Motion Objects

Introduction

A **Motion** object manages a single axis or group of axes. Its primary function is to provide an interface to command movement on a coordinate system. It also provides status information about all the axes under its control, so motion can be either stopped or resumed in a controlled manner, especially in the event of error recovery. The Motion object is really a host-based object, with a corresponding Motion Supervisor object in the controller. The Motion Supervisor handles the real-time issues associated with axis data and status synchronization.

Some careful consideration should be given to Motion object (Motion Supervisor) to axis mapping. While it's possible to have multiple Motion objects share the same axes, only one Motion Supervisor can command motion to an axis at a time. Motion object to axis maps can be changed dynamically at any time, but Motion Supervisor to axis maps should NOT be changed when the axes are moving.

To learn more about usind MPI Motion Attributes and MEI Motion Attributes, click here.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiMotion**Create**	Create Motion object
mpiMotion**Delete**	Delete Motion object
mpiMotion**Validate**	Validate Motion object

Configuration and Information Methods

mpiMotion**ConfigGet**	Get Motion configuration
mpiMotion**ConfigSet**	Set Motion configuration
mpiMotion**FlashConfigGet**	Get Motion flash config
mpiMotion**FlashConfigSet**	Set Motion flash config
mpiMotion**ParamsGet**	Get Motion parameters
mpiMotion**ParamsSet**	Set Motion parameters
meiMotion**ParamsValidate**	Validate Motion parameters
mpiMotion**PositionGet**	Get position parameters of all axes associated with Motion
mpiMotion**PositionSet**	Set position parameters of all axes associated with Motion
mpiMotion**Status**	Get Motion status
mpiMotion**Trajectory**	Get trajectories for all Axis associated with Motion

Event Methods

mpiMotion**EventNotifyGet**	Get event mask
mpiMotion**EventNotifySet**	Set event mask
mpiMotion**EventReset**	Reset events specified in event mask

Action Methods

mpiMotion**Action**	Perform an action on a Motion
mpiMotion**Modify**	Modify parameters of Motion while it is executing
mpiMotion**Start**	Start Motion (idle state > moving state)

Memory Methods

mpiMotion**Memory**	Set address to be used to access Motion memory
mpiMotion**MemoryGet**	Get bytes of Motion memory and place it into application memory
mpiMotion**MemorySet**	Put (set) bytes of application memory into Motion memory

Relational Methods

mpiMotion**Control**	Return handle of Control object associated with Motion
mpiMotion**Number**	Get index of Motion
mpiMotion**Axis**	Return handle of axis by index number
mpiMotion**AxisAppend**	Append axis to list
mpiMotion**AxisCount**	Return number of axes in list
mpiMotion**AxisFirst**	Return handle to first axis in list
mpiMotion**AxisIndex**	Return index value of an axis in list
mpiMotion**AxisInsert**	Insert axis into list associated with Motion
mpiMotion**AxisLast**	Return handle to last axis in list
mpiMotion**AxisListGet**	Get list of axes associated with Motion
mpiMotion**AxisListSet**	Create a list of axes associated with Motion
mpiMotion**AxisNext**	Get handle to next axis in list
mpiMotion**AxisPrevious**	Get handle to previous axis in list
mpiMotion**AxisRemove**	Remove handle to axis in list

Data Types

MPIMotion**Attr** / MEIMotion**Attr**

MEIMotion**AttrHold**

MEIMotion**AttrHoldSource**

MEIMotion**AttrHoldType**

MPIMotion**AttrMask** / MEIMotion**AttrMask**

MEIMotion**AttrOutput**

MEIMotion**AttrOutputType**

MPIMotion**Attributes** / MEIMotion**Attributes**

MPIMotion**BESSEL**

MPIMotion**BSPLINE**

MPIMotion**Config** / MEIMotion**Config**

MPIMotion**DecelTime**

MEIMotion**Frame**

MEIMotion**FrameBufferStatus**

Macros

mpiMotionCreate

Declaration

```
const MPIMotion mpiMotionCreate(MPIControl control,
                                long       number,
                                MPIAxis    axis)
```

Required Header

stdmpi.h

Description

MotionCreate creates a Motion object associated with the motion supervisor identified by ***number*** located on motion controller ***control***. *MotionCreate* is the equivalent of a C++ constructor.

If ***number*** is **-1**, MotionCreate selects the next unused motion supervisor. The ***axis*** parameter specifies the initial element in the list of Axis objects which determine the coordinate system (axis may be MPIHandleVOID).

Return Values

handle	handle to a Motion object
MPIHandleVOID	if the object could not be created

See Also

mpiMotionDelete | mpiMotionValidate

mpiMotionDelete

Declaration

```
long mpiMotionDelete(MPIMotion motion)
```

Required Header stdmpi.h

Description

MotionDelete deletes a Motion object (***motion***) and invalidates its handle. *MotionDelete* is the equivalent of a C++ destructor.

Deleting a Motion object does not delete any of the Axis objects in the coordinate system.

Return Values	
MPIMessageOK	if *MotionDelete* successfully deletes a Motion object and invalidates its handle

See Also mpiMotionCreate | mpiMotionValidate

mpiMotionValidate

Declaration `long mpiMotionValidate(MPIMotion motion)`

Required Header stdmpi.h

Description **MotionValidate** validates the Motion object (***motion***) and its handle. Always call *mpiMotionValidate* after creating a new Motion object.

Return Values	
MPIMessageOK	if Motion is a handle to a valid object.

See Also mpiMotionCreate | mpiMotionDelete

mpiMotionConfigGet

Declaration

```
long mpiMotionConfigGet(MPIMotion          motion,
                        MPIMotionConfig    *config,
                        void               *external)
```

Required Header

stdmpi.h

Description

MotionConfigGet gets the configuration of a Motion object (***motion***) and puts (writes) it in the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

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

Return Values

MPIMessageOK	if *MotionConfigGet* successfully gets the configuration of a Motion object and writes it into the structure(s)

See Also

mpiMotionConfigSet | MEIMotionConfig

mpiMotionConfigSet

Declaration

```
long mpiMotionConfigSet(MPIMotion        motion,
                        MPIMotionConfig  *config,
                        void             *external)
```

Required Header

stdmpi.h

Description

MotionConfigSet sets (writes) the configuration of a Motion object (***motion***) using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

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

Return Values

MPIMessageOK	if *MotionConfigSet* successfully writes the configuration of a Motion object using data from the structure(s)

See Also

mpiMotionConfigGet | MEIMotionConfig

mpiMotionFlashConfigGet

Declaration

```
long mpiMotionFlashConfigGet(MPIMotion        motion,
                             void             *flash,
                             MPIMotionConfig  *config,
                             void             *external)
```

Required Header

stdmpi.h

Description

MotionFlashConfigGet gets the flash configuration for a Motion object (motion) and puts (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 Motion's flash configuration information in ***external*** is in addition to the Motion's flash configuration information in ***config***, i.e., the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL). The implementation-specific ***flash*** argument is used to access flash memory.

XMP Only	***external*** either points to a structure of type **MEIMotionConfig{}** 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 *MotionFlashConfigGet* successfully gets the Motion's flash configuration and writes it into the structure(s).

See Also

mpiMotionFlashConfigSet

mpiMotionFlashConfigSet

Declaration

```
long mpiMotionFlashConfigSet(MPIMotion        motion,
                             void             *flash,
                             MPIMotionConfig  *config,
                             void             *external)
```

Required Header

stdmpi.h

Description

MotionFlashConfigSet sets (writes) the flash configuration of a Motion object (motion) 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 Motion's flash configuration information in ***external*** is in addition to the Motion's flash configuration information in config, i.e., the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL). The implementation-specific ***flash*** argument is used to access flash memory.

XMP Only

external either points to a structure of type MEIMotionConfig{} 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 *MotionFlashConfigSet* successfully sets (writes) the Motion's flash configuration using data from the structure(s)

See Also

MEIMotionConfig | MEIFlash | mpiMotionFlashConfigGet

mpiMotionParamsGet

Declaration

```
long mpiMotionParamsGet(MPIMotion        motion,
                        MPIMotionParams  *params)
```

Required Header

stdmpi.h

Description

MotionParamsGet reads the parameters of a Motion object (motion) and writes it to the structure pointed to by params. These motion parameters will be used if mpiMotionStart(…) is called with Null motion params.

motion	a handle to the Motion object
***params**	a pointer to the motion parameters structure returned by the method

Return Values

MPIMessageOK	if *MotionParamsGet* successfully returns the parameters associated with a Motion object
motion parameters	that are associated with a Motion object (***motion***). To set these motion parameters, call either mpiMotionParamsSet(...) or mpiMotionStart(...)

See Also

mpiMotionParamsSet | mpiMotionStart | meiMotionParamsValidate

mpiMotionParamsSet

Declaration

```
long mpiMotionParamsSet(MPIMotion        motion,
                        MPIMotionParams  *params)
```

Required Header stdmpi.h

Description

.**MotionParamsSet** sets the parameters of a Motion object (***motion***). These motion parameters will be used if mpiMotionStart(...) is called with a Null motion parameter.

If Motion (***motion***) is active, *MotionParamsSet* will set motion parameters "on-the-fly," i.e., at the first opportunity.

Return Values	
MPIMessageOK	if *MotionParamsSet* successfully sets the parameters of a Motion object

See Also mpiMotionStart | mpiMotionParamsGet

meiMotionParamsValidate

Declaration

```
long meiMotionParamsValidate(MPIMotion          motion,
                             MPIMotionType      type,
                             MPIMotionParams    *params,
                             MEIXmpAction       action,
                             long               *points)
```

Required Header

stdmei.h

Description

MotionParamsValidate validates the type-specific motion parameters pointed to by ***params***, using the coordinate system of ***motion***.

*If "*point" is*	*Then*
not Null	the number of points specified by ***params*** is written to the location (pointed to by ***points***)

Return Values

MPIMessageOK	if *MotionParamsValidate* successfully validates the type-specific motion parameters using the coordinate system of ***motion***

See Also

mpiMotionPositionGet

Declaration

```
long mpiMotionPositionGet(MPIMotion  motion,
                          double     *actual,
                          double     *command)
```

Required Header

stdmpi.h

Description

MotionPositionGet gets the actual and command position values for all axes associated with a Motion (***motion***). The ***actual*** and ***command*** arguments each point to an array with a size equal to the number of axes associated with ***motion***.

Return Values	
MPIMessageOK	if *MotionPositionGet* successfully gets the actual and command position values for all axes associated with a Motion object

See Also

mpiMotionPositionSet

mpiMotionPositionSet

Declaration

```
long mpiMotionPositionSet(MPIMotion motion,
                          double    *actual,
                          double    *command)
```

Required Header

stdmpi.h

Description

MotionPositionSet sets the actual and command position values for all axes associated with a Motion (***motion***). The ***actual*** and ***command*** arguments each point to an array with a size equal to the number of axes associated with ***motion***.

Return Values

MPIMessageOK	if *MotionPositionSet* successfully sets the actual and command position values for all axes associated with a Motion object

See Also

mpiMotionPositionGet

mpiMotionStatus

Declaration

```
long mpiMotionStatus(MPIMotion   motion,
                     MPIStatus   *status,
                     void        *external)
```

Required Header

stdmpi.h

Description

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

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

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

Return Values

MPIMessageOK	if *MotionStatus* successfully writes the Motion's status to the structure(s)

See Also

mpiMotionTrajectory

Declaration

```
long mpiMotionTrajectory(MPIMotion      motion,
                         MPITrajectory  *trajectory)
```

Required Header stdmpi.h

Description **MotionTrajectory** gets the trajectories for all axes associated with a Motion object (***motion***). The ***trajectory*** argument points to an array of MPITrajectory structures, with a size equal to the number of axes associated with the Motion object (***motion***).

Return Values	
MPIMessageOK	if *MotionTrajectory* successfully gets the trajectories for all axes associated with a Motion object

See Also

mpiMotionEventNotifyGet

Declaration

```
long mpiMotionEventNotifyGet(MPIMotion      motion,
                             MPIEventMask   *eventmask,
                             void           *external)
```

Required Header

stdmpi.h

Description

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

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

Event notification is enabled for event types specified in ***eventmask***, which is a bit mask generated **by the logical OR** of the MPIEventMask bits associated with the desired MPIEventType values.
Event notification is disabled for event types not specified in ***eventmask***.

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

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo
mpiMotionEventNotifySet

mpiMotionEventNotifySet

Declaration

```
long mpiMotionEventNotifySet(MPIMotion      motion,
                             MPIEventMask   eventmask,
                             void           *external)
```

Required Header

stdmpi.h

Description

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

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

Event notification is enabled for event types specified in ***eventMask***, a bit mask generated by the bitwise OR of the MPIEventMask bits *associated with* the desired MPIEventType values. 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.

XMP Only

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

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

Return Values

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

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo
mpiEventMaskMOTION | mpiEventMaskAXIS | mpiMotionEventNotifyGet

mpiMotionEventReset

Declaration

```
long mpiMotionEventReset(MPIMotion    motion,
                         MPIEventMask eventMask)
```

Required Header

stdmpi.h

Description

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

Return Values

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

See Also

mpiMotionAction

Declaration

```
long mpiMotionAction(MPIMotion motion,
                     MPIAction action)
```

Required Header

stdmpi.h

Description

MotionAction performs the specified action on the Motion object (motion).

The deceleration stop and Estop times can be set with:

mpiMotionConfigSet(MPIMotion **motion**, MPIMotionConfig ***config**, void ***external**);

where the structure MPIMotionConfig, contains the elements:

MPIMotionConfig.decelTime.stop /* seconds */
MPIMotionConfig.decelTime.eStop /* seconds */

If "action" is	*Then*
MPIActionABORT *or* **MPIActionE_STOP** *or* **MPIActionE_STOP_ABORT**	the Motion will perform an emergency stop and/or abort on all axes, and then change to the error state (MPIStateSTOPPING_ERROR to MPIStateERROR). Before starting another Motion, you must first make a call to mpiMotionAction(motion, MPIActionRESET).
MPIActionABORT	MotionAction will disable the PID control (or other algorithm), set the DAC output to the offset value, and disable the amp enable outputs. After the abort completes, the Motion will be set to the ERROR state .
MPIActionE_STOP	MotionAction will decelerate the axis (or axes) to a stop in the time specified by the “eStop” time. After the Motion stops, the Motion will be set to the ERROR state.
MPIActionE_STOP_ABORT	MotionAction will decelerate the axis (or axes) to a stop in the time specified by the “eStop” time. Next, MotionAction will generate an MPIActionABORT. After the abort completes, the Motion is set to the ERROR state. The E_STOP_ABORT first performs an E_STOP, and then performs an ABORT. The E_STOP decelerates the axis to a stop, then the ABORT disables PID control and disables the amp enable output.
MPIActionRESET	if the Motion (motion) is in the error state (MPIStateERROR), the Motion will change to the idle state (MPIStateIDLE), so that motion can be restarted. If the error state was caused by an Abort action, then the command position will be set equal to the actual position during the Reset.
MPIActionRESUME	if the Motion is in the idle state (MPIStateIDLE), it will change to the moving state (MPIStateMOVING), and the Motion will resume if it was stopped by a prior call to mpiMotionAction(motion, MPIActionSTOP).

MPIActionSTOP	if the Motion is in the moving state (MPIStateMOVING), it will performing a stop on all axes and change to the idle state (MPIStateSTOPPING to MPIStateIDLE). MPIActionSTOP decelerates the axis (or axes) to a stop in the time specified by the “stop” time. After the motion stops, the Motion is set to the ERROR state.

Return Values

MPIMessageOK	if *MotionAction* successfully performs the specified action on the Motion object

See Also

mpiMotionModify

Declaration

```
long mpiMotionModify(MPIMotion        motion,
                     MPIMotionType    type,
                     MPIMotionParams  *params)
```

Required Header

stdmpi.h

Description

MotionModify modifies the parameters of a Motion object (***motion***) if motion is in progress (MPIStateMOVING). The types of motion whose parameters can be modified while moving are MPIMotionTypeTRAPEZOIDAL, MPIMotionTypeS_CURVE, MPIMotionTypeVELOCITY, MPIMotionTypePT and MPIMotionTypePVT.

Use the MPIMotionAttrAUTO_START attribute to automatically start a motion profile if the MotionModify call is made too late (i.e., after the previous move has finished).

Return Values	
MPIMessageOK	if *MotionModify* successfully modifies the parameters of a Motion object
MPIMotionMessageIDLE	if no Motion was in progress when *MotionModify* was called. In order for *MotionModify* to work, there must be a Motion in progress.
MPIMotionMessageAUTO_START	if *MotionModify* was called when no motion was in progress and the Auto-Start attribute was specified. In this case, the MotionModify is automatically converted into a MotionStart.
MPIMotionMessagePROFILE_ERROR	if the controller cannot generate the motion profile (based on the specified motion parameters and attributes).
For more Returns, click here	

See Also

mpiMotionStart

Declaration

```
long mpiMotionStart(MPIMotion        motion,
                    MPIMotionType    type,
                    MPIMotionParams  *params)
```

Required Header

stdmpi.h

Description

MotionStart changes a Motion object (***motion***) from the idle state (MPIStateIDLE) to the moving state (MPIStateMOVING), by initiating a motion of the given ***type*** using the specified parameters (***params***).If ***params*** is Null, then the motion parameters that were set by the most recent call to mpiMotionParamsSet(...) will be used to define the motion.

The coordinate system is defined by the ordered list of Axis object(s) that have been associated with the Motion object (***motion***). There must be at least one Axis in the coordinate system.

Return Values

MPIMessageOK	*MotionStart* successfully changes a Motion object from the idle state to the moving state
MPIMotionMessageMOVING	if *MotionStart* was called when a Motion is in progress
For more Returns, click here	

See Also

mpiMotionParamsSet | MPIState | See diagram on how mpiMotionStart works

mpiMotionMemory

Declaration

```
long mpiMotionMemory(MPIMotion  motion,
                     void       **memory)
```

Required Header

stdmpi.h

Description

MotionMemory sets (writes) the address [that is used to access a Motion's (***motion***) memory] to the contents of ***memory***. This address (or an address calculated from it) is passed as the ***src*** argument to mpiMotionMemoryGet(...), and also as the dst argument to mpiMotionMemorySet(...).

Return Values

MPIMessageOK	if *MotionMemory* successfully writes the address (used to access a Motion's memory) to the contents of memory

See Also

mpiMotionMemoryGet | mpiMotionMemorySet

mpiMotionMemoryGet

Declaration

```
long mpiMotionMemoryGet(MPIMotion motion,
                        void      *dst,
                        void      *src,
                        long      count)
```

Required Header

stdmpi.h

Description

MotionMemoryGet copies ***count*** bytes of a Motion's (***motion***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *MotionMemoryGet* successfully copies ***count*** bytes of a Motion's memory to application memory

See Also

mpiMotionMemorySet | mpiMotionMemory

mpiMotionMemorySet

Declaration

```
long mpiMotionMemorySet(MPIMotion motion,
                        void      *dst,
                        void      *src,
                        long      count)
```

Required Header

stdmpi.h

Description

MotionMemorySet copies ***count*** bytes of application memory (starting at address src) to a Motion's (***motion***) memory (starting at address dst).

Return Values	
MPIMessageOK	if *MotionMemorySet* successfully copies ***count*** bytes of application memory to a Motion's memory

See Also

mpiMotionMemoryGet | mpiMotionMemory

mpiMotionControl

Declaration `const MPIControl mpiMotionControl(MPIMotion motion)`

Required Header stdmpi.h

Description **MotionControl** returns a handle to the Control object with which the motion is associated.

motion	a handle to the Motion object

Return Values	
MPIControl	handle to a Control object
MPIHandleVOID	if ***motion*** is invalid

See Also mpiMotionCreate | mpiControlCreate

mpiMotionNumber

Declaration

```
long mpiMotionNumber(MPIMotion  motion,
                     long       *number)
```

Required Header

stdmpi.h

Description

MotionNumber writes the index of a Motion object (***motion***, on the motion controller that the Motion object is associated with) to the contents of ***number***.

Return Values	
MPIMessageOK	if *MotionNumber* successfully writes the index of a Motion object to the contents of ***number***

See Also

mpiMotionAxis

Declaration

```
const MPIAxis mpiMotionAxis(MPIMotion motion,
                            long      index)
```

Required Header

stdmpi.h

Description

MotionAxis returns the element at the position on the list indicated by ***index***.

motion	a handle to the Motion object.
index	a position in the list.

Return Values

handle	to the ***index***th Axis of a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if ***index*** is less than 0 if ***index*** is greater than or equal to **mpiMotionAxisCount(motion)**
MPIMessageARG_INVALID	***index*** is a negative number.
MEIListMessageELEMENT_NOT_FOUND	***index*** is greater than or equal to the number of elements in the list.
MPIMessageHANDLE_INVALID	***motion*** is an invalid handle.

See Also

mpiMotionAxisAppend

Declaration

```
long mpiMotionAxisAppend(MPIMotion  motion,
                         MPIAxis    axis)
```

Required Header

stdmpi.h

Description

MotionAxisAppend appends an Axis (***axis***) to a Motion (***motion***).

motion	a handle to the Motion object.
axis	a handle to an Axis object.

Return Values

MPIMessageOK	if *MotionAxisAppend* successfully appends an Axis to a Motion object
MPIMessageHANDLE_INVALID	Either ***motion*** or ***axis*** is an invalid handle.
MPIMessageUNSUPPORTED	The list already contains the maximum number of elements (MEIXmpMAX_COORD_AXES). **-or-** ***motion*** and axis are on different controllers.
MPIMessageOBJECT_NOT_ENABLED	***axis*** is not an enabled axis.
MPIMessageOBJECT_ON_LIST	***axis*** is already on the list.
MPIMessageNO_MEMORY	Not enough memory was available.

See Also

mpiMotionAxisCount

Declaration

```
long mpiMotionAxisCount(MPIMotion motion)
```

Required Header

stdmpi.h

Description

MotionAxisCount returns the number of elements on the list.

motion	a handle to the Motion object.

Return Values

number	of Axes in a Motion (***motion***)
-1	if ***motion*** is invalid
0	if ***motion*** is empty

See Also

mpiMotionAxis | mpiMotionAxisAppend

mpiMotionAxisFirst

Declaration `const MPIAxis mpiMotionAxisFirst(MPIMotion motion)`

Required Header stdmpi.h

Description **MotionAxisFirst** return the first element in the list. This function can be used in conjuntion with mpiMotionAxisNext() in order to iterate through the list.

motion	a handle to the Motion object.

Return Values

handle	to the first Axis of a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if ***motion*** is empty
MPIMessageHANDLE_INVALID	***motion*** is an invalid handle.

See Also mpiMotionAxisNext | mpiMotionAxisLast

mpiMotionAxisIndex

Declaration

```
long mpiMotionAxisIndex(MPIMotion motion,
                        MPIAxis   axis)
```

Required Header

stdmpi.h

Description

MotionAxisIndex returns the position of "axis" on the list.

motion	a handle to the Motion object.
axis	a handle to an Axis object.

Return Values

index	of an Axis (***axis***) in a Motion (***motion***)
-1	if ***motion*** is invalid if the Axis (axis) was not found in the Motion (***motion***)

See Also

mpiMotionAxis

mpiMotionAxisInsert

Declaration

```
long mpiMotionAxisInsert(MPIMotion motion,
                         MPIAxis   axis,
                         MPIAxis   insert)
```

Required Header

stdmpi.h

Description

MotionAxisInsert inserts an Axis (***insert***) in a Motion (***motion***), just after the specified Axis (***axis***).

Return Values	
MPIMessageOK	if *MotionAxisInsert* successfully inserts an Axis (***insert***) in a Motion (***motion***) following a specified Axis (***axis***)

See Also

mpiMotionAxis

mpiMotionAxisLast

Declaration

```
const MPIAxis mpiMotionAxisLast(MPIMotion motion)
```

Required Header

stdmpi.h

Description

MotionAxisLast returns the last element in the list. This function can be used in conjuntion with mpiMotionAxisPrevious() in order to iterate through the list backwards.

motion	a handle to the Motion object.

Return Values

handle	to the last Axis of a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if ***motion*** is empty
MPIMessageHANDLE_INVALID	Either ***motion*** or ***axis*** is an invalid handle.

See Also

mpiMotionAxisPrevious

mpiMotionAxisListGet

Declaration

```
long mpiMotionAxisListGet(MPIMotion  motion,
                          long       *axisCount,
                          MPIAxis    *axisList)
```

Required Header

stdmpi.h

Description

MotionAxisListGet gets the coordinate system of a Motion object (***motion***).

MotionAxisListGet writes the number of axes (in the coordinate system) to a location (pointed to by ***axisCount***), and also writes an array (of ***axisCount*** Axis handles) to another location (pointed to by ***axisList***).

Return Values

MPIMessageOK	if *MotionAxisListGet* successfully gets the coordinate system of a Motion object

See Also

mpiMotionAxisListSet | mpiMotionAxis

mpiMotionAxisListSet

Declaration

```
long mpiMotionAxisListSet(MPIMotion  motion,
                          long       axisCount,
                          MPIAxis    *axisList)
```

Required Header

stdmpi.h

Description

MotionAxisListSet creates a coordinate system of ***axisCount*** dimensions, using the Axis handles specified by ***axisList***. Any existing coordinate system is completely replaced.

The ***axisList*** parameter is the address of an array of ***axisCount*** Axis handles, or is ***NULL*** (if ***axisCount*** is equal to zero).

A coordinate system may also be created incrementally (i.e. one Axis at a time) by using the append and/or insert methods described in this section. The initial Axis of a coordinate system may be specified using the ***axis*** parameter of mpiMotionCreate(...). The list methods in this section may be used to examine and manipulate a coordinate system (i.e. axis list) regardless of how it was created.

Return Values

MPIMessageOK	if *MotionAxisListSet* successfully creates a coordinate system

See Also

mpiMotionCreate | mpiMotionAxisListGet | mpiMotionAxis

mpiMotionAxisNext

Declaration

```
const MPIAxis mpiMotionAxisNext(MPIMotion motion,
                                MPIAxis   axis)
```

Required Header

stdmpi.h

Description

MotionAxisNext returns the next element following "axis" on the list. This function can be used in conjuntion with mpiMotionAxisFirst() in order to iterate through the list.

motion	a handle to the Motion object.
axis	a handle to an Axis object.

Return Values

handle	to the Axis just after the specified Axis (***axis***) in a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if the specified Axis (***axis***) is the last axis in a Motion (***motion***)
MPIMessageHANDLE_INVALID	Either ***motion*** or ***axis*** is an invalid handle.

See Also

mpiMotionAxisFirst | mpiMotionAxisPrevious

mpiMotionAxisPrevious

Declaration

```
const MPIAxis mpiMotionAxisPrevious(MPIMotion motion,
                                    MPIAxis   axis)
```

Required Header

stdmpi.h

Description

MotionAxisPrevious returns the previous element prior to "axis" on the list. This function can be used in conjuntion with mpiMotionAxisLast() in order to iterate through the list backwards.

motion	a handle to the Motion object.
axis	a handle to an Axis object.

Return Values

handle	to the Axis preceding the specified Axis (***axis***) in a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if the specfied Axis (***axis***) is the first Axis in a Motion (***motion***)
MPIMessageHANDLE_INVALID	Either ***motion*** or ***axis*** is an invalid handle.

See Also

mpiMotionAxisLast | mpiMotionAxisNext

mpiMotionAxisRemove

Declaration

```
long mpiMotionAxisRemove(MPIMotion  motion,
                         MPIAxis    axis)
```

Required Header stdmpi.h

Description **MotionAxisRemove** removes an Axis (***axis***) from a Motion (***motion***).

Return Values	
MPIMessageOK	if *MotionAxisRemove* successfully removes the Axis from the Motion object

See Also

MPIMotionAttr / MEIMotionAttr

MPIMotionAttr

```
typedef enum {
    MPIMotionAttrINVALID,

    MPIMotionAttrAPPEND,
    MPIMotionAttrAUTO_START,
    MPIMotionAttrDELAY,
    MPIMotionAttrMaskID,
    MPIMotionAttrELEMENT_ID,
    MPIMotionAttrRELATIVE,
    MPIMotionAttrSYNC_END,
    MPIMotionAttrSYNC_START,
    MPIMotionAttrCOUNT,
} MPIMotionAttr;
```

Description

The motion attributes are used to generate the motion attribute masks to enable features with mpiMotionStart(…) and mpiMotionModify(…). Please see MPIMotionAttrMask data type for more information.

MEIMotionAttr

```
typedef enum {

    MEIMotionAttrEVENT,
    MEIMotionAttrFINAL_VEL,
    MEIMotionAttrNO_REVERSAL,
    MEIMotionAttrHOLD,
    MEIMotionAttrOUTPUT,

    MEIMotionAttrCOUNT,
} MEIMotionAttr;
```

Description

The motion attributes are used to generate the motion attribute masks to enable features with mpiMotionStart(…) and mpiMotionModify(…). Please see MPIMotionAttrMask for more information.

See Also

MPIMotionAttrMask | mpiMotionStart | mpiMotionModify

MEIMotionAttrHold

MEIMotionAttrHold

```
typedef struct MEIMotionAttrHold {
    MEIMotionAttrHoldType       type;
    MEIMotionAttrHoldSource     source;
    float                       timeout;
} MEIMotionAttrHold;
```

Description

type	This value specifies the motion hold type. Please see MEIMotionAttrHoldType for more information.
source	This value specifies the motion hold conditions. Please see MEIMotionAttrHoldSource for more information.
timeout	This value specifies the motion hold expiration time (seconds). When the time exceeds the timeout value or the hold conditions are met, the motion profile will execute.

See Also

MEIMotionAttrHoldType | MEIMotionAttrHoldSource

MEIMotionAttrHoldSource

MEIMotionAttrHoldSource

```
typedef union {
    long        gate;
    struct {
        long    *input;
        long    mask;
        long    pattern;
    } input;
    struct {
        long    number;
        long    mask;
        long    pattern;
        } motor;
} MEIMotionAttrHoldSource;
```

Description

gate	This value specifies the control gate number when MEIMotionAttrHoldTypeGATE is used. Valid values are between 0 and 31. See meiControlGateGet/Set(…) for more information.
input (input)	This value specifies the input address when MEIMotionAttrHoldTypeINPUT is used.
mask (input)	This value specifies the AND mask when MEIMotionAttrHoldTypeINPUT is used.
pattern (input)	This value specifies the comparison pattern when MEIMotionAttrHoldTypeINPUT is used. The value at input.input is bit-wise ANDed with the input.mask and compared to the input.pattern.
number (motor)	This value specifies the motor number when MEIMotionAttrHoldTypeMOTOR is used.
mask (motor)	This value specifies the AND mask when MEIMotionAttrHoldTypeMOTOR is used.
pattern (motor)	This value specifies the comparison pattern when MEIMotionAttrHoldTypeMOTOR is used. The motor input word is bit-wise ANDed with the motor.mask and compared to the motor.pattern.

See Also

MEIMotionAttrHoldType | meiControlGateGet | meiControlGateSet

MEIMotionAttrHoldType

MEIMotionAttrHoldType

```
typedef enum {
    MEIMotionAttrHoldTypeINVALID,
    MEIMotionAttrHoldTypeNONE,
    MEIMotionAttrHoldTypeGATE,
    MEIMotionAttrHoldTypeINPUT,
    MEIMotionAttrHoldTypeMOTOR,
} MEIMotionAttrHoldType;
```

Description

MEIMotionAttrHoldTypeNONE	This type disables the hold trigger condition.
MEIMotionAttrHoldTypeGATE	This type configures a control gate for the hold trigger condition. See meiControlGateGet/Set(…) for more information.
MEIMotionAttrHoldTypeINPUT	This type configures a memory address for the hold trigger condition.
MEIMotionAttrHoldTypeMOTOR	This type configures a motor's input bit(s) for the hold trigger condition.

Remarks

These types specify the motion profile trigger condition. The hold trigger value is specified with the MEIMotionAttrHoldSource data type.

See Also

MEIMotionAttrHoldSource | MEIMotionAttrHold

MPIMotionAttrMask / MEIMotionAttrMask

MPIMotionAttrMask

```
typedef enum {
    MPIMotionAttrMaskAPPEND,
    MPIMotionAttrMaskAUTO_START,
    MPIMotionAttrMaskDELAY,
    MPIMotionAttrMaskID,
    MPIMotionAttrMaskELEMENT_ID,
    MPIMotionAttrMaskRELATIVE,
    MPIMotionAttrMaskSYNC_END,
    MPIMotionAttrMaskSYNC_START,
    MPIMotionAttrMaskALL,
} MPIMotionAttrMask;
```

Description

MPIMotionAttrMaskAPPEND	This mask enables the motion profile to be added to the end of a previous motion profile, in the controller's memory buffer. The APPENDed profile will begin execution after the previous profile has completed and the settling criteria has been met. The APPEND mask can be used with mpiMotionStart(…) or mpiMotionModify(…).
MPIMotionAttrMaskAUTO_START	This mask converts a mpiMotionModify(…) call to a mpiMotionStart(…) if the modify occurs after the previous motion profile has completed. If the previous profile had completed, then mpiMotionModify(…) will return an error code, MPIMotionMessageAUTO_START.
MPIMotionAttrMaskDELAY	This mask enables a time delay (seconds) before the motion profile begins. Please see MPIMotionAttributes for more information. This mask can be used with mpiMotionStart(…).
MPIMotionAttrMaskID	This mask enables an identification tag to be stored in the motion profile. Please see MPIMotionAttributes for more information. This mask can be used with mpiMotionStart(…) and mpiMotionModify(…).
MPIMotionAttrMaskELEMENT_ID	- This mask enables an identification tag to be stored in the path motion profiles. Please see MPIMotionAttributes for more information.
MPIMotionAttrMaskRELATIVE	This mask changes the profile target position from absolute to relative coordinates.
MPIMotionAttrMaskSYNC_END	This mask synchronizes the motion profiles for multiple axes so they will all end at the same time. Delays are inserted before the shorter profiles. When enabled, each axis will use its own MPITrajectory values.
MPIMotionAttrMaskSYNC_START	This mask synchronizes the motion profiles for multiple axes so they will all start at the same time. Delays are inserted after the shorter profiles. When enabled, each axis will use its own MPITrajectory values.

Remarks

The motion attribute masks are used to enable features with mpiMotionStart(…) and mpiMotionModify(…). The masks are **OR**ed with the MPIMotionType to enable each feature.

MEIMotionAttrMask

```
typedef enum {
    MEIMotionAttrMaskEVENT,
    MEIMotionAttrMaskFINAL_VEL,
    MEIMotionAttrMaskNO_REVERSAL,
    MEIMotionAttrMaskHOLD,
    MEIMotionAttrMaskOUTPUT,
    MEIMotionAttrMaskALL,
} MEIMotionAttrMask;
```

Description

MEIMotionAttrMaskEVENT	This mask allows the user to specify an MPIEventMask during a motion.
MEIMotionAttrMaskFINAL_VEL	This mask allows the user to specify a non-zero target velocity for point to point motion types.
MEIMotionAttrMaskNO_REVERSAL	This mask prevents a motion profile from changing direction.
MEIMotionAttrMaskHOLD	This mask prevents a motion profile from executing until the specified trigger conditions are met.
MEIMotionAttrMaskOUTPUT	This mask allows the user to set or clear bits during a motion.

See Also

mpiMotionStart | mpiMotionModify | MPIMotionType | MPITrajectory | MPIEventMask

MEIMotionAttrOutput

MEIMotionAttrOutput

```
typedef struct MEIMotionAttrOutput {
    MEIMotionAttrOutputType     type;
    union {
        long    *output;
        long    motor;
    } as;
    long        mask;
    long        pattern;
    long        pointIndex;     /* MEIMotionAttrMaskOUTPUT for path motion - point
                        index for turning on output - used with point lists */
} MEIMotionAttrOutput;
```

Description

type	This value specifies the output type to determine the output bits to be set or cleared.
***output**	This value specifies the memory address when MEIMotionAttrOutputTypeOUTPUT is used.
motor	This value specifies the motor number when MEIMotionAttrOutputTypeMOTOR is used.
mask	This value specifies the AND mask when MEIMotionAttrHoldTypeOUTPUT is used.
pattern	This value specifies the comparison pattern when MEIMotionAttrHoldTypeOUTPUT is used. The motor output word or output address is bit-wise ANDed with the mask and compared to the pattern.
pointIndex	This value specifies an index to a point, when multiple point motion is used.

See Also

MEIMotionAttrOutputType

MEIMotionAttrOutputType

MEIMotionAttrOutputType

```
typedef enum {
    MEIMotionAttrOutputTypeINVALID,
    MEIMotionAttrOutputTypeNONE,
    MEIMotionAttrOutputTypeMOTOR,
    MEIMotionAttrOutputTypeOUTPUT,
} MEIMotionAttrOutputType;
```

Description

MEIMotionAttrOutputTypeNONE	This type disables the setting/clearing of output bit(s) during motion.
MEIMotionAttrOutputTypeMOTOR	This type configures a motor's output bit(s) to be set or cleared during motion.
MEIMotionAttrOutputTypeOUTPUT	This type configures bit(s) at a memory address to be set or cleared during motion.

See Also

MEIMotionAttrOutput

MPIMotionAttributes / MEIMotionAttributes

MPIMotionAttributes

```
typedef struct MPIMotionAttributes {
    double      *delay;         /* MPIMotionAttrMaskDELAY */
    long        id;             /* MPIMotionAttrMaskID */
    long        *elementId;     /* MPIMotionAttrMaskELEMENT_ID */
} MPIMotionAttributes;
```

Description

delay	This array defines the delay time (seconds) before a motion profile begins execution.
id	This value defines the identity for a point to point motion.
elementId	This array defines the identity for each element of a path motion.

MEIMotionAttributes

```
typedef struct MEIMotionAttributes {
    MPIEventMask        eventMask;      /* MEIMotionAttrMaskEVENT */
    double              *finalVelocity; /* MEIMotionAttrMaskFINAL_VEL */
    MEIMotionAttrHold   *hold;          /* MEIMotionAttrMaskHOLD */
    long                *outputCount;   /* MEIMotionAttrMaskOUTPUT for path motion
                                       - number of outputs - per axis */
    MEIMotionAttrOutput *output;        /* MEIMotionAttrMaskOUTPUT for path and non
                                       path motion - outputs - per axis */
} MEIMotionAttributes;
```

Description

eventMask	This structure specifies the mask to enable event generation. See MPIEventMask for more information.
***finalVelocity**	This array specifies the target velocity for each axis when MEIMotionAttrMaskFINAL_VEL is used.
***hold**	This array specifies the hold configurations for each axis when MEIMotionAttrMaskHOLD is used.
***outputCount**	This array specifies the number of points per axis, to set/clear an output when MEIMotionAttrMaskOUTPUT is used.
***output**	This structure specifies the output configuration for each axis when MEIMotionAttrMaskOUTPUT is used.

See Also

MPIEventMask | MEIMotionAttrMask

MPIMotionBESSEL

MPIMotionBESSEL

```
typedef struct MPIMotionBESSEL {
    long        pointCount;
    double      *position;
    double      *time;

    MPIMotionPoint  point;
} MPIMotionBESSEL;
```

Description

positionCount	This value specifies the number of points.
***position**	This array stores the positions for the motion profile. There is one position value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The positions are interleaved in the array by the axis index.
***time**	This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.
point	This structure contains the point configuration. Please see MPIMotionPoint data type for more information.

See Also

MPIMotionPoint

MPIMotionBSPLINE

MPIMotionBSPLINE

```
typedef struct MPIMotionBSPLINE {
    long                pointCount;
    double              *position;
    double              *time;

    MPIMotionPoint      point;
} MPIMotionBSPLINE;
```

Description

pointCount	This value specifies the number of points.
***position**	This array stores the positions for the motion profile. There is one position value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The positions are interleaved in the array by the axis index.
***time**	This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.
point	This structure contains the point configuration. Please see MPIMotionPoint data type for more information.

See Also

MPIMotionPoint

MPIMotionConfig / MEIMotionConfig

MPIMotionConfig

```
typedef struct MPIMotionConfig {
    MPIMotionDecelTime  decelTime;
    float               normalFeedrate;
    float               pauseFeedrate;
}MPIMotionConfig;
```

Description

decelTime	This structure defines the deceleration time for Stop and E-Stop actions. Please see MPIMotionDecelTime data type documentation for more information.
normalFeedrate	This value defines the normal feed speed rate. The default value is 1.0 (100%).
pauseFeedrate	This value defines the feed speed rate for the Stop action. The default value is 0.0.

MEIMotionConfig

```
typedef struct  MEIMotionConfig {
    long        axisCount;
    long        axisNumber[MEIXmpMAX_COORD_AXES];
    double      blendLimit;
} MEIMotionConfig;
```

Description

axisCount	The current number of axes mapped to the Motion Supervisor on the controller.
axisNumber	This array specifies the axis numbers of the current Axis to Motion Supervisor mapping on the controller.
blendLimit	This value specifies the acceleration blending limit criteria. If the change direction is greater than 90 degrees (0 degrees = no change, 180 degrees = about face) the acceleration resulting from the blending can exceed the acceleration limit specified in the motion parameters (180 degrees = acceleration*2.0). The blendLimit allows the user to limit the sharpness of turns to be blended. If cosine (turn angle defined above) is greater than the blendLimit, the motion will be blended. A blend limit value of 0 exclude turns sharper than 90 degrees. 1.0 causes all moves to be blended. -1.0 allows no blending

See Also

MPIMotionDecelTime | mpiMotionModify

MPIMotionDecelTime

MPIMotionDecelTime

```
typedef struct MPIMotionDecelTime {
    float       stop;   /* seconds */
    float       eStop;  /* seconds */
} MPIMotionDecelTime;
```

Description

stop	This value defines the deceleration time (seconds) for a Stop action. The default value is .5 seconds.
eStop	This value defines the deceleration time (seconds) for an E-Stop action. The default value is .05 seconds.

See Also

MEIMotionFrame

MEIMotionFrame

```
typedef struct MEIMotionFrame {
    long                pointCount;
    MEIXmpFrame         *frame;
    MPIMotionPoint      point;
} MEIMotionFrame;
```

Description

pointCount	The value specifies the number of frames.
***frame**	This structure contains the frame data for each frame. See MEIXmpFrame for more information.
point	This structure specifies the points configuration. See MPIMotionPoint for more information.

See Also

MPIMotionPoint

MEIMotionFrameBufferStatus

MEIMotionFrameBufferStatus

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

Description

size	This value specifies the size of the controller's frame buffer.
frameCount	This value specifies the number of frames in the controller's frame buffer.

See Also

MPIMotionMessage / MEIMotionMessage

MPIMotionMessage

```
typedef enum {
    MPIMotionMessageMOTION_INVALID,
    MPIMotionMessageAXIS_NOT_FOUND,
    MPIMotionMessageAXIS_COUNT,
    MPIMotionMessageTYPE_INVALID,
    MPIMotionMessageATTRIBUTE_INVALID,
    MPIMotionMessageNOT_READY,
    MPIMotionMessageIDLE,
    MPIMotionMessageMOVING,
    MPIMotionMessageSTOPPING,
    MPIMotionMessageSTOPPING_ERROR,
    MPIMotionMessageERROR,
    MPIMotionMessageAUTO_START,
    MPIMotionMessagePROFILE_ERROR,
    MPIMotionMessagePATH_ERROR,
    MPIMotionMessageFRAMES_LOW,
    MPIMotionMessageFRAMES_EMPTY,
} MPIMotionMessage;
```

Description

MPIMotionMessageMOTION_INVALID	This message code occurs when the motion supervisor number is not valid.
MPIMotionMessageAXIS_NOT_FOUND	This message code occurs when an axis that is being removed is not a member of the motion supervisor.
MPIMotionMessageAXIS_COUNT	This message code occurs when the axis count exceeds MEIXmpMAX_COORD_AXES.
MPIMotionMessageTYPE_INVALID	This message code occurs when the MPIMotionType is not valid.
MPIMotionMessageATTRIBUTE_INVALID	This message code occurs when the MPIMotionAttrMask is not valid.
MPIMotionMessageNOT_READY	This message code occurs when the controller is in the MPIStateINIT and is not ready to generate motion profiles.
MPIMotionMessageIDLE	This message code occurs when mpiMotionModify(…) or mpiMotionAction(…) is called when the motion supervisor is in the IDLE state.
MPIMotionMessageMOVING	This message code occurs when mpiMotionStart(…) or mpiMotionAction(…) is called when the motion supervisor is in the MOVING state.
MPIMotionMessageSTOPPING	This message code occurs when mpiMotionStart(…), mpiMotionModify(…) or mpiMotionAction(…) is called when the motion supervisor is in the STOPPING state. Motion cannot be started or modified at this time.
MPIMotionMessageSTOPPING_ERROR	This message code occurs when mpiMotionStart(…), mpiMotionModify(…) or mpiMotionAction(…) is called when the motion supervisor is in the STOPPING_ERROR state. Motion cannot be started or modified at this time.
MPIMotionMessageERROR	This message code occurs when mpiMotionStart(…), mpiMotionModify(…) or mpiMotionAction(…) is called when the motion supervisor is in the ERROR state. Motion cannot be started or modified at this time.

MPIMotionMessageAUTO_START	This message code occurs when mpiMotionModify(…) was called when the motion supervisor was in the IDLE state and was automatically converted into an mpiMotionStart(…).
MPIMotionMessagePROFILE_ERROR	This message code occurs when the velocity must be reversed, but mpiMotionModify(…) was called with the NO_REVERSAL attribute.
MPIMotionMessagePATH_ERROR	This message code occurs when the path motion time slice is set to less than one controller sample period.
MPIMotionMessageFRAMES_LOW	XMP internal point list buffer is low.
MPIMotionMessageFRAMES_EMPTY	XMP internal point list ran out of frames.

MEIMotionMessage

```
typedef enum {
    MEIMotionMessageRESERVED0,
    MEIMotionMessageRESERVED1,
    MEIMotionMessageRESERVED2,
    MEIMotionMessageNO_AXES_MAPPED,
} MEIMotionMessage;
```

Description

MEIMotionMessageRESERVED0	Reserved for specialized use.
MEIMotionMessageRESERVED1	Reserved for specialized use.
MEIMotionMessageRESERVED2	Reserved for specialized use.
MEIMotionMessageNO_AXES_MAPPED	Returned from methods that require at least one axes to be associated to motion supervisor object.

See Also

MPIMotionType | MPIMotionAttrMask | mpiMotionModify | mpiMotionAction
mpiMotionStart

MPIMotionParams / MEIMotionParams

MPIMotionParams

```
typedef struct MPIMotionParams {
    MPIMotionJog          jog;

    MPIMotionPT           pt;
    MPIMotionPVT          pvt;
    MPIMotionSPLINE       spline;
    MPIMotionBESSEL       bessel;
    MPIMotionBSPLINE      bspline;

    MPIMotionSCurve          sCurve;
    MPIMotionSCurve          sCurveJerk;
    MPIMotionTrapezoidal     trapezoidal;

    MPIMotionVelocity     velocity;
    MPIMotionVelocity     velocityJerk;

    MPIMotionAttributes   attributes;

    void                  *external;
} MPIMotionParams;
```

Description

pt	This structure contains the parameters for a PT motion type. Please see MPIMotionPT data type for more information.
pvt	This structure contains the parameters for a PVT motion type. Please see MPIMotionPVT data type for more information.
spline	This structure contains the parameters for a SPLINE motion type. Please see MPIMotionSPLINE data type for more information.
bessel	This structure contains the parameters for a BESSEL motion type. Please see MPIMotionBESSEL data type for more information.
bspline	This structure contains the parameters for a BSPLINE motion type. Please see MPIMotionBSPLINE data type for more information.
sCurve	This structure contains the parameters for a S_CURVE motion type. Please see MPIMotionSCurve data type for more information.
velocity	This structure contains the parameters for a VELOCITY motion type. Please see MPIMotionVelocity data type for more information.
velocityJerk	This structure contains the parameters for a VELOCITY_JERK motion type. Please see MPIMotionVelocity data type for more information.
attributes	This structure contains the parameters for motion attributes. Please see MPIMotionAttributes data type for more information.

***external**	This points to an external structure, containing controller specific parameters. Presently, this only supports MEIMotionAttributes. Please see MEIMotionAttributes data type for more information.

MEIMotionParams

```
typedef         struct MEIMotionParams {
    MEIMotionFrame          frame;
    MPIMotionAttributes     attributes;
    MEIMotionAttributes     attributesMEI;
} MEIMotionParams;
```

Description

frame	This structure contains the frame data and points configuration. See MEIMotionFrame for more information.
attributes	This structure contains the motion attributes data. See MPIMotionAttributes for more information.
attributesMEI	This structure contains the motion attributes data. See MEIMotionAttributes for more information.

See Also

MEIMotionFrame | MPIMotionAttributes | MEIMotionAttributes

MPIMotionPoint

MPIMotionPoint

```
typedef struct MPIMotionPoint {
    long        retain;         /* FALSE => flush points after use */
    long        final;          /* FALSE => more points to come */
    long        emptyCount;     /* # of remaining points to trigger empty event, -1
=> disable */
} MPIMotionPoint;
```

Description

retain	This value specifies whether or not the points should be stored in a buffer after execution. If retain=0, the points will not be stored after execution. If retain=1, the points will be stored in a buffer. This feature is useful for backing up on path.
final	This value specifies if more points will be loaded. If final=1, no more points will be loaded. If final=0, more points can be loaded using mpiMotionModify(…) with the MPIMotionAttrMaskAPPEND attribute mask.
emptyCount	This value specifies the minimum number of points in the buffer. If the number of points in the buffer is below emptyCount, an E_STOP action will occur. When emptyCount is (-1), the buffer low trigger is disabled.

See Also

mpiMotionModify | MPIMotionAttrMaskAPPEND | MPIMotionPT

MPIMotionPT

MPIMotionPT

```
typedef          struct MPIMotionPT {
    long       pointCount;
    double     *position;
    double     *time;
    MPIMotionPoint       point;
} MPIMotionPT;
```

Description

pointCount	This value specifies the number of points.
position	This array stores the positions for the motion profile. There is one position value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The positions are interleaved in the array by the axis index.
time	This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.
point	This structure contains the point configuration. Please see MPIMotionPoint data type for more information.

See Also

MPIMotionPoint

MPIMotionPVT

MPIMotionPVT

```
typedef struct MPIMotionPVT {
    long                pointCount;
    double              *position;
    double              *velocity;
    double              *time;
    MPIMotionPoint      point;
} MPIMotionPVT;
```

Description

pointCount	This value specifies the number of points.
position	This array stores the positions for the motion profile. There is one position value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The positions are interleaved in the array by the axis index.
velocity	This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.
time	This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.
point	This structure contains the point configuration. Please see MPIMotionPoint data type for more information.

See Also

MPIMotionPoint

MPIMotionSCurve

MPIMotionSCurve

```
typedef struct MPIMotionSCurve {
    MPITrajectory   *trajectory;
    double          *position;
} MPIMotionSCurve;
```

Description

***trajectory**	This structure specifies the parameters for the motion profile.
***position**	This array specifies the target positions for the axes. There is one position for each axis.

See Also

MPIMotionSPLINE

MPIMotionSPLINE

```
typedef         struct MPIMotionSPLINE {
    long                pointCount;
    double              *position;
    double              *time;

    MPIMotionPoint      point;
} MPIMotionSPLINE;
```

Description

pointCount	This value specifies the number of points.
***position**	This array stores the positions for the motion profile. There is one position value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The positions are interleaved in the array by the axis index.
***time**	This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.
point	This structure contains the point configuration. Please see MPIMotionPoint data type for more information.

See Also

MPIMotionPoint

MEIMotionTrace

MEIMotionTrace

```
typedef enum {

    MEIMotionTraceSTATUS,
    MEIMotionTracePARAMS,
} MEIMotionTrace;
```

Description

MEIMotionTraceSTATUS	This trace bit enables motion status tracing for mpiMotion calls.
MEIMotionTracePARAMS	This trace bit enables motion parameters tracing for mpiMotion calls.

See Also

MPIMotionTrapezoidal

MPIMotionTrapezoidal

```
typedef struct MPIMotionTrapezoidal {
    MPITrajectory   *trajectory;
    double          *position;
} MPIMotionTrapezoidal;
```

Description

***trajectory**	This structure specifies the parameters for the motion profile.
***position**	This array specifies the target positions for the axes. There is one position for each axis.

See Also

MPIMotionType / MEIMotionType

MPIMotionType

```
typedef enum {
    MPIMotionTypeINVALID,
    MPIMotionTypeJOG,

    MPIMotionTypePT,
    MPIMotionTypePVT,
    MPIMotionTypeSPLINE,
    MPIMotionTypeBESSEL,
    MPIMotionTypeBSPLINE,
    MPIMotionTypeBSPLINE2,

    MPIMotionTypeS_CURVE,
    MPIMotionTypeTRAPEZOIDAL,
    MPIMotionTypeS_CURVE_JERK,

    MPIMotionTypeVELOCITY,
    MPIMotionTypeVELOCITY_JERK,
    MPIMotionTypeMASK,
} MPIMotionType;
```

Description

MPIMotionType specifies the particular motion profile algorithm to be generated with mpiMotionStart(…) and mpiMotionModify(…).

MPIMotionTypePT	This type fits constant velocity profile segments through a list of position and time points. Please see MPIMotionPT for more information.
MPIMotionTypePVT	This type fits jerk profile segments through a list of position, velocity and time points. Please see MPIMotionPVT for more information.
MPIMotionTypeSPLINE	This type fits a Cubic spline through a specified list of position and time points. Please see MPIMotionSPLINE data type for more information.
MPIMotionTypeBESSEL	This type fits a Bessel spline through a specified list of position and time points. Please see MPIMotionBESSEL data type for more information.
MPIMotionTypeBSPLINE	This type fits a 3rd order B spline through a list of position and time points. Please see MPIMotionBSPLINE data type for more information.
MPIMotionTypeBSPLINE2	This type fits a 2nd order B spline through a list of position and time points. Please see MPIMotionBSPLINE data type for more information.
MPIMotionTypeS_CURVE	This type specifies point to point motion using a S-Curve velocity profile. The profile is specified by acceleration, velocity, deceleration, jerkPercent, and final position. Please see MPIMotionSCurve data type for more information.
MPIMotionTypeS_CURVE_JERK	This type specifies point to point motion using a S-Curve velocity profile. The profile is specified by acceleration, velocity, deceleration, accelerationJerk, decelerationJerk, and final position. Please see MPIMotionSCurve data type for more information.
MPIMotionTypeTRAPEZOIDAL	This type specifies simple point to point motion using a trapezoidal velocity profile. The profile trajectory is specified by acceleration, velocity, deceleration and final position. Please see MPIMotionTrapezoidal data type for more information.
MPIMotionTypeVELOCITY	This type specifies S-Curve acceleration to a constant velocity. The profile trajectory is specified by acceleration, velocity, and jerkPercent. Please see MPIMotionVelocity data type for more information.
MPIMotionTypeVELOCITY_JERK	This type specifies S-Curve acceleration to a constant velocity. The profile trajectory is specified by acceleration, velocity, accelerationJerk and decelerationJerk. Please see MPIMotionVelocity data type for more information.

MEIMotionType

```
typedef enum {
    MEIMotionTypeFRAME,
} MEIMotionType;
```

Description

MEIMotionTypeFRAME	This motion type is used to construct motion profiles at the frame level.

See Also

mpiMotionStart | mpiMotionModify | mpiMotionTYPE

MPIMotionVelocity

MPIMotionVelocity

```
typedef struct MPIMotionVelocity {
    MPITrajectory       *trajectory;
} MPIMotionVelocity;
```

Description

***trajectory**	This structure specifies the parameters for the motion profile, except deceleration is ignored. Please see MPITrajectory data type for more information.

See Also

MPITrajectory

mpiMotionATTR

Declaration

```
#define  mpiMotionATTR(type,attr)
((type) |= mpiMotionAttrMaskBIT(attr))
```

Required Header

stdmpi.h

Description

MotionATTR turns on the specified motion attribute mask bits in the motion type.

See Also

MPIMotionAttr | MPIMotionAttrMask

mpiMotionAttrMaskBIT

Declaration `#define mpiMotionAttrMaskBIT(attr) (0x1 << (attr))`

Required Header stdmpi.h

Description **MotionAttrMaskBIT** converts the motion attribute into the motion attribute mask.

See Also MPIMotionAttr | MPIMotionAttrMask

mpiMotionTYPE

Declaration

```
#define  mpiMotionTYPE(type)    ((type) & MPIMotionTypeMASK)
```

Required Header

stdmpi.h

Description

MotionType masks off all other bits in ***type***, leaving the motion type.

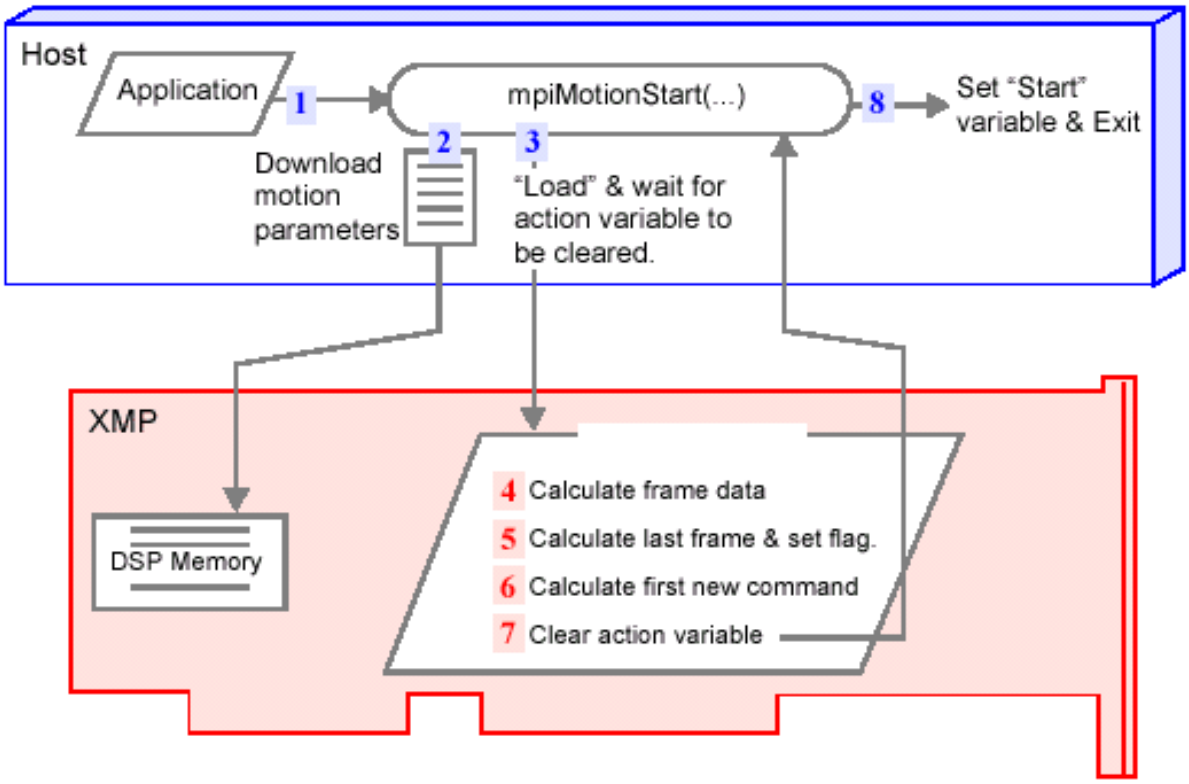
See Also

MPIMotionType

Diagram of mpiMotionStart

The diagram below explains how mpiMotionStart(...) executes.

Host	1	Application calls mpiMotionStart(...)	Timing starts.	
	2	Host transfers motion parameters to DSP memory.	Delays are due to validation, locks, etc., and depend mostly on CPU speed, and debug/release versions.	
	3	Host sets the Load action in XMP Motion Supervisor and waits (sleeps) until the action variable is cleared by the XMP.		
Xmp	4	XMP Motion Supervisor calculates frame data for each axis using the motion parameters.	Calculation occurs once per sample during each background cycle of XMP (1 usec with 12 axes at 2 kHz sample rate), and takes only a few µsec per axis. Number of frames/point vary with type of move (1frame/point for trapezoidal, to 8 frames/point for S-curve).	Background
	5	XMP calculates the last frame for the move and sets a flag to enable the axis to use the new frames.	Occurs during same background cycle as step 4.	
	6	XMP axis routine calculates the first new command using data from the first frame of the new move. It calculates a new PID output and sets the DACs accordingly.	Occurs in foreground (sample interrupt, every 500 µsec at 2 kHz sample rate).	Foreground
	7	XMP clears the Motion Supervisor's action variable.		
Host	8	Host sets the "Start" action in the XMP Motion Supervisor and exits mpiMotionStart(...).	Host doesn't wait for action variable to clear, but will check it before setting the variable in any subsequent calls to mpiMotionStart(...).	



Return to mpiMotionStart

Motion Error Codes and Messages

Code	Message	Tips / Fixes
0x0d01 (3329)	Motion: motion invalid	The Motion parameters are incorrect. Recheck them.
0x0d02 (3330)	Motion: axis not found	The Axis number is invalid.
0x0d03 (3331)	Motion: axis count invalid	You may be commanding moves on Axes that don't exist, or on zero Axes.
0x0d04 (3332)	Motion: type invalid	You may have called a motion type that doesn't exist.
0x0d05 (3333)	Motion: attribute invalid	The value for a motion attribute is wrong.
0x0d06 (3334)	Motion: not ready	The motion controller is still executing the power-up sequence.
0x0d07 (3335)	Motion: MPIStateIDLE	No Motion was in progress when mpiMotionModify(...) was called. In order for mpiMotionModify(...) to work, there must be a motion in progress.
0x0d08 (3336)	Motion: MPIStateMOVING	You may have called mpiMotionStart(...) when a Motion was in progress.
0x0d09 (3337)	Motion: MPIStateSTOPPING	Indicates that a Stop event is in progress.
0x0d0a (3338)	Motion: MPIStateSTOPPING_ERROR	Indicates that an E-Stop event is in progress.
0x0d0b (3339)	Motion: MPIStateERROR	Indicates that an E-Stop or Abort event has occurred. Use mpiMotionAction(MPIActionRESET) to recover.
0x0d0c (3340)	Motion: auto-start	mpiMotionModify(...) was called with the MPIMotionAttrMaskAUTO_START attribute when the motion state was MPIStateIDLE. In this case, the motion profile is automatically started.
0x0d0d (3341)	Motion: profile error	The XMP couldn't generate the next set of points. Check the motion parameters and attributes.

Return to Motion Objects page

Motion Attributes

Introduction | MPI Motion Attributes | MEI Motion Attributes

Introduction

This section details the use of various Motion Attributes, which are listed individually below. To use Motion Attributes, OR the attribute mask with the MPIMotionType value.

TOP

MPI Motion Attributes

MPIMotionAttrAPPEND

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskAPPEND attribute. This makes it possible to buffer several point-to-point motion profiles in the controller. Each successful call to mpiMotionStart(...) with the APPEND attribute will generate an MPIEventTypeMOTION_DONE from the controller when the motion is complete.

The MPIMotionAttrMaskID attribute is supported with MPIMotionAttrMaskAPPEND when calling mpiMotionStart(...). The XMP-Series controller firmware has been modified to buffer the ID values inside the axis' frames. Therefore, applications using the motion and axis event user data must be changed. Since the ID is stored in the controller's axis frames, the mpiMotionEventNotifySet(...) and mpiAxisEventNotifySet(...) must explicitly configure an appropriate axis memory location for the firmware to retrieve the ID. The sample programs motionID1.c and motionID2.c demonstrate this feature.

The MEIMotionAttrHOLD attribute is supported with MPIMotionAttrMaskAPPEND when calling mpiMotionStart(...).

Move Types: **PT, PVT, Bessel, Bspline, Bspline2, S-Curve, Trapezoidal, Velocity, Frame**

MPIMotionAttrAUTO_START

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskAUTO_START attribute. When AUTO_START is enabled, calls to mpiMotionModify(…) will automatically start a new motion if the previous motion is done. If AUTO_START is not enabled, and mpiMotionModify(...) is commanded after the initial motion has completed, the method will return an MPIMotionStateIDLE error.

Move Types: **S-Curve, Trapezoidal, Velocity**

MPIMotionAttrDELAY

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskDELAY attribute. This motion attribute will delay the move for a given number of seconds. MPIMotionParams.attributes.delay is a pointer to an array of doubles that assign delay times for each Axis.

Move Types: **S-Curve, Trapezoidal, Velocity**

MPIMotionAttrELEMENT_ID

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskELEMENT_ID attribute. Similar to the MPIMotionAttrMaskID, the ELEMENT_ID allows the application to set an identification value for each element of a path motion. The ID values are long values configured in the MPIMotionParams.attributes.elementId array. Each element in the array will be the ID value for that sequential portion of the motion.

In order to retrieve the ElementID, a pointer to the element ID is placed into the MEIEventNotifyData structure. This will cause the XMP to send the ElementID up to the MEIEventStatusInfo structure when an event occurs that causes an interrupt.

Move Types: PT, PVT, Bessel, Bspline, Bspline2, S-Curve, Trapezoidal, Velocity

MPIMotionAttrMaskID

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskID attribute. The ID attributes allows the application to assign an identification number to each motion. This number can be returned in the MPIEventStatus structure so the application will know which move has ended. This is particularly useful when multiple moves are buffered and the application needs to know which move is executing or has returned an event.

The MPIMotionParams.attributes.id value is a long that identifies the Motion. This ID value is passed to each Axis associated with the MS. In order to retrieve the MoveID, a pointer to the move ID is placed in the MEIEventNotifyData structure. This will cause the XMP to send the MoveID up to the MEIEventStatusInfo structure when an event occurs that causes an interrupt.

Move Types: **PT, PVT, Spline, Bessel, Bspline, Bspline2, S-Curve, Trapezoidal, Velocity, Frame**

MPIMotionAttrRELATIVE

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskRELATIVE attribute. When this mask is ANDed into the attribute mask, all position values will be used as relative motion distances instead of final absolute positions. For example, without the RELATIVE motion attribute, a move beginning at position 1000 with a position parameter value of 2000 will move to position 2000. If the RELATIVE attribute is turned on, the final move position will be 3000, a relative distance of 2000 counts from the starting value of 1000.

Move Types: **PT, PVT, Spline, Bessel, Bspline, Bspline2, S-Curve, Trapezoidal, Velocity**

MPIMotionAttrSYNC_END

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskSYNC_END attribute. SYNC_END is used for motions that include more than one axis. The SYNC_END attribute will generate trajectories for all axes appended to an MS that will end simultaneously. For all but the longest motion profile, wait frames will be added to the beginning of the moves. This will ensure that all axes end simultaneously.

Move Types: **S-Curve, Trapezoidal**

MPIMotionAttrSYNC_START

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskSYNC_START attribute. SYNC_START is used for Motions that include more than one Axis. All Axes will begin simultaneously. For those axes that finish first, a delay frame will be added to the end of the move.

Move Types: **S-Curve, Trapezoidal**

TOP

MEI Motion Attributes

MEIMotionAttrEVENT

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskEVENT attribute.

MEIMotionAttrFINAL_VEL

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskFINAL_VEL attribute.

MEIMotionAttrNO_REVERSAL

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskNO_REVERSAL attribute.

MEIMotionAttrHOLD

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskHOLD attribute. The HOLD attribute prevents execution of a Motion. The HOLD attribute is applied at the beginning of the motion (one HOLD frame) before the execution of the point list. This prevents execution of the Motion until the HOLD frame is disabled.

The HOLD attribute is used to synchronize the start of motion with a host function call, XMP internal variable, or Motor Input state change. More than one Motion Supervisor may be synchronized. The type field of the MEIMotionAttrHold{} structure determines whether the synchronization comes from a host call (meiControlGateSet(), see below), internal variable (Axis Status, Position, etc.) or a Motor Input signal (transceiver, home input, user input, etc.).

Gated Moves: If the value of type is MEIMotionAttrHoldTypeGATE, the motion will be held until a call to mpiControlGateSet() is made with the closed parameter set to FALSE. When a MEIMotionAttrHoldTypeGATE is used, the source.gate field of MEIMotionAttrHold{} must be set to the gate number (0-31). The same gate number must be used for the gate parameter of meiControlGateSet().

Input Hold Moves: If the value of type is MEIMotionAttrHoldTypeINPUT, the motion will be held until then value of the internal Xmp variable specified (pointed to) by source.input bitwise anded with source.mask matches source.pattern.

Motor Input Hold Moves: If the value of type is MEIMotionAttrHoldTypeMOTOR, the motion will be held until the value of the internal dedicated input word (Motor[n].IO. DedicatedIN.IO) for the motor specified by source.motorNumber bitwise anded with source.mask matches source.pattern.

The timeout field of MEIMotionAttrHold{} will cause the motion to start after the specified timeout period (in seconds), even if the other hold criteria have not been satisfied. A value of zero for timeout causes the timeout feature to be disabled and forces the motion to wait for the hold criteria (gate open, or pattern match).

The MPI expects an array of hold attributes specifying separate attributes form each axis of a motion supervisor. All axes holding with the same hold attributes (same gate, same input, mask, and pattern) will start motion in the same sample even if the moves are specified using different motion supervisors.

MEIMotionAttrOUTPUT

The MPI has been extended to support mpiMotionStart(...) with the MPIMotionAttrMaskAPPEND attribute. This motion attribute allows a Motion to change the state of a register in the controller memory. This configures a frame to toggle an output bit as a move begins.

TOP

Return to Motion Objects page

Returns for mpiMotionStart and mpiMotionModify

Both mpiMotionStart and mpiMotionModify use motion attributes, which are summarized in the table below.

Return Values - ***for both mpiMotionStart and MotionModify***	
MPIMotionMessageERROR	if *MotionStart* is called when the Motion Supervisor is in the error state.
MPIMotionMessageAXIS_COUNT	if the motion object has no axes.
MPIMotionMessagePROFILE_ERROR	if the Motion profile specified by the parameters cannot be calculated correctly.
MPIMotionMessageATTRIBUTE_INVALID	if the specified motion attribute mask is not valid with the motion type.
MPIMessagePARAM_INVALID	if the specified motion parameters are not valid with the Motion type.
MPIMessageARG_INVALID	if any of the *MotionStart* arguments are not valid.
MPIMessageUNSUPPORTED	if the motion type is not supported by the controller.

Return to mpiMotionStart or mpiMotionModify