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.

| Error Messages |

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**ActualPositionGet32**	Gets the lower 32 bits of 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**CommandPositionGet32**	Gets the lower 32 bits of command position
mpiAxis**CommandPositionSet**	Set command position
mpiAxis**FlashConfigGet**	Get Axis flash config
mpiAxis**FlashConfigSet**	Set Axis flash config
mpiAxis**FlashOriginGet**	
mpiAxis**FlashOriginSet**	
meiAxis**FrameBufferStatus**	
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**EstopModify**
MEIAxis**FrameBufferStatus**
MPIAxis**InPosition**
MPIAxis**Master**
MPIAxis**MasterType**
MPIAxis**Message**

MEI**PreFilter**
MEI**PreFilterForm**

Constants

MEI**PreFilterCoeffsMAX**

MEI**PreFilterCountMAX**

mpiAxisCreate

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisCreate 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	

See Also

mpiAxisDelete | mpiAxisValidate

mpiAxisDelete

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisDelete 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	

See Also

mpiAxisCreate | mpiAxisValidate

mpiAxisValidate

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisValidate 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	

See Also

mpiAxisCreate | mpiAxisDelete

mpiAxisActualPositionGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

AxisCommandPositionSet | Using the Origin Variable

mpiAxisActualPositionGet32

Declaration

```
long mpiAxisActualPositionGet32(MPIAxis  axis,
                                double   *actual)
```

Required Header: stdmpi.h
Change History: Added in the 03.04.00

Description

mpiAxisActualPositionGet32 gets the lower 32 bits of the actual position of an Axis (***axis***) and puts it in the location pointed to by ***actual***. The command and actual positions are stored in the controller as 64 bit values (2x 32bit words). Use mpiAxisActualPositionGet(…) to read the full position value. Internally, the MPI performs several reads and operations to transfer the full 64 bit position value. For applications that need optimum performance and if the position range is less than 32 bits, then use mpiAxisActualPositionGet32(…).

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

Return Values	
MPIMessageOK	

See Also

mpiAxisActualPositionGet | AxisCommandPositionSet | Using the Origin Variable

mpiAxisActualPositionSet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

AxisCommandPositionSet | Using the Origin Variable | Controller Positions

mpiAxisActualVelocity

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisActualVelocity 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	

See Also

mpiAxisConfigGet

Declaration

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

Required Header: stdmpi.h
Change History: Modified in the 03.03.00

Description

mpiAxisConfigGet 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 an Axis object.
***config**	pointer to the MPIAxisConfig structure
***external**	pointer to an external. See remarks below

Return Values	
MPIMessageOK	

Remarks

For XMP and ZMP 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);
}
```

See Also

MPIAxisConfig | mpiAxisConfigSet | MEIAxisConfig

mpiAxisConfigSet

Declaration

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

Required Header: stdmpi.h
Change History: Modified in the 03.03.00

Description

mpiAxisConfigSet 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 an Axis object.
***config**	pointer to the MPIAxisConfig structure
***external**	pointer to an external. See remarks below

Return Values	
MPIMessageOK	

Remarks

For XMP and ZMP 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);
}
```

See Also

mpiAxisConfigGet | MEIAdcConfig | MEIAxisConfig

mpiAxisCommandPositionGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisCommandPositionGet 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 an Axis object.
***command**	a pointer to the Axis command position returned by the method

Return Values	
MPIMessageOK	

See Also

mpiAxisCommandPositionSet | Controller Positions

mpiAxisCommandPositionGet32

Declaration

```
long mpiAxisCommandPositionGet32(MPIAxis   axis,
                                 double    *command)
```

Required Header: stdmpi.h
Change History: Added in the 03.04.00

Description

mpiAxisCommandPositionGet32 gets the lower 32 bits of the command position of an Axis (***axis***) and puts it in the location pointed to by ***command***. The command and actual positions are stored in the controller as 64 bit values (2x 32bit words). Use mpiAxisCommandPositionGet(…) to read the full position value. Internally, the MPI performs several reads and operations to transfer the full 64 bit position value. For applications that need optimum performance and if the position range is less than 32 bits, then use mpiAxisCommandPositionGet32(…).

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

Return Values	
MPIMessageOK	

See Also

mpiAxisCommandPositionGet | mpiAxisCommandPositionSet | Controller Positions

mpiAxisCommandPositionSet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisCommandPositionSet sets the value of the command position of an Axis (***axis***) from ***command***. The motor will servo directly to the new command position the next servo sample after the new command position is set. This change in position will not be gradual or controlled, as it is in an mpiMotionStart(...). Use mpiMotionStart(...) and/or mpiMotionModify(...) for controlled, gradual motion.

mpiAxisCommandPositionSet truncates ***command*** to an integer value before sending the new position to the controller. If a different type of rounding is desired, then it should be implemented by the application prior to calling **AxisCommandPositionSet**.

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 will cause the axis to jump. See the discussion of the Axis Origin before using the mpiAxisActualPositionSet(...) and mpiAxisCommandPositionSet(...) methods. The origin is not changed when mpiAxisCommandPositionSet(...) is called. The change is made directly to the command position.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	if ***command*** lies outside the range of signed 32-bit integers: [-2147483648, 2147483647]
MPIAxisMessageCOMMAND_NOT_SET	

See Also

MEIMotorDisableAction | AxisActualPositionSet | AxisCommandPositionSet | MPIAxisMessage | Controller Positions

mpiAxisFlashConfigGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisFlashConfigGet 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 ***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**	a handle to the Flash object
***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	

See Also

MEIFlash | mpiAxisFlashConfigSet | MEIAxisConfig

mpiAxisFlashConfigSet

Declaration

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

Required Header: stdmpi.h

Description

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

Remarks

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	

See Also

MEIFlash | mpiAxisFlashConfigGet | MEIAxisConfig

mpiAxisFlashOriginGet

Declaration

```
long mpiAxisFlashOriginGet(MPIAxis   axis,
                           void      *flash,
                           double    *origin)
```

Required Header: stdmpi.h
Change History: Added in the 03.04.00

Description

mpiAxisFlashOriginGet reads the flash value of the origin for an Axis and writes it into the location pointed to by ***origin***. The ***flash*** origin value is useful for applying an offset value to the controller's actual position with absolute feedback devices.

axis	a handle to the Axis object.
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***origin**	pointer to the Origin value returned by the method.

Return Values	
MPIMessageOK	

See Also

mpiAxisFlashOriginSet | mpiAxisOriginGet | Using the Origin Variable | Controller Positions

mpiAxisFlashOriginSet

Declaration

```
long mpiAxisFlashOriginSet(MPIAxis   axis,
                           void      *flash,
                           double    origin)
```

Required Header: stdmpi.h
Change History: Added in the 03.04.00

Description

mpiAxisFlashOriginSet writes the flash origin for an Axis using the origin value. The flash origin value is useful for applying an offset value to the controller's actual position with absolute feedback devices.

axis	a handle to the Axis object.
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
origin	a value to which the Axis origin will be set.

Return Values	
MPIMessageOK	

See Also

mpiAxisFlashOriginGet | mpiAxisOriginSet | Using the Origin Variable | Controller Positions

meiAxisFrameBufferStatus

Declaration

```
long  meiAxisFrameBufferStatus(MPIAxis                    axis,
                               MEIAxisFrameBufferStatus   *status);
```

Required Header: stdmpi.h
Change History: Added in the 03.04.00

Description

meiAxisFrameBufferStatus reads an axis's frame buffer status and writes it into the structure pointed to by ***status***.

axis	a handle to the Axis object.
status	a pointer to the frame buffer status structure returned by the method.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessageHANDLE_INVALID	

Sample Code

```
void printAxisFrameBufferInfo(MPIAxis axis)
{
   MEIAxisFrameBufferStatus status;
   long returnValue = meiAxisFrameBufferStatus(axis,
                                               &status);
   msgCHECK(returnValue);

   printf("Size of frame buffer: %d\n", status.size);
   printf("Number of remaining frames: %d\n", status.frameCount);
   printf("Number of free frames %d\n",(status.size - status.
frameCount));
}
```

See Also

MEIAxisFrameBufferStatus

mpiAxisOriginGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisOriginGet 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	

See Also

mpiAxisOriginSet | Using the Origin Variable | Controller Positions

mpiAxisOriginSet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisOriginSet 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	

See Also

mpiAxisOriginGet | Using the Origin Variable | Controller Positions

mpiAxisPositionError

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisPositionError 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	

See Also

mpiAxisCommandPositionGet | mpiAxisActualPositionGet | Controller Positions

mpiAxisStatus

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisStatus 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.

Remarks

external should always be set to NULL.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiAxisCommandPositionGet | mpiAxisActualPositionGet | Controller Positions

mpiAxisTrajectory

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisTrajectory 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	

See Also

mpiMotionStart | mpiMotionModify | MPITrajectory

mpiAxisEventNotifyGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisEventNotifyGet 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

Remarks

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	

See Also

MEIEventNotifyData | MEIEventStatusInfo | mpiAxisEventNotifySet

mpiAxisEventNotifySet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisEventNotifySet 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

Remarks

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	

See Also

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

mpiAxisEventReset

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiControlEventReset | mpiMotionEventReset | mpiMotorEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | meiSqNodeEventReset

Event Notification Methods

mpiAxisMemory

Declaration

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

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiAxisMemoryGet | mpiAxisMemorySet

mpiAxisMemoryGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisMemoryGet 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	

See Also

mpiAxisMemory | mpiAxisMemorySet

mpiAxisMemorySet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisMemorySet 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	

See Also

mpiAxisMemory | mpiAxisMemoryGet

mpiAxisControl

Declaration

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

Required Header: stdmpi.h

Description

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

axis	a handle to an Axis object.

Return Values	
MPIHandleVOID	

See Also

mpiAxisFilterMapGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisFilterMapGet 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. Ex: 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	

See Also

mpiAxisFilterMapSet

mpiAxisFilterMapSet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisFilterMapSet 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	

See Also

mpiAxisFilterMapGet | MPIObjectMap

mpiAxisMotorMapGet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisMotorMapGet 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. Ex: 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 mpiAxisFilterMap and mpiFilterMotorMap.

Return Values	
MPIMessageOK	

See Also

mpiAxisFilterMapGet | MPIObjectMap

mpiAxisNumber

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisNumber 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	

See Also

MPIAxisConfig / MEIAxisConfig

Definition: MPIAxisConfig

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

Change History: Modified in the 03.03.00.

Description

estopModify	See MPIAxisEstopModify.
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.

Definition: MEIAxisConfig

```
typedef struct MEIAxisConfig {
    char                      userLabel[MEIObjectLabelCharMAX+1];
                                 /* +1 for NULL terminator */
    long                      *FeedbackDeltaPtr[MEIXmpAxisPosInputs];
    MEIXmpAxisPreFilter   PreFilter;
    MEIXmpAxisGear        Gear;
    MEIXmpAxisGantryType  GantryType;
}MEIAxisConfig;
```

Change History: Modified in the 03.04.00. Modified in the 03.03.00.

Description

userLabel - consists of 16 characters that are used to label the axis object for user identification purposes. The userLabel field is NOT used by the controller.

***FeedbackDeltaPtr** - Pointer to the position feedback delta, which is the difference in the feedback position between two sample periods, calculated by the controller.

PreFilter

- 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

GantryType -

```
typedef enum {
   MEIXmpAxisGantryTypeNONE = 0,
   MEIXmpAxisGantryTypeLINEAR = 1,
   MEIXmpAxisGantryTypeTWIST = 2
} MEIXmpAxisGantryType;
```

- **MEIXmpAxisGantryTypeNONE** - is the default. No gantry enabled.
- **MEIXmpAxisGantryTypeLINEAR** - is used to add the axis' two feedback values.
- **MEIXmpAxisGantryTypeTWIST** - is used to subtract the axis' two feedback values.

See Also

mpiAxisConfigGet | mpiAxisConfigSet | MPIAxisInPosition

MPIAxisEstopModify

Definition

```
typedef struct MPIAxisEstopModify {
    float    deceleration;
    float    decelerationJerk;
    float    jerkPercent;
} MPIAxisEstopModify
```

Change History: Added in the 03.03.00

Description

MPIAxisEstopModify is used with mpiAxisConfigGet(...) and mpiAxisConfigSet(...) as part of the MPIAxisConfig structure. This structure can be used to configure the deceleration and jerk applied to a motor when an EStopModify event occurs.

Any of the limits can be configured to generate an EStopModify instead of a standard EStop (a standard EStop stops the motor by stepping down the feedrate until it reaches 0.0).

See mpiMotorEventConfigSet(...) and mpiMotorEventConfigGet(...).

NOTE: Standard firmware uses jerkPercent and does not support decelerationJerk. See the MPITrajectory structure documentation.

deceleration	Specifies the Deceleration applied to the motor when an EStopModify occurs. Units are counts/sec^2.
decelerationJerk	Specifies the Jerk applied to the motor when an EstopModify occurs. Units are counts/sec^3. Jerk moves specified in counts/sec^3 are not supported in the standard firmware. Please contact MEI if this feature is required in your application.
jerkPercent	Specifies the JerkPercent applied to the motor when an EstopModify occurs. Units are in percent. Range is 0.0 to 100.0.

See Also

MPIAxisConfig | mpiMotorEventConfigSet | mpiMotorEventConfigGet | MPIAction

MEIAxisFrameBufferStatus

Definition

```
typedef struct MEIAxisFrameBufferStatus {
    long   size;
    long   frameCount;
} MEIAxisFrameBufferStatus;
```

Change History: Added in the 03.04.00.

Description

MEIAxisFrameBufferStatus provides status information of the frame buffer for a specified axis.

size	The value specifies the size (maximum number of frames) of the frame buffer for the given axis. This value is controlled by mpiControlConfigGet /Set under axisFrameCount. The default size the frame buffer (axisFrameCount) is 128.
frameCount	The value is equal to the number of frames on the controller that still need to be executed. At the default buffer size (128 frames), the range for frameCount is 0 to 127.

See Also

meiAxisFrameBufferStatus | mpiControlConfigGet | mpiControlConfigSet

MPIAxisInPosition

Definition

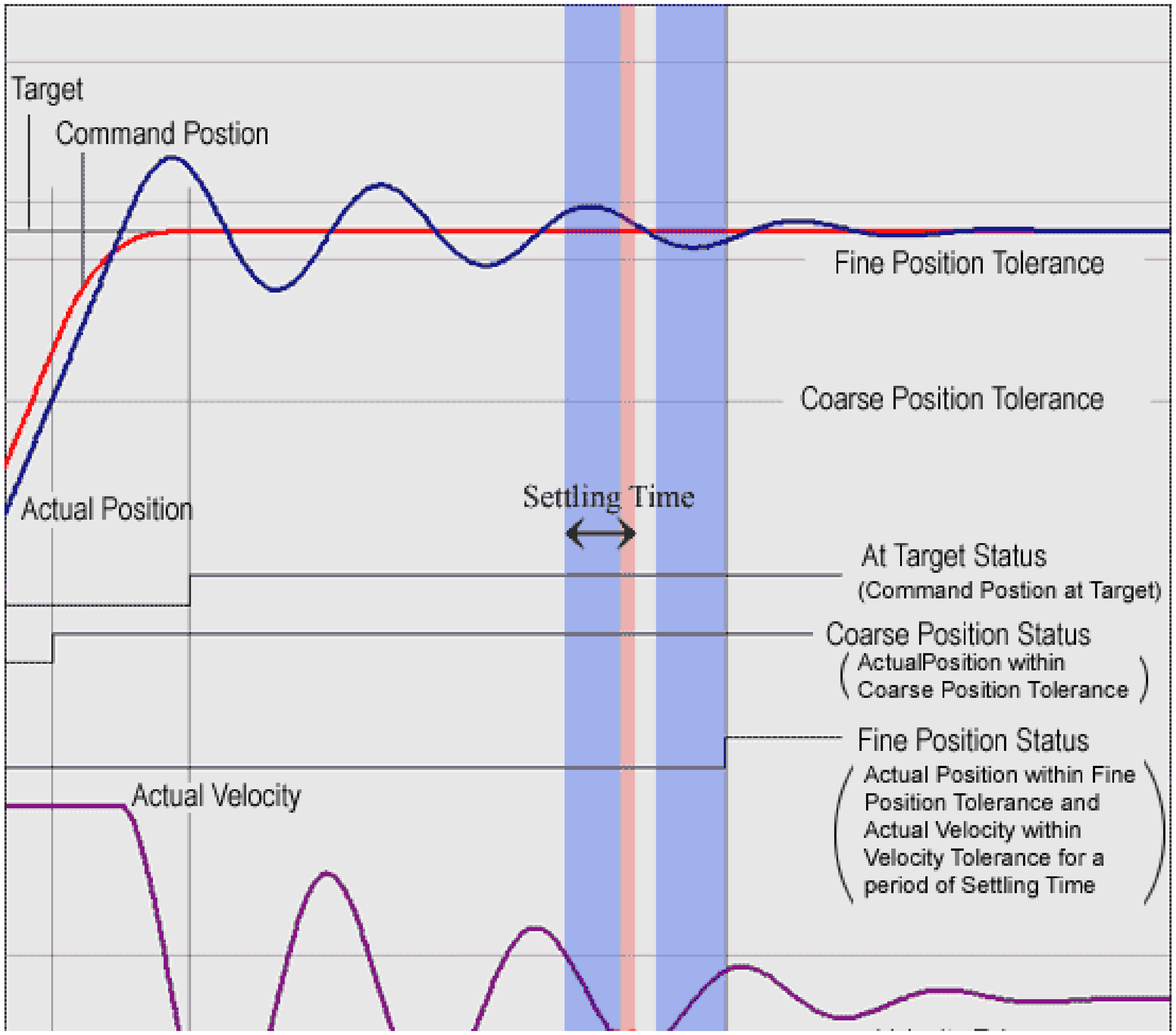
```
typedef struct MPIAxisInPosition {
    struct {
        float   positionFine;
        long    positionCoarse;
        float   velocity;
    } tolerance;
        float   settlingTime;  /* seconds */
        long    settleOnStop;
        long    settleOnEstop
        long    settleOnEstopCmdEqAct;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

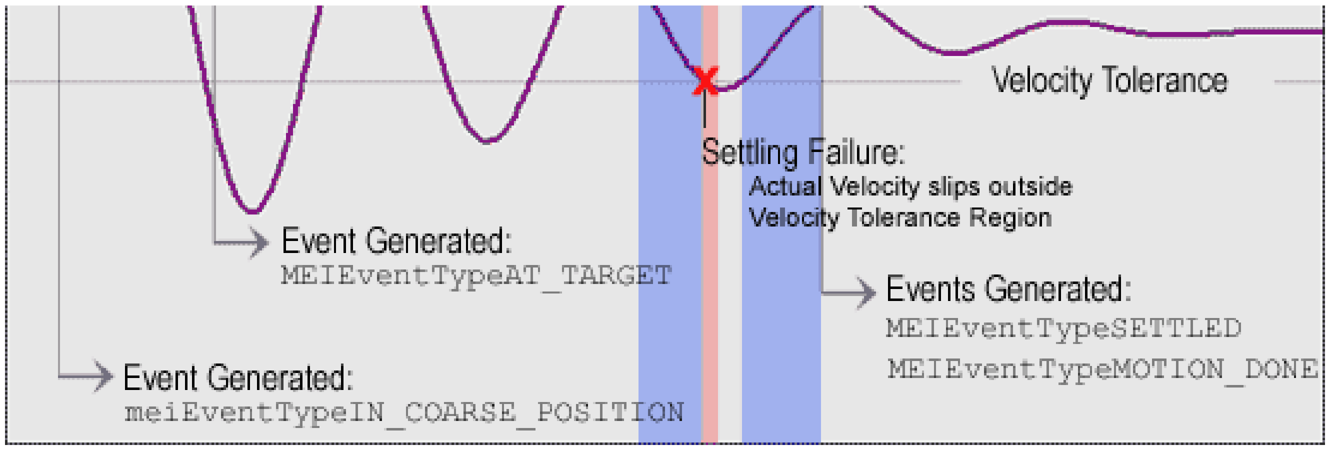
Description

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. **NOTE**: Always enter a Tolerance Velocity value that is a multiple of the controller sample rate. The controller will then receive velocity in counts/ controller sample. 1 Counts/second = (1 counts/second) * (1/sample rate(Hz)) = (1/sample rate) counts/controller sample

	NOTE: Value is truncated to the next smallest integer. **Example**: With a sample rate of 2000Hz, • a Tolerance Velocity value of 500 counts/second = 0.25 = 0 count/controller sample (after truncation). • a Tolerance Velocity value of 2000 counts/second = 1 count/controller sample.
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: • The settling criteria are satisfied AND. • The stop duration for the axis' Motion Supervisor has elapsed. • 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: • The settling criteria are satisfied AND. • The Estop duration for the axis' Motion Supervisor has elapsed. • 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 Related Events: How Motion Related Events are Generated

Configuration of IN_POSITION and Done Events after STOP or E_STOP Events

MPIAxisMaster

Definition

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

Description

MPIAxisMaster 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
MPIAxisMasterTypeNONE	Not used	Not used

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

Definition

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

Change History: Modified in the 03.04.00. Modified in the 03.03.00.

Description

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

Fields	Number	Address
MPIAxisMasterTypeNONE	Not used	Not used
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

Definition

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

Description

MPIAxisMessage 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(...), mpiMotionAxisListGet(...), 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

MEIPreFilter

Definition

```
typedef struct MEIPreFilter {
    long                    axisNumber;
    MEIPreFilterForm        form;
    long                    length;
    long                    coeff[MEIPreFilterCoeffsMAX];
} MEIPreFilter;
```

Change History: Added in the 03.04.00.

Description

PreFilters are used to filter motion trajectories. The command positions generated by the controller firmware during a move are passed through the filter before being used as set points by the control algorithm. PreFilters are useful for removing unwanted frequencies from the motion profile or for smoothing out motion generated by joysticks or other manual input devices.

Two forms of PreFilters are supported: BOXCAR and SHAPING. The BOXCAR filter is a simple averager where the output of the filter is the of average a number of previous command positions. The number of points is determined by the length parameter. For BOXCAR filters the coeff[] array is ignored.

The SHAPING PreFilter passes the trajectory through a special filter type patented by **Convolve, Inc.** ® (www.convolve.com). This filter can greatly enhance the performance of mechanical systems with resonances or flexible hardware. The length and coefficients of the SHAPING filter are generated by Convolve® for the specific system using the filter. See the Convolve website for information about the advantages of Input Shaping®.

axisNumber	The number of the axis to apply the filter.
form	The type of filter (NONE, BOXCAR, or SHAPING).
length	The filter length (number of stages).
coeff	Used only for SHAPING filters. The coefficients are generated by Convolve®, Inc. software (see www.convolve.com).

See Also

MEIPreFilterForm | MEIControlConfig

MEIPreFilterForm

Definition

```
typedef enum {
   MEIPreFilterFormNONE,
   MEIPreFilterFormBOXCAR,
   MEIPreFilterFormSHAPING,

} MEIPreFilterForm;
```

Change History: Added in the 03.04.00.

Description

MEIPreFilterForm specifies the filter types for filtering the command position profile produced by the controller.

MEIPreFilterFormNONE	The PreFilter is disabled and can be used by another axis.
MEIPreFilterFormBOXCAR	The axis' command positions are averaged using a BOXCAR averager. The length of the averager (number of samples to average) can be specified.
MEIPreFilterFormSHAPING	The axis' command positions are filtered using a special resonance elimination filter patented by Convolve®, Inc. See www.convolve.com for more information about SHAPING filters.

See Also

MEIPreFilter

MEIPreFilterCoeffsMAX

Definition

```
#define  MEIPreFilterCoeffsMAX      (MEIXmpMAX_PreCoeffs)
```

Change History: Added in the 03.04.00.

Description

MEIPreFilterCoeffsMAX defines the maximum number of coefficients for a SHAPING filter. (See MEIPreFilter description.)

See Also

MEIPreFilterCountMAX

MEIPreFilterCountMAX

Definition

```
#define  MEIPreFilterCountMAX      (MEIXmpMAX_PreFilters)
```

Change History: Added in the 03.04.00.

Description

MEIPreFilterCountMAX defines the maximum number of axes that can be filtered (with either BOXCAR or SHAPING filters).

See Also

MEIPreFilterCoeffsMAX