiEventMaskALL(eventMask)
disable host notification of all events	configure ***eventmask*** with mpiEventMaskCLEAR(eventMask)

Return Values

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

Remarks

Event notification is enabled for event types specified in ***eventMask***, a bit mask generated by the logical OR of the MPIEventMask bits associated with the desired MPIEventType values. Configuration of the eventMask should be done with Event macros. Event notification is disabled for event types that are not specified in ***eventMask***.

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

See Also

MEIEventNotifyData | MEIEventStatusInfo | MPIEventType | mpiEventMaskALL
mpiEventMaskCLEAR | MPIEventMaskMOTION | MPIEventMaskAXIS | MPIEventMask

mpiAxisMemory

Declaration

```
long mpiAxisMemory(MPIAxis  axis,
                   void     **memory)
```

Required Header

stdmpi.h

Description

AxisMemory writes an address (that is used to access Axis memory) to the contents of ***memory***. This address (or an address calculated from it) is passed as the ***src*** argument to mpiAxisMemoryGet(...) and as the ***dst*** argument to mpiAxisMemorySet(...).

axis	a handle to the Axis object

Return Values

MPIMessageOK	if *AxisMemory* successfully writes the Axis memory address to the contents of ***memory***

See Also

mpiAxisMemoryGet | mpiAxisMemorySet

mpiAxisMemoryGet

Declaration

```
long mpiAxisMemoryGet(MPIAxis axis,
                      void    *dst,
                      void    *src,
                      long    count)
```

Required Header

stdmpi.h

Description

AxisMemoryGet copies ***count*** bytes of Axis (***axis***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

axis	a handle to the Axis object
***dst**	pointer to the destination location to where the memory will be written
***src**	pointer to the source location of memory being read
count	size of memory to be read

Return Values

MPIMessageOK	if *AxisMemory* successfully writes the Axis memory address to the contents of memory

See Also

mpiAxisMemory | mpiAxisMemorySet

mpiAxisMemorySet

Declaration

```
long mpiAxisMemorySet(MPIAxis axis,
                      void    *dst,
                      void    *src,
                      long    count)
```

Required Header

stdmpi.h

Description

AxisMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to Axis (***axis***) memory (starting at address ***dst***).

axis	a handle to the Axis object
***dst**	pointer to the destination location to where the memory will be written
***src**	pointer to the source location of memory being read
***count**	size of memory to be written

Return Values

MPIMessageOK	if *AxisMemorySet* successfully copies ***count*** bytes of application memory to Axis memory

See Also

mpiAxisMemory | mpiAxisMemoryGet

mpiAxisControl

Declaration

```
MPIControl mpiAxisControl(MPIAxis  axis)
```

Required Header

stdmpi.h

Description

AxisControl returns a handle to the motion controller (Control) with which an Axis (***axis***) is associated.

axis	a handle to the Axis object

Return Values

MPIHandleVOID	if **axis** is invalid

See Also

mpiAxisFilterMapGet

Declaration

```
long mpiAxisFilterMapGet(MPIAxis        axis,
                         MPIObjectMap  *filterMap)
```

Required Header

stdmpi.h

Description

AxisFilterMapGet gets the object map of the Filters [associated with an Axis (***axis***)] and writes it into the structure pointed to by ***motorMap***.

axis	a handle to the Axis object
***filterMap**	a pointer to an ObjectMap of Filters mapped to the axis

Remarks

MPIObjectMap is a *long* that maps the Filters in controller memory to each bit. E.g. A map value of 1 would indicate Filter 0 is mapped the Axis. A value of 6 would indicate that Filters 2 and 3 are mapped to the Axis.

Return Values

MPIMessageOK	if *AxisFilterMapGet* successfully gets and writes the object map to the structure

See Also

mpiAxisFilterMapSet

mpiAxisFilterMapSet

Declaration

```
long mpiAxisFilterMapSet(MPIAxis       axis,
                         MPIObjectMap  filterMap)
```

Required Header

stdmpi.h

Description

AxisFilterMapSet sets the Filters [associated with an Axis (***axis***)] using data from the object map specified by ***filterMap***.

axis	a handle to the Axis object
filterMap	a list of Filters to be mapped to the axis

Remarks

MPIObjectMap is a *long* that maps the Filters in controller memory to each bit. E.g. A map value of 1 will map Filter 0 to the Axis. A value of 6 will map both Filters 2 and 3 to the Axis.

Return Values

MPIMessageOK	if *AxisFilterMapSet* successfully sets the Filters using data from the object map

See Also

mpiAxisFilterMapGet | MPIObjectMap

mpiAxisMotorMapGet

Declaration

```
long mpiAxisMotorMapGet(MPIAxis       axis,
                        MPIObjectMap *motorMap)
```

Required Header

stdmpi.h

Description

AxisMotorMapGet gets the object map [of the Motors associated with an Axis (***axis***)] and writes it into the structure pointed to by ***motorMap***.

axis	a handle to the Axis object.
***motorMap**	a pointer to an ObjectMap of Motors mapped to the axis

Remarks

MPIObjectMap is a *long* that maps the Motors in controller memory to each bit. E.g. A **map** value of 1 would indicate Motor 0 is mapped the Axis. A value of 6 would indicate that Motors 2 and 3 are mapped to the Axis.

Remember that Motors are mapped to Axes through the Filter object. To configure the Axis/Motor map, the application will need to set the AxisFilterMap and FilterMotorMap.

Return Values

MPIMessageOK	if *AxisMotorMapGet* successfully gets the object map and writes it into the structure

See Also

mpiAxisNumber

Declaration

```
long mpiAxisNumber(MPIAxis  axis,
                   long     *number)
```

Required Header

stdmpi.h

Description

AxisNumber writes the index of an Axis (***axis***, on the motion controller that the Axis is associated with) to the contents of ***number***.

axis	a handle to the Axis object
***number**	pointer to the number

Return Values

MPIMessageOK	if *AxisNumber* successfully writes the index of Axis to the contents of ***number***

See Also

MPIAxisConfig / MEIAxisConfig

MPIAxisConfig

```
typedef struct MPIAxisConfig {
    MPIAxisInPosition    inPosition;
    MPIAxisMaster        master;
    long                 masterCorrection;
    MPIObjectMap         filterMap;
}MPIAxisConfig;
```

Description

inPosition	See MPIAxisInPosition.
master	This field defines the source of the position and velocities used as the master for cam motion. See Master Position Source.
masterCorrection	Specifies which axis provides the master position correction. A value of -1 stops any stops master corrections from being used. See Camming: Correctional Moves.
filterMap	bitmap indicating which Filter objects are mapped to the Axis. See MPIObject for more details.

MEIAxisConfig

```
typedef struct MEIAxisConfig {
    MEIXmpAPosInput        APos [MEIXmpAxisAPosInputs];
    MEIXmpAxisFilter       Filter;
    MEIXmpAxisGear         Gear;
}MEIAxisConfig;
```

Description

APos - an array of structures that set Actual position inputs. The structure has two elements:

- **Ptr** - Pointer to Actual position input register. Default value is corresponding encoder input.
- **Coeff** - Coefficient that multiplies the encoder input. Coeff is a custom unit. The range of Coeff is +/- 2.0 (+/- 2*MEIXmpFRACTIONAL_UNITY).

  For a 1:1 ratio of encoder input to reported encoder input set:
  Coeff = MEIXmpFRACTIONAL_UNITY.

  For 0.5:1 ratio, set:
  Coeff = MEIXmpFRACTIONAL_UNITY / 2.

When the distance between the positive and negative limit configurations exceed 32 bits

(4,294,967,296 counts), both limits are triggered. The distance between the positive and negative software position limits must be less than 32 bits (4,294,967,296 counts).

Filter

- Input
- Output
- Delta
- Delay
- Timer
- Pointer

Gear - Coefficients for gearing off a position input. The MEIXmpAxisGear firmware feature only supports servo motor types. The axis gear feature does not support step motor types.

- **Ptr** - Host pointer to a gear master

  Example:

```
MEIXmpData         *firmware;
MEIXmpBufferData   *bufferData;

mpiControlMemory(control,&firmware,&bufferData);
...
msgCHECK(mpiAxisConfigGet(axis, &axisConfig, &axisConfigXmp));
axisConfigXmp.Gear.Ptr = &bufferData->PreFilter[0].Output;
msgCHECK(mpiAxisConfigSet(axis, &axisConfig, &axisConfigXmp));
```

- **Ratio.A** - numerator of multiplier
- **Ratio.B** - denominator of multiplier
- **Ratio.Old** -
- **Ratio.Remainder** -
- **Position** - final geared position

Sample Code

```
/* Change axis encoder scaling.
   limit scale to +/- 2.0 */
void axisScale(MPIAxis axis, float scale)
{
    MPIAxisConfig config;
    MEIAxisConfig xmpConfig;

    mpiAxisConfigGet(axis, &config, &xmpConfig);
    xmpConfig.APos[0].Coeff = (long)(scale * MEIXmpFRACTIONAL_UNITY);
    mpiAxisConfigSet(axis, &config, &xmpConfig);
}
```

See Also

mpiAxisConfigGet | mpiAxisConfigSet | MPIAxisInPosition | MPIObject

MPIAxisInPosition

MPIAxisInPosition

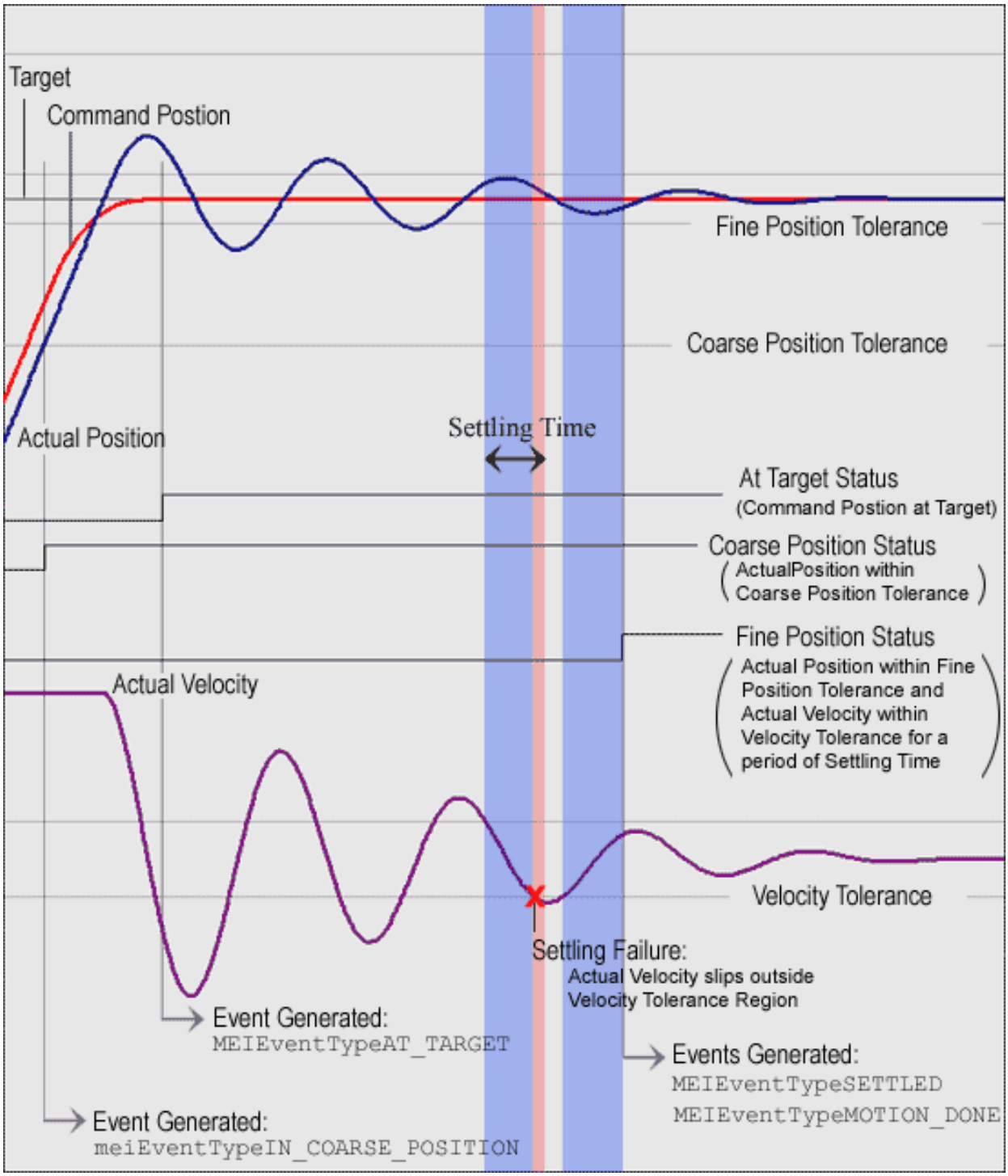
```
typedef struct MPIAxisInPosition {
    struct {
        float   positionFine;
        long    positionCoarse;
        float   velocity;
    } tolerance;
        float   settlingTime;  /* seconds */
        long    settleOnStop;
        long    settleOnEstop
        long    settleOnEstopCmdEqAct;
} MPIAxisInPosition;
```

Description

tolerance	Includes the following 3 elements that determine settling tolerances for an axis.
positionFine	Value, in counts, from the move target position at which the controller sets the "in fine position" status flag. This parameter is used as part of the Axis settling criteria to determine when a point-to-point motion is complete and when MPIEventTypeMOTION_DONE and MEIEventTypeSETTLEDevents are generated.
positionCoarse	Value, in counts, from a move target position at which the controller sets the "in coarse position" status flag. This value does not affect the settling time status.
velocity	Value, in counts/second, from the final move velocity at which the controller sets the "at velocity" status flag. This parameter is used as part of the Axis settling criteria to determine when: - a position-based move is complete and an MPIEventTypeMOTION_DONE event is generated - a velocity move is complete and an MPIEventTypeMOTION_AT_VELOCITY event is generated - an axis is settled and an MPIEventTypeSETTLED event is generated
settlingTime	Duration in seconds that an axis must satisfy the positionFine and/or velocity tolerance, before the respective status flag is set.

settleOnStop	If TRUE, the controller will use settle on stop mode. If FALSE, the controller will not use the settle on stop mode. When in settleOnStop mode and a STOP event has occurred, the axis will stay in an MPIStateSTOPPING state until: **1.** The settling criteria are satisfied AND **2.** The stop duration for the axis' Motion Supervisor has elapsed. **3.** This state can be read with mpiAxisStatus(MPIAxis axis, MPIStatus *status, void *external). The value to look for is (MPIState) status.state. If settleOnStop = FALSE, the axis will stay in an MPIStateSTOPPING state only until the stop duration for the axis' Motion Supervisor has elapsed.
settleOnEstop	If TRUE, the controller will use settle on Estop mode. If FALSE, the controller will not use the settle on Estop mode. When in settleOnEstop mode and a ESTOP event has occurred, the axis will stay in an MPIStateSTOPPING_ERROR state until: **1.** The settling criteria are satisfied AND **2.** The Estop duration for the axis' Motion Supervisor has elapsed. **3.** This state can be read with mpiAxisStatus(MPIAxis axis, MPIStatus *status, void *external). The value to look for is (MPIState) status.state. If settleOnEStop = FALSE, the axis will stay in an MPIStateSTOPPING_ERROR state only until the Estop duration for the axis' Motion Supervisor has elapsed.
settleOnEstopCmdEqAct	If TRUE, the controller will use settle on EstopCmdEqAct mode. If FALSE, the controller will not use the settle on EstopCmdEqAct mode. ***settleOnEstopCmdEqAct mode is not recommended*** SettleOnEstopCmdEqAct is an alternative to Estop mode. When this mode is enabled, the following things happen: - During normal motion, there is no difference. - During an Estop, Cmd Eq Act action, the command position is set equal to the actual position from the previous servo sample. This can have a damping effect in some systems with some tuning parameters, causing the stage to slow. The behavior of the stage in this mode can be vastly different than in normal

	servoing mode. Approach this mode with great caution. The axis will stay in this mode for the amount of time that the Axis' Motion Supervisor Estop time. - After the Estop time elapses, the axis' motors will disable the amplifiers.



Sample Code

```
/*
    Set the settling time of an axis.  Sample usage:
    returnValue =
        setAxisSettlingTime(axis, 0.05);
*/
long setAxisSettlingTime(MPIAxis axis, double settlingTime)
{
    MPIAxisConfig config;
    long returnValue;

    returnValue =
        mpiAxisConfigGet(axis, &config, NULL);

    if (returnValue == MPIMessageOK)
    {
        config.inPosition.settlingTime = (float) settlingTime;
        returnValue =
            mpiAxisConfigSet(axis, &config, NULL);
}

    return returnValue;
}
```

See Also

MPIAxisConfig | MPIAction

Axis Tolerances and How Motion Related Events are Generated
How Motion Completion Events are Generated

Special Note on Configuration of IN_POSITION and Done Events after STOP or E_STOP Events

MPIAxisMaster

MPIAxisMaster

```
typedef enum {
    MPIAxisMasterType  type;
    long               number;
    long               *address;
    long               encoderFaultMotorNumber;
}MPIAxisMaster;
```

Description

AxisMaster defines the source of the position and velocities used as the master for cam motion. See also Master Position Source.

The *type* field specifies if the number or address fields are used and which object the number field refers to.

MPIMasterType	Number	Address
MPIAxisMasterTypeMOTOR_FEEDBACK_PRIMARY	motor number	Not used
MPIAxisMasterTypeMOTOR_FEEDBACK_SECONDARY	motor number	Not used
MPIAxisMasterTypeAXIS_COMMANDED POSITION	Axis number	Not used
MPIAxisMasterTypeAXIS_ACTUAL POSITION	Axis number	Not used
MPIAxisMasterTypeADDRESS	Not used	Any controller address

type	This field defines the type of master position source is being used.
number	the motor or axis number.
address	The controller address to be used as the master position.
encoderFaultMotorNumber	The number of the motor that is checked for an encoder fault. If this motor detects an encoder fault this axis will abort. A value of -1 disables this encoder fault function. See Master Encoder Faults.

See Also

MPIAxisMasterType

MPIAxisMasterType

MPIAxisMasterType

```
typedef enum {
    MPIAxisMasterTypeMOTOR_FEEDBACK_PRIMARY,
    MPIAxisMasterTypeMOTOR_FEEDBACK_SECONDARY,
    MPIAxisMasterTypeAXIS_COMMANDED_POSITION,
    MPIAxisMasterTypeAXIS_ACTUAL_POSITION,
    MPIAxisMasterTypeADDRESS,
}MPIAxisMasterType;
```

Description

AxisMaster specifies the type of master position source used with cam motions. See also MPIAxisMaster.

Fields	Number	Address
MPIAxisMasterTypeMOTOR_FEEDBACK_PRIMARY	Motor number	Not used
MPIAxisMasterTypeMOTOR_FEEDBACK_SECONDARY	Motor number	Not used
MPIAxisMasterTypeAXIS_COMMANDED_POSITION	Axis number	Not used
MPIAxisMasterTypeAXIS_ACTUAL_POSITION	Axis number	Not used
MPIAxisMasterTypeADDRESS	Not used	Any controller address

See Also

MPIAxisMaster

MPIAxisMessage

MPIAxisMessage

```
typedef enum {
    MPIAxisMessageAXIS_INVALID,
    MPIAxisMessageCOMMAND_NOT_SET,
    MPIAxisMessageNOT_MAPPED_TO_MS,
}MPIAxisMessage;
```

Description

AxisMessage is an enumeration of Axis error messages that can be returned by the MPI library.

MPIAxisMessageAXIS_INVALID

The axis number is out of range. This message code is returned by mpiAxisCreate(…) if the axis number is less than zero or greater than or equal to MEIXmpMAX_Axes.

MPIAxisMessageCOMMAND_NOT_SET

The axis command position did not get set. This message code is returned by mpiAxisCommandPositionSet(…) if the controller's command position does not match the specified value. Internally, the mpiAxisCommandPositionSet(…) method requests the controller to change the command position, waits for the controller to process the request, and reads back the controller's command position. There are several cases where the controller will calculate a new command position to replace the requested command position. For example, if motion is in progress, stopped, or if the amp enable is disabled (when the motor's disableAction is configured for command equals actual), the controller will calculate a new command position every sample. To prevent this problem, set the command position when the motion is in an IDLE state and the motor's disableAction is configured for no action.

mpiAxisCommandPositionSet(...) Error Check
The mpiAxisCommandPositionSet(...) error check has been extended. If the controller is updating the axis's command position when mpiAxisCommandPositionSet(...) is called, MPIAxisMessageCOMMAND_NOT_SET will be returned. mpiAxisCommandPositionSet(...) checks for the following conditions:

- Axis is in a STOPPING, STOPPED, or MOVING state.
- Any motor associated with the axis has the disableAction configuration set to MEIMotorDisableActionCMD_EQ_ACT and the motor's Amp Enable is disabled.
- If the command position read from the controller does not match the requested position.

MPIAxisMessageNOT_MAPPED_TO_MS

An axis is not mapped to the motion supervisor. This message code is returned by mpiMotionDelete(…), mpiMotionAxisList(…), or mpiMotionAxisRemove(…) when an axis is associated with a motion object, but not mapped to a motion supervisor. To correct this problem, map the axes to the motion supervisor in the controller by calling: mpiMotionAction(…) with MEIActionMAP or MPIActionRESET, mpiMotionStart(…), mpiMotionModify(…), or mpiMotionEventNotifySet(…).

See Also

Configuration of IN_POSITION and DONE Events after STOP or E-STOP Events

Two fields, **settleOnStop** and **settleOnEstop** are incorporated into the MPIAxisInPosition{} structure. These fields control the generation and use of IN_FINE_POSITION, and DONE status bits and events. A value of FALSE in these fields causes the IN_FINE_POSITION to be held false after STOP (or E-STOP) events and DONE to be based solely on command velocity (i.e. DONE is true as soon as the command velocity reaches 0). A value of TRUE in these fields causes IN_FINE_POSITION and DONE to be calculated in the same manner as that for normal motion, except that the position where the command velocity reaches zero is used for a target rather than the original Target Position.

The following table shows the generation of these status bits with settleOnStop (settleOnEstop) = FALSE (the default value):

Motion Status	After S-curve or Trapezoidal Move	During Velocity Move	After STOP (E-STOP)	After ABORT
IN_FINE_POSITION	Based on target distance (see note 3)	FALSE	FALSE	FALSE
IN_COARSE_POSITION	Based on target distance	FALSE	FALSE	FALSE
AT_TARGET	TRUE when command = target	FALSE	FALSE	FALSE
DONE	TRUE if both TC and IN_FINE_POSITION are true	FALSE	TRUE when command velocity = 0	TRUE

The following table shows the generation of these status bits with settleOnStop (settleOnEstop) = TRUE:

Motion Status	After S-curve or Trapezoidal Move	During Velocity Move	After STOP (E-STOP)	After ABORT
IN_FINE_POSITION	Based on target distance (see note 3)	FALSE	Based on position error (see note 1)	FALSE
IN_COARSE_POSITION	Based on target distance	FALSE	FALSE	FALSE
AT_TARGET	TRUE when command = target	FALSE	FALSE	FALSE
DONE	TRUE if both TC and IN_FINE_POSITION are true	FALSE	Same as IN_FINE_ POSITION	TRUE

NOTE 1: IN_FINE_POSITION is based on four criteria:

- The trajectory has completed (see note 2).
- |command position – actual position| < fine position tolerance.
- Target velocity – actual velocity| < velocity tolerance (the default setting for velocity tolerance so large that this criteria is ignored).
- The above 3 criteria have been satisfied for the duration specified by the settling time parameter.

NOTE 2: The reference to "TC" above refers to TRAJECTORY_COMPLETE, an internal status that is set when all of the current motion segments (frames) have completed.

NOTE 3: The criteria used for calculation of IN_FINE_POSITION after s-curve or trapezoidal motion has changed to the following: (This is the same as the MPI-1 criteria.)

- The trajectory has completed (see note 2).
- |target position – actual position| < fine position tolerance.
- |command velocity – actual velocity| < velocity tolerance (the default setting for velocity tolerance so large that this criteria is ignored).
- The above 3 criteria have been satisfied for a duration specified by the settling time parameter.

Return to MPIAxisInPosition