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 using 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
meiMotion**FrameBufferLoad**	
meiMotion**LogClose**	Frees memory buffer for motion logging
meiMotion**LogOpen**	Allocates a memory buffer for motion logging.
meiMotion**LogPrint**	Writes motion log buffer data
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
meiMotion**ActionFunction**	
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

*MEIMotion**ActionFunction**
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**Cam**
MPIMotion**Config** / MEIMotion**Config**
MPIMotion**DecelTime**
MEIMotion**Frame**
MEIMotion**FrameBufferStatus**
MPIMotion**Message** / MEIMotion**Message**
MPIMotion**Params** / MEIMotion**Params**
MPIMotion**Point**
MPIMotion**PT**
MPIMotion**PTF**
MPIMotion**PVT**
MPIMotion**PVTF**
MPIMotion**SCurve**
MPIMotion**SCurveJerk**
MPIMotion**SPLINE**
MEIMotion**Trace**
MPIMotion**Trapezoidal**
MPIMotion**Type** / MEIMotion**Type**
MPIMotion**Velocity**

Macros

mpiMotion**ATTR**

mpiMotion**AttrMaskBIT**

mpiMotion**TYPE**

mpiMotionCreate

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionCreate 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

mpiMotionDelete 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	

See Also

mpiMotionCreate | mpiMotionValidate

mpiMotionValidate

Declaration

```
long mpiMotionValidate(MPIMotion  motion)
```

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiMotionCreate | mpiMotionDelete

mpiMotionConfigGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionConfigGet 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).

Remarks

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

Return Values	
MPIMessageOK	

Sample Code

```
void modifyFeedrate(MPIMotion motion,
float normalFeedrate,
float pauseFeedrate)
{
MPIMotionConfig motionConfig;
long returnValue;

returnValue = mpiMotionConfigGet(motion,
&motionConfig,
NULL);
msgCHECK(returnValue);

printf("Before: normalFeedrate %lf, pauseFeedrate %lf\n",motionConfig.
normalFeedrate,motionConfig.pauseFeedrate);
motionConfig.normalFeedrate = normalFeedrate;
motionConfig.pauseFeedrate = pauseFeedrate;

returnValue = mpiMotionConfigSet(motion,
&motionConfig,
NULL);
msgCHECK(returnValue);

printf("After: normalFeedrate %lf, pauseFeedrate %lf\n",motionConfig.
normalFeedrate,motionConfig.pauseFeedrate);
}
```

See Also

mpiMotionConfigSet | MEIMotionConfig

mpiMotionConfigSet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionConfigSet 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).

Remarks

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

Return Values	
MPIMessageOK	

Sample Code

```
void modifyFeedrate(MPIMotion motion,
float normalFeedrate,
float pauseFeedrate)
{
MPIMotionConfig motionConfig;
long returnValue;

returnValue = mpiMotionConfigGet(motion,
&motionConfig,
NULL);
msgCHECK(returnValue);

printf("Before: normalFeedrate %lf, pauseFeedrate %lf\n",motionConfig.
normalFeedrate,motionConfig.pauseFeedrate);
motionConfig.normalFeedrate = normalFeedrate;
motionConfig.pauseFeedrate = pauseFeedrate;

returnValue = mpiMotionConfigSet(motion,
&motionConfig,
NULL);
msgCHECK(returnValue);

printf("After: normalFeedrate %lf, pauseFeedrate %lf\n",motionConfig.
normalFeedrate,motionConfig.pauseFeedrate);
}
```

See Also

mpiMotionConfigGet | MEIMotionConfig

mpiMotionFlashConfigGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionFlashConfigGet 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.

Remarks

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.

motion	a handle to a Motion 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.
***config**	a pointer to a configuration structure for the motion object of type MPIMotionConfig.
***external**	a pointer to a configuration structure for the motion object of type MEIMotionConfig.

Return Values	
MPIMessageOK	

See Also

mpiMotionFlashConfigSet

mpiMotionFlashConfigSet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionFlashConfigSet 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.

Remarks

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	

See Also

MEIMotionConfig | MEIFlash | mpiMotionFlashConfigGet

meiMotionFrameBufferLoad

Declaration

```
long meiMotionFrameBufferLoad(MPIMotion    motion,         /* TRUE/FALSE */
                              MPI_BOOL     initial,        /* TRUE/FALSE */
                              MPI_BOOL     lock,           /* TRUE/FALSE */
                              MPI_BOOL     frameLowEvent); /* TRUE/FALSE */
```

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

Description

meiMotionFrameBufferLoad

motion	a handle to the Motion object
initial	
lock	
frameLowEvent	

Return Values	
MPIMessageOK	

See Also

MEIMotionFrameBufferStatus

meiMotionLogClose

Declaration

```
long meiMotionLogClose(MPIMotion    motion);
```

Required Header: stdmei.h

Description

meiMotionLogClose frees the memory buffer for motion logging.

motion	a handle to the Motion object.

Return Values	
MPIMessageOK	

See Also

meiMotionLogOpen | meiMotionLogPrint

meiMotionLogOpen

Declaration

```
long meiMotionLogOpen(MPIMotion    motion,
                      long         maxEntries);
```

Required Header: stdmei.h

Description

meiMotionLogOpen allocates a memory buffer for motion logging.

motion	a handle to the Motion object.
maxEntries	The maximum number of log entries. This value determines the log buffer memory size.

Return Values	
MPIMessageOK	

See Also

meiMotionLogClose | meiMotionLogPrint

meiMotionLogPrint

Declaration

```
long meiMotionLogPrint(MPIMotion    motion,
                       const  char  *filename);
```

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

Description

meiMotionLogPrint reads the motion log buffer entries and writes the data into a file specified by the ***fileName***.

motion	a handle to the Motion object.
***filename**	a pointer to a file name string.

Return Values	
MPIMessageOK	

See Also

meiMotionLogOpen | meiMotionLogClose

mpiMotionParamsGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionParamsGet 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	
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

mpiMotionParamsSet 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	

See Also

mpiMotionStart | mpiMotionParamsGet

meiMotionParamsValidate

Declaration

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

Required Header: stdmei.h

Description

meiMotionParamsValidate 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	

See Also

mpiMotionPositionGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionPositionGet 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	

See Also

mpiMotionPositionSet | Controller Positions

mpiMotionPositionSet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionPositionSet 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	

See Also

mpiMotionPositionGet | Controller Positions

mpiMotionStatus

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionStatus 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).

Remarks

external should always be set to NULL.

motion	a handle to a Motion object
***status**	a pointer to MPIStatus structure
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiMotionTrajectory

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionTrajectory reads the current velocity and acceleration 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***).

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

Remarks

external should always be set to NULL.

Return Values	
MPIMessageOK	

Sample Code

```
MPITrajectory trajectory;

    mpiAxisTrajectory(axis, &trajectory);

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

See Also

MPITrajectory

mpiMotionEventNotifyGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionEventNotifyGet 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***.

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

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo | mpiMotionEventNotifySet

mpiMotionEventNotifySet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionEventNotifySet 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.

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	set eventmask to MPIEventMaskALL
disable host notification of all events	set eventmask to MPIEventTypeNONE

Return Values	
MPIMessageOK	

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo | mpiEventMaskMOTION | mpiEventMaskAXIS | mpiMotionEventNotifyGet

mpiMotionEventReset

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionEventReset 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	

See Also

mpiControlEventReset | mpiMotorEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | meiSqNodeEventReset | mpiAxisEventReset

Event Notification Methods

mpiMotionAction

Declaration

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

Required Header: stdmpi.h

Description

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

motion	a handle to the Motion object.
action	an enumerated value representing the action to be performed. Both the MPIAction and MEIAction enumerations are valid for the action argument.

The Stop, E-Stop, and E-Stop Modify actions have configurable parameters for the deceleration profiles. See MPIMotionConfig and MPIAxisEstopModify for details.

The mpiMotionAction(…) requires axes mapped to the Motion Supervisor, except for the actions MPIActionRESET and MEIActionMAP. If there are no axes mapped to the Motion Supervisor, mpiMotionAction(…) will return MEIMotionMessageNO_AXES_MAPPED. The MPIActionRESET and MEIActionMAP actions will automatically map the axes associated with the motion object.

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 STOPPED 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 supervisor (motion) is in the error state (MPIStateERROR), the motion supervisor will attempt to clear the error and change to the idle state (MPIStateIDLE), so that motion supervisor can be restarted. If the error state was caused by an Abort action, then the command position of associated axes 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), a STOP action will decelerate the axis (or axes) to a stop in the time specified by the "stop" time configuration. The motion state will transition to MPIStateSTOPPING during the deceleration and then to MPIStateSTOPPED after the motion completes.
MEIActionMAP	MotionAction will write the axis mapping relationship of the motion supervisor to the controller. This mapping is written automatically when mpiMotionStart() is called.

Return Values	
MPIMessageOK	
MEIMotionMessageNO_AXES_MAPPED	
MPIMotionMessageIDLE	
MPIMotionMessageMOVING	

See Also

MPIAction | MEIAction | mpiMotionConfigSet | mpiMotionConfigGet

How STOP Events Work

meiMotionActionFunction

Declaration

```
MEIMotionActionFunction  meiMotionActionFunction(MPIMotion    motion,
                                  MEIMotionActionFunction
function);
```

Required Header: stdmei.h

Description

meiMotionActionFunction allows the user to specify a callback function for the motion object. In order to start an action (motion start, motion modify, stop, abort, reset, etc…), the callback function is called whenever the MPI writes to the action variable in the firmware's motion supervisor object.

motion	a handle to the Motion object
function	pointer to a callback function.

Return Values	
MPIMessageOK	

See Also

MEIFlashFiles | mpiControlReset

mpiMotionModify

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionModify 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). It is impossible to use the HOLD attributes with mpiMotionModify, even when using the AUTO_START attribute.

Each axis has a 128 frame buffer (FIFO). mpiMotionStart and mpiMotionModify calls will load up to 10 frames. No provision has been made to check if the new frames will overwrite the currently executing frames. This could happen after about 12 Start/Modify calls are made with the APPEND attribute.

The XMP firmware velocity frame execution time for point-to-point moves cannot exceed 16,384,000 samples. With the sample rate configured for 2000 (default), the maximum velocity time is 2.27 hours. If the commanded motion exceeds the maximum frame time, the axes will stop abruptly after 16,384,000 samples. The motors will still maintain servo control and no errors are reported.

Return Values	
MPIMessageOK	
MPIMotionMessageIDLE	
MPIMotionMessageAUTO_START	
MPIMotionMessagePROFILE_ERROR	
For more Returns, click here	

See Also

mpiMotionStart

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionStart 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.

Each axis has a 128 frame buffer (FIFO). mpiMotionStart and mpiMotionModify calls will load up to 10 frames. No provision has been made to check if the new frames will overwrite the currently executing frames. This could happen after about 12 Start/Modify calls are made with the APPEND attribute.

If E-stop deceleration rates are not set high enough to stop within the number of frames specified by the empty frame limit, the axis jumps on a frame underflow. The axis will E-stop along the path of the last frames in the buffer, then continue onto the next frames (which are the frames from 128 frames ago). This can potentially cause a dangerous condition.

When successive non-integer length relative motions are commanded, the fractional portion is truncated and discarded. This may cause problems if the fractional value needs to be summed over multiple moves.

The XMP firmware velocity frame execution time for point-to-point moves cannot exceed 16,384,000 samples. With the sample rate configured for 2000 (default), the maximum velocity time is 2.27 hours. If the commanded motion exceeds the maximum frame time, the axes will stop abruptly after 16,384,000 samples. The motors will still maintain servo control and no errors are reported.

Return Values	
MPIMessageOK	
MPIMotionMessageMOVING	
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

mpiMotionMemory 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	

See Also

mpiMotionMemoryGet | mpiMotionMemorySet

mpiMotionMemoryGet

Declaration

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

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

Description

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

Return Values	
MPIMessageOK	

See Also

mpiMotionMemorySet | mpiMotionMemory

mpiMotionMemorySet

Declaration

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

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

Description

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

Return Values	
MPIMessageOK	

See Also

mpiMotionMemoryGet | mpiMotionMemory

mpiMotionControl

Declaration

```
MPIControl mpiMotionControl(MPIMotion  motion)
```

Required Header: stdmpi.h

Description

mpiMotionControl 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

mpiMotionNumber 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	

See Also

mpiMotionAxis

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxis 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	
MEIListMessageELEMENT_NOT_FOUND	
MPIMessageHANDLE_INVALID	

See Also

mpiMotionAxisAppend

Declaration

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

Required Header: stdmpi.h

Description

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

When using multiple Motion Supervisors that share axes, Motion events (Done, AtVelocity) are sent to both Motion objects, no matter which Motion Supervisor commanded the motion. This occurs, because the Motion events are derived from the Motion Supervisor status, which is derived from each axis' status.

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

Return Values	
MPIMessageOK	
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

mpiMotionAxisCount 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

```
MPIAxis mpiMotionAxisFirst(MPIMotion  motion)
```

Required Header: stdmpi.h

Description

mpiMotionAxisFirst 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	

See Also

mpiMotionAxisNext | mpiMotionAxisLast

mpiMotionAxisIndex

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisIndex 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

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

motion	a handle to the Motion object.

Return Values	
MPIMessageOK	

See Also

mpiMotionAxis

mpiMotionAxisLast

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisLast 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 first Axis of a Motion (***motion***)

MPIHandleVOID	if ***motion*** is invalid
	if ***motion*** is empty

MPIMessageHANDLE_INVALID	

See Also

mpiMotionAxisPrevious

mpiMotionAxisListGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisListGet 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***).

motion	a handle to the Motion object.

Return Values	
MPIMessageOK	

See Also

mpiMotionAxisListSet | mpiMotionAxis

mpiMotionAxisListSet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisListSet 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	

Sample Code

```
    /* Creates a 1:1 mapping between motion supervisor and axis */
    MPIMotion motion;
    MPIAxis axis;
    long returnValue;

    axis = mpiAxisCreate(control, axisNumber);
    returnValue = mpiAxisValidate(axis);
    msgCHECK(returnValue);

    motion = mpiMotionCreate(control,
                             axisNumber,
                             NULL);
    returnValue = mpiMotionValidate(motion);
    msgCHECK(returnValue);

    returnValue = mpiMotionAxisListSet(motion,
                                       1,
                                                     &axis);
    msgCHECK(returnValue);

    returnValue = mpiMotionAction(motion, MEIActionMAP);
    msgCHECK(returnValue);

    /* Don't forget to delete your motion supervisor and axis objects */
```

See Also

mpiMotionCreate | mpiMotionAxisListGet | mpiMotionAxis

mpiMotionAxisNext

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisNext 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 ***index***th Axis of a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if the specified Axis (***axis***) is the last axis in a Motion (***motion***)
MPIMessageHANDLE_INVALID	

See Also

mpiMotionAxisFirst | mpiMotionAxisPrevious

mpiMotionAxisPrevious

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisPrevious 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 ***index***th Axis of a Motion (***motion***)
MPIHandleVOID	if ***motion*** is invalid if the specified Axis (***axis***) is the last axis in a Motion (***motion***)
MPIMessageHANDLE_INVALID	

See Also

mpiMotionAxisLast | mpiMotionAxisNext

mpiMotionAxisRemove

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionAxisRemove removes an Axis (***axis***) from a Motion (***motion***). Since the Motion Supervisor is a host-based object, a Motion Supervisor's axes mapping will only be loaded onto the controller when the application runs and an action is performed. View the Motion Supervisor and Axis Mapping: MS Host-based Object video tutorial for more information.

Return Values	
MPIMessageOK	

Sample Code

Call **mpiMotionAction(..., MEIActionMAP)** after all calls to mpiMotionRemove(...) and before mpiMotionDelete(...). See the sample code below.

```
mpiMotionAxisRemove

mpiMotionAction(motion, MEIActionMAP)

mpiMotionDelete
```

See Also

mpiMotionAction | mpiMotionDelete | MEIAction

MEIMotionActionFunction

Definition

```
typedef long   /* Callback prototype */
    (*MEIMotionActionFunction)(MPIMotion        motion,
                               MPIMotionType    type,
                               MEIXmpAction     xmpAction);
```

Description

MEIMotionActionFunction is a function prototype definition. This is the prototype for a user defined callback function in the mpiMotionAction(...) method. If a callback function is set using meiMotionActionFunction(...), then every time the application calls mpiMotionAction(...), the callback function will be executed. Your motion action callback function must implement the exact interface of the above prototype.

Using a callback function can be useful to log details of mpiMotionAction(...) calls.

WARNING:
Defining a callback function will affect the execution time of the mpiMotionAction(...) method call. In order to preserve system performance, the callback function should be written to execute as fast as possible.

motion	The motion object handle used to call mpiMotionAction() will be passed to the callback function.
type	The type of motion that the action was called to act upon will be passed to the callback function.
xmpAction	The type of action that was requested will be passed to the callback function.

See Also

mpiMotionAction | meiMotionActionFunction

MPIMotionAttr / MEIMotionAttr

Definition: MPIMotionAttr

```
typedef enum {

    MPIMotionAttrAPPEND,
    MPIMotionAttrAUTO_START,
    MPIMotionAttrDELAY,
    MPIMotionAttrID,
    MPIMotionAttrELEMENT_ID,
    MPIMotionAttrRELATIVE,
    MPIMotionAttrSYNC_END,
    MPIMotionAttrSYNC_START,
    MPIMotionAttrREPEAT,
    MPIMotionAttrMASTER_START,
    MPIMotionAttrNO_BACKTRACK,
    MPIMotionAttrNO_BACKTRACK_HOLD,

    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.

MPIMotionAttrAPPEND	This bit 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 bit can be used with mpiMotionStart(...) or mpiMotionModify(...).
MPIMotionAttrAUTO_START	This bit 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.

MPIMotionAttrDELAY	This bit enables a time delay (seconds) before the motion profile begins. This mask can be used with mpiMotionStart(...). Motion with Modify is not supported with the DELAY attribute. Please see MPIMotionAttributes for more information.
MPIMotionAttrID	This bit enables an identification tag to be stored in the motion profile. Please see MPIMotionAttributes for more information. This bit can be used with mpiMotionStart(...) and mpiMotionModify(...).
MPIMotionAttrELEMENT_ID	This bit enables an identification tag to be stored in the path motion profiles. Please see MPIMotionAttributes for more information.
MPIMotionAttrRELATIVE	This bit changes the profile target position from absolute to relative coordinates. Currently only supports APPEND and ID attributes. Support for PT, PVT, SPLINE, BESSEL, or BSPLINE motion types will be added in the future.
MPIMotionAttrSYNC_END	This bit 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.
MPIMotionAttrSYNC_START	This bit 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.
MPIMotionAttrREPEAT	This attribute generates a repeating cam motion. If you use this attribute you need to fill in the repeatFrom field in the MPIMotionAttributes structure. See Repeating Cams.
MPIMotionAttrMASTER_START	This attribute specifics the position of the master that a cam will start.If you use this attribute you need to fill in the masterStart field in the MPIMotionAttributes structure. See Starting at Specific Master Positions.

MPIMotionAttrREPEAT_NO_BACKTRACK	This attribute modifies a cam motion so that when the slave axes will only progress if the master is moving in a positive direction, if the master changes direction and moves backwards the slave axes will hold that position until the master velocity returns to the original direction.This attribute cannot be specified with the MPIMotionAttrNO_BACKTRACK attribute. See Reversal of the Master - Backtracking.
MPIMotionAttrREPEAT_NO_BACKTRACK_HOLD	This attribute modifies a cam motion so that when the master changes direction the slave axes will hold that position until the master returns to the point where the direction changed.This attribute cannot be specified with the MPIMotionAttrFORWARD_ONLY attribute. See Reversal of the Master with NO_BACKTRACK_HOLD.

Definition: MEIMotionAttr

```
typedef enum {
    MEIMotionAttrEVENT,
    MEIMotionAttrFINAL_VEL,
    MEIMotionAttrNO_REVERSAL,
    MEIMotionAttrHOLD,
    MEIMotionAttrHOLD_LESS,
    MEIMotionAttrHOLD_GREATER,
    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.

MEIMotionAttrEVENT	This bit allows the user to specify an MPIEventMask during a motion.
MEIMotionAttrFINAL_VEL	This bit allows the user to specify a non-zero target velocity for point to point motion types.
MEIMotionAttrNO_REVERSAL	This bit prevents a motion profile from changing direction.
MEIMotionAttrHOLD	This bit prevents a motion profile from executing until the specified trigger conditions are met. MPIMotionMessageATTRIBUTE_INVALID will be returned if MEIMotionAttrHOLD is used with a mpiMotionModify(...) method.
MEIMotionAttrHOLD_LESS	Motion attribute bit for less than or equal hold logic. MPIMotionMessageATTRIBUTE_INVALID will be returned if MEIMotionAttrHOLD_LESS is used with a mpiMotionModify (...) method.
MEIMotionAttrHOLD_GREATER	Motion attribute bit for greater than or equal hold logic. MPIMotionMessageATTRIBUTE_INVALID will be returned if MEIMotionAttrHOLD_GREATER is used with a mpiMotionModify(...) method.
MEIMotionAttrOUTPUT	This bit allows the user to set or clear bits during a motion.

See Also

MPIMotionAttrMask | mpiMotionStart | mpiMotionModify

MEIMotionAttrHold

Definition

```
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 (samples). When the time exceeds the timeout value or the hold conditions are met, the motion profile will execute.

See Also

MEIMotionAttrHoldType | MEIMotionAttrHoldSource | MEIMotionAttrMask

MEIMotionAttrHoldSource

Definition

```
typedef union {
    long        gate;
    struct {
        long    *input;
        long    mask;
        long    pattern;
    } input;
    struct {
        long    number;
        long    mask;
        long    pattern;
    } motor;
    struct {
        long    number;
        long    position;
    } axis;
    struct {
        long    *address;
        long    mask;
        long    pattern;
    } user;
} 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.
input.mask	This value specifies the AND mask when MEIMotionAttrHoldTypeINPUT is used.
input.pattern	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.
motor.number	This value specifies the motor number when MEIMotionAttrHoldTypeMOTOR is used.

motor.mask	This value specifies the AND mask when MEIMotionAttrHoldTypeMOTOR is used.
motor.pattern	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.
axis.number	An axis's number (0, 1, 2, etc.). Must be specified when the AXIS_POSITION_COMMAND or AXIS_POSITION_ACTUAL motion hold types are used.
axis.position	A position comparison value. Must be specified when the AXIS_POSITION_COMMAND or AXIS_POSITION_ACTUAL motion hold types are used.
user.address	A controller memory address. Must be specified when the USER_ADDRESS motion hold type is used.
user.mask	A bitwise AND mask for the user specified controller memory address. Must be specified when the USER_ADDRESS motion hold type is used.
user.pattern	A comparison value for the masked user.address. When the pattern matches the masked value, the motion will be triggered. Must be specified when the USER_ADDRESS motion hold type is used.

See Also

MEIMotionAttrMask | MEIMotionAttrHoldType | meiControlGateGet | meiControlGateSet |

MEIMotionAttrHoldType

Definition

```
typedef enum {
   MEIMotionAttrHoldTypeINVALID,
   MEIMotionAttrHoldTypeNONE,
   MEIMotionAttrHoldTypeGATE,
   MEIMotionAttrHoldTypeINPUT,
   MEIMotionAttrHoldTypeMOTOR_GENERAL_IO,
   MEIMotionAttrHoldTypeMOTOR_DEDICATED_IO,
   MEIMotionAttrHoldTypeAXIS_POSITION_ACTUAL,
   MEIMotionAttrHoldTypeAXIS_POSITION_COMMAND,
   MEIMotionAttrHoldTypeUSER_ADDRESS,
} MEIMotionAttrHoldType;
```

Change History: Modified in the 03.03.00

Description

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

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_GENERAL_IO	Motion hold input triggers from a General IO bit.
MEIMotionAttrHoldTypeMOTOR_DEDICATED_IO	Motion hold input triggers from a Dedidcated I/O bit.
MEIMotionAttrHoldTypeAXIS_POSITION_ACTUAL	Motion hold input trigger from an axis's actual position.
MEIMotionAttrHoldTypeAXIS_POSITION_COMMAND	Motion hold input trigger from an axis's command position.
MEIMotionAttrHoldTypeUSER_ADDRESS	Motion hold input trigger from a user specifiable controller memory address.

See Also

MEIMotionAttrHoldSource | MEIMotionAttrHold | MEIMotionAttrMask

Motion Attributes

MPIMotionAttrMask / MEIMotionAttrMask

Definition: MPIMotionAttrMask

```
typedef enum {
    MPIMotionAttrMaskAPPEND,
    MPIMotionAttrMaskAUTO_START,
    MPIMotionAttrMaskDELAY,
    MPIMotionAttrMaskID,
    MPIMotionAttrMaskELEMENT_ID,
    MPIMotionAttrMaskRELATIVE,
    MPIMotionAttrMaskSYNC_END,
    MPIMotionAttrMaskSYNC_START,
    MPIMotionAttrMaskREPEAT,
    MPIMotionAttrMaskMASTER_START,
    MPIMotionAttrMaskNO_BACKTRACK,
    MPIMotionAttrMaskNO_BACKTRACK_HOLD,

    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. This mask can be used with mpiMotionStart(...). Motion with Modify is not supported with the DELAY attribute. Please see MPIMotionAttributes for more information.

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. Currently only supports APPEND and ID attributes. Support for PT, PVT, SPLINE, BESSEL, or BSPLINE motion types will be added in the future.
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.
MPIMotionAttrMaskREPEAT	This attribute generates a repeating cam motion. If you use this attribute you need to fill in the repeatFrom field in the MPIMotionAttributes structure. See Repeating Cams.
MPIMotionAttrMaskMASTER_START	This attribute specifics the position of the master that a cam will start.If you use this attribute you need to fill in the masterStart field in the MPIMotionAttributes structure. See Starting at Specific Master Positions.
MPIMotionAttrMaskNO_BACKTRACK	This attribute modifies a cam motion so that when the slave axes will only progress if the master is moving in a positive direction, if the master changes direction and moves backwards the slave axes will hold that position until the master velocity returns to the original direction.This attribute cannot be specified with the MPIMotionAttrNO_BACKTRACK attribute. See Reversal of the Master - Backtracking.

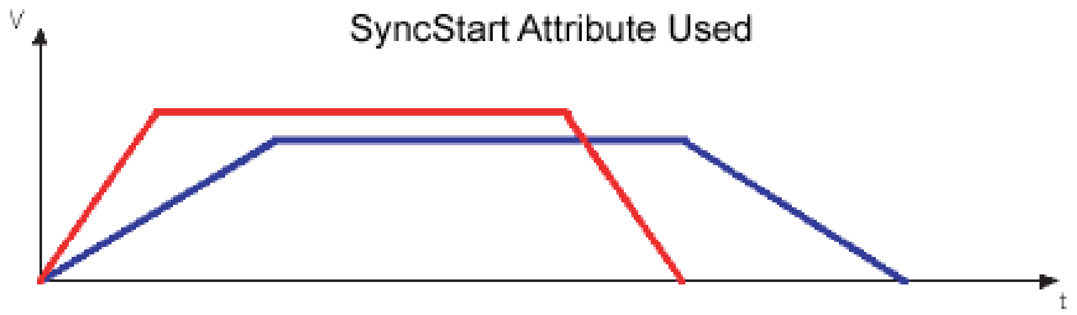
MPIMotionAttrMaskNO_BACKTRACK_HOLD	This attribute modifies a cam motion so that when the master changes direction the slave axes will hold that position until the master returns to the point where the direction changed.This attribute cannot be specified with the MPIMotionAttrFORWARD_ONLY attribute. See Reversal of the Master with NO_BACKTRACK_HOLD.

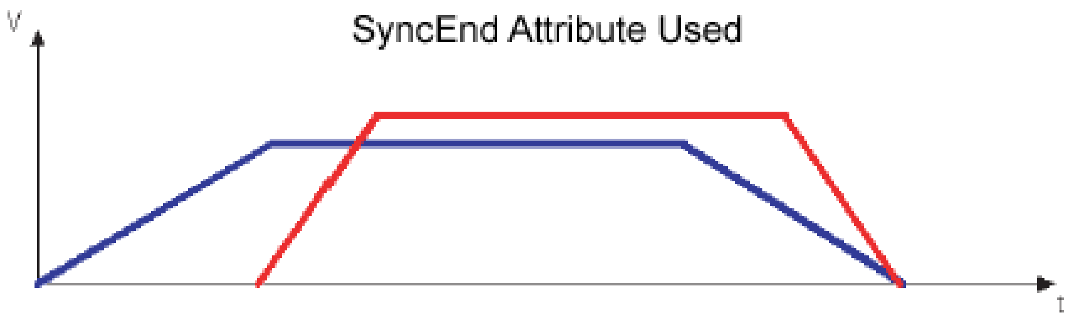
Remarks

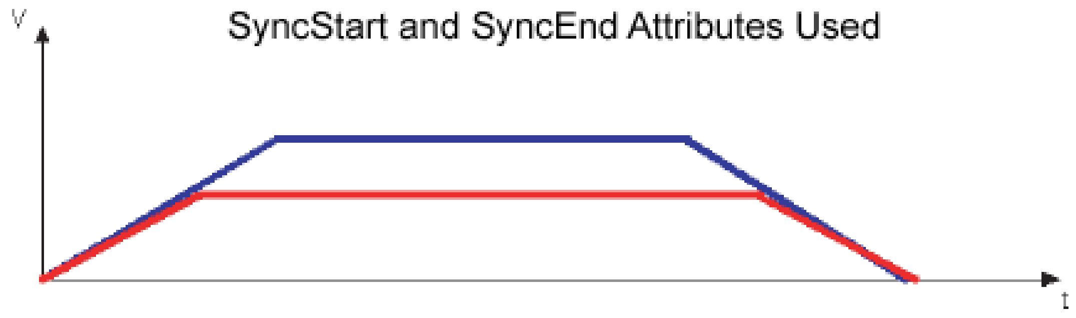
The motion attribute masks are used to enable features with mpiMotionStart(...) and mpiMotionModify (...). The masks are **OR**ed with the MPIMotionType to enable each feature.

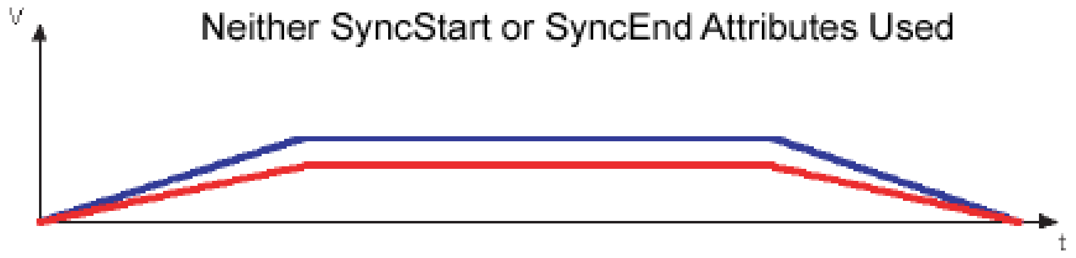
For the motion types MPIMotionTypeS_CURVE_JERK, MPIMotionTypeS_CURVE, MPIMotionTypeTRAPEZOIDAL, if neither MPIMotionAttrMaskSYNC_START nor MPIMotionAttrMaskSYNC_END are specified, then only one MPITrajectory structure will be used for by mpiMotionStart() and mpiMotionModify(). Please refer to MPIMotionAttr for more information.

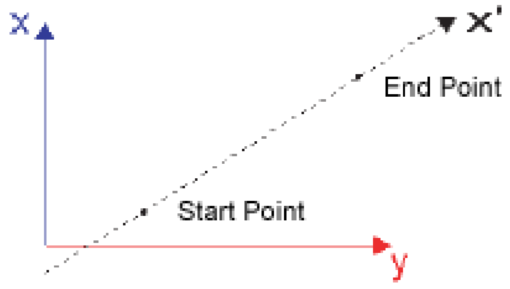
Trajectory











Definition: MEIMotionAttrMask

```
typedef enum {
    MEIMotionAttrMaskEVENT,
    MEIMotionAttrMaskFINAL_VEL,
    MEIMotionAttrMaskNO_REVERSAL,
    MEIMotionAttrMaskHOLD,
    MEIMotionAttrMaskHOLD_LESS,
    MEIMotionAttrMaskHOLD_GREATER,
    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.
MEIMotionAttrMaskHOLD_LESS	Motion attribute mask for less than or equal hold logic.
MEIMotionAttrMaskHOLD_GREATER	Motion attribute mask for greater than or equal hold logic.
MEIMotionAttrMaskOUTPUT	This mask allows the user to set or clear bits during a motion.
MEIMotionAttrOUTPUT	This bit allows the user to set or clear bits during a motion.

See Also

mpiMotionStart | mpiMotionModify | MPIMotionType | MPITrajectory | MPIEventMask

MEIMotionAttrHold | MEIMotionAttrHoldSource | MEIMotionAttrHoldType

MEIMotionAttrOutput

Definition

```
typedef struct MEIMotionAttrOutput {
    MEIMotionAttrOutputType     type;
    union {
        long  *output;
        long  motion;
    } as;
    long   offMask;
    long   onMask;
    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.
offMask	This value specifies the bits to be turned OFF when MEIMotionAttrOutputTypeOFFMASK is used.
onMask	This value specifies the bits to be turned ON when MEIMotionAttrOutputTypeONMASK is used.
pointIndex	This value specifies an index to a point, when multiple point motion is used.

See Also

MEIMotionAttrOutputType

MEIMotionAttrOutputType

Definition

```
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

Definition: MPIMotionAttributes

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

Change History: Modified in the 03.04.00

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. The id is limited to 16-bit resolution by the controller firmware.
elementId	This array defines the identity for each element of a path motion. The elementId is limited to 16-bit resolution by the controller firmware.
masterStart	This field is only used with the MASTER_START motion attribute and when commanding a cam motion. This field defines the position of the master at the start of the cam motion. See Starting at Specific Master Positions.
repeatFrom	This field is only used with the REPEAT motion attribute and when commanding a cam motion. This field defines the first frame that is repeated. Any frames before this frame will act as a run-in sequence. See Repeating Cams .

Definition: 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

Definition

```
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 previous 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

Definition

```
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 previous 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

MPIMotionCam

Definition

```
typedef struct MPIMotionCam {
    long     pointCount;
    double   **slavePosition;
    double   *masterDistance;
    double   **gearRatio;
             /* This field is only used with MPIMotionTypeCUBIC_CAM. */
} MPIMotionCam;
```

Change History: Modified in the 03.04.00

Description

MPIMotionCam contains the cam segments for a linear or cubic interpolated cam move.

pointCount	This field defines the type of master position source that is being used.
****slavePosition**	This field defines the distance that each slave axis will move during each segment of the cam.
***masterDistance**	An array containing pointCount number of elements. Each element is the distance the master axis will move during each segment of the cam.
****gearRatio**	This field is only used with cubic cams, with linear cams this field is completely ignored and can be either be set to zero or not initialised.This field defines the gear ratio (first derivative of the slave position with respect to the master position) to the cam at each of the transitions between cam segments. The initial gear ration is assumed to be zero.

See Also

MPIMotionConfig / MEIMotionConfig

Definition: 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.

Definition: MEIMotionConfig

```
typedef struct  MEIMotionConfig {
    char    userLabel[MEIObjectLabelCharMAX+1];
    long    axisCount;
    long    axisNumber[MEIXmpMAX_COORD_AXES];
    double  blendLimit;
} MEIMotionConfig;
```

Change History: Modified in the 03.03.00

Description

userLabel	This value consists of 16 characters and is used to label the motion object for user identification purposes. The userLabel field is NOT used by the controller.
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

Sample Code

```
void modifyFeedrate(MPIMotion motion,
float normalFeedrate,
float pauseFeedrate)
{
MPIMotionConfig motionConfig;
long returnValue;

returnValue = mpiMotionConfigGet(motion,
&motionConfig,
NULL);
msgCHECK(returnValue);

printf("Before: normalFeedrate %lf, pauseFeedrate %lf\n",motionConfig.
normalFeedrate,motionConfig.pauseFeedrate);

motionConfig.normalFeedrate = normalFeedrate;
motionConfig.pauseFeedrate = pauseFeedrate;

returnValue = mpiMotionConfigSet(motion,
&motionConfig,
NULL);
msgCHECK(returnValue);

printf("After: normalFeedrate %lf, pauseFeedrate %lf\n",motionConfig.
normalFeedrate,motionConfig.pauseFeedrate);
}
```

See Also

MPIMotionDecelTime | mpiMotionModify

MPIMotionDecelTime

Definition

```
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 0.5 seconds. This value is also used during a change from one feedrate to another feedrate with the normal feedrate parameter. The normal feedrate parameter is described in MPIMotionConfig. Note that the stop time is for a 100% change in the feedrate. Any changes in feedrate which are less than 100% are automatically scaled. This applies to both a change in feedrate due to a Stop action, or a change in feedrate due to a modification of the normal feedrate parameter. For example, if the stop time is defined as 0.5 seconds and the normal feedrate is changed from 1.0 to 0.5, it will take 0.25 seconds to reach the new feedrate of 0.5. If the axis is moving with a 0.5 normal feedrate and a stop action occurs, it will also take 0.25 seconds to ramp from a feedrate of 0.5 to a feedrate of 0.0.
eStop	This value defines the deceleration time (seconds) for an E-Stop action. The default value is 0.05 seconds.

See Also

MEIMotionFrame

Definition

```
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

Definition

```
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

Definition: MPIMotionMessage

```
typedef enum {
    MPIMotionMessageMOTION_INVALID,
    MPIMotionMessageAXIS_NOT_FOUND,
    MPIMotionMessageAXIS_COUNT,
    MPIMotionMessageAXIS_FRAME_COUNT,
    MPIMotionMessageTYPE_INVALID,
    MPIMotionMessageATTRIBUTE_INVALID,
    MPIMotionMessageIDLE,
    MPIMotionMessageMOVING,
    MPIMotionMessageSTOPPING,
    MPIMotionMessageSTOPPED,
    MPIMotionMessageSTOPPING_ERROR,
    MPIMotionMessageERROR,
    MPIMotionMessageAUTO_START,
    MPIMotionMessageILLEGAL_DELAY,
    MPIMotionMessagePROFILE_ERROR,
    MPIMotionMessagePROFILE_NOT_SUPPORTED,
    MPIMotionMessagePATH_ERROR,
    MPIMotionMessageFRAMES_LOW,
    MPIMotionMessageFRAMES_EMPTY,
    MPIMotionMessageFRAME_BUFFER_TOO_SMALL,
} MPIMotionMessage;
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00. Modified in the 03.02.00

Description

MPIMotionMessage is an enumeration of Motion error messages that can be returned by the MPI library.

MPIMotionMessageMOTION_INVALID

The motion supervisor number is out of range. This message code is returned by mpiMotionCreate (...) if the motion supervisor number is less than zero or greater than or equal to MEIXmpMAX_MSs.

MPIMotionMessageAXIS_NOT_FOUND

The specified axis object is not available. This message is returned from mpiMotionAxisRemove(...) if the axis that is being removed is not a member of the motion object.

MPIMotionMessageAXIS_COUNT

The number of axes is out of range. This message is returned from mpiMotionConfigSet(...), mpiMotionStart(...), or mpiMotionModify(...) if there are no axes associated with the motion object or if the axis count exceeds MEIXmpMAX_COORD_AXES.

MPIMotionMessageAXIS_FRAME_COUNT

The axis frame count invalid on this Motion object. All axes appended to a Motion object must have the same frame buffer size. The axis frame buffer sizes are configured with the low-level controller configuration routine mpiControlConfigSet(...).

MPIMotionMessageTYPE_INVALID

The motion type or motion attribute is not valid. This message code is returned from mpiMotionStart (...) or mpiMotionModify(...) if the motion type or motion attribute mask is not recognized by the library. To correct the problem, select a motion type/attribute from the MPI and/or MEI enumerations.

MPIMotionMessageATTRIBUTE_INVALID

The motion attribute is not valid. This message code is returned from mpiMotionStart(...) or mpiMotionModify(...) if the motion attribute mask is not compatible with the specified motion type. To correct the problem, do not use the motion attribute mask with the specified motion type or select a different motion type.

Please see possible causes for receiving this message.

MPIMotionMessageIDLE

All motion supervisor axes are not moving and are ready to move. This message code is returned from mpiMotionModify(...) when the motion supervisor is in the IDLE state. A motion cannot be modified if no motion is in progress. To correct the problem, use the AUTO_START attribute for mpiMotionModify(...) or use mpiMotionStart(...) instead. This message code is also returned from mpiMotionAction(...) when a STOP or RESUME is commanded when the motion supervisor is in the IDLE state. A motion cannot be stopped if there is no motion in progress and a motion cannot be resumed if there is no motion profile pending.

MPIMotionMessageMOVING

At least one motion supervisor axis is moving. This message code is returned from mpiMotionStart(...) when the motion supervisor is in the MOVING state. A motion cannot be started if a motion is in progress. To correct the problem, use mpiMotionModify(...) instead. This message code is also returned from mpiMotionAction(...) when a RESUME or RESET is commanded when the motion supervisor is in the MOVING state. A motion cannot be resumed or reset while a motion is in progress.

MPIMotionMessageSTOPPING

Motion supervisor axes are stopping due to a STOP action. This message code is returned from mpiMotionStart(...) or mpiMotionModify(...) when the motion supervisor is in the STOPPING state. A motion cannot be commanded when a stop is in progress. This message code is also returned from mpiMotionAction(...) when a RESUME or RESET is commanded when the motion supervisor is in the STOPPING state. A motion cannot be resumed or reset while a stop is in progress.

MPIMotionMessageSTOPPED

Motion supervisor axes are stopped due to a STOP action. This message code is returned from mpiMotionAction(...) when a STOP action is commanded while the motion supervisor is already in the STOPPED state.

MPIMotionMessageSTOPPING_ERROR

Motion supervisor axes are stopping due to an E_STOP or ABORT action. This message code is returned from mpiMotionStart(...) or mpiMotionModify(...) when the motion supervisor is in the STOPPING_ERROR state. A motion cannot be commanded when a stopping on error action is in progress. This message code is also returned from mpiMotionAction(...) when a RESUME or RESET is commanded when the motion supervisor is in the STOPPING state. A motion cannot be resumed or reset while a stopping on error action is in progress.

MPIMotionMessageERROR

Motion supervisor axes are in an error state due to an E_STOP or ABORT action. This message code is returned from mpiMotionStart(...) or mpiMotionModify(...) when the motion supervisor is in the ERROR state. A motion cannot be commanded when the motion supervisor is in an error state. This message code is also returned from mpiMotionAction(...) when a STOP or RESUME is commanded when the motion supervisor is in the ERROR state. To correct the problem, fix the condition that caused the E_STOP or ABORT action and then clear the ERROR state using mpiMotionAction(...) with a RESET.

MPIMotionMessageAUTO_START

The motion modify was automatically converted to a motion start. This message code is returned from mpiMotionModify(...) when the AUTO_START attribute mask was specified and the motion was commanded while the motion supervisor is in the IDLE state. This message code is useful for notifying an application of an auto-start, it is not an error condition.

MPIMotionMessageILLEGAL_DELAY

Returned if a motion modify is called with a non-zero delay value and the MPIMotionAttrDELAY attribute is set while in motion. If it is currently not in motion and the AUTO_START attribute is set, this message will not be returned and the specified delay will be used at the beginning of the motion.

MPIMotionMessagePROFILE_ERROR

The motion profile is not possible with the specified constraints. This message code is returned by mpiMotionModify(...) when the NO_REVERSAL attribute mask is specified, but the specified target position is in the reverse direction. To correct the problem, either remove the NO_REVERSAL constraint or specify a target position that is in the same direction as the motion profile in progress.

MPIMotionMessagePROFILE_NOT_SUPPORTED

The controller firmware does not support the specified motion profile type. This message code is returned by mpiMotionStart(...) or mpiMotionModify(...) when an S_CURVE_JERK or VELOCITY_JERK motion type is commanded, but not supported by the controller firmware. Due to programming memory space constraints, specific revisions of controller firmware may not support jerk motion types. To correct the problem, use the S_CURVE or VELOCITY motion instead.

MPIMotionMessagePATH_ERROR

The specified multi-point motion path is not valid. This message code is returned by mpiMotionStart(...) or mpiMotionModify(...) when the final point is not specified or the time slice is less than one controller sample.

MPIMotionMessageFRAMES_LOW

The controller's frame buffer is low. This message code is returned by the internal method, meiMotionFramesLow(...). This is an internal message code used by the library to manage the frame buffering.

MPIMotionMessageFRAMES_EMPTY

The controller's frame buffer is empty. This message code is returned by the internal method, meiMotionFramesLow(...). This is an internal message code used by the library to manage the frame buffering.

MPIMotionMessageFRAME_BUFFER_TOO_SMALL

When trying to start a cam, an insufficient amount of space was found on the controller. To remedy this problem, you can either reduce the number of points within the cam (See Camming) or increase the number of points in the frame buffer. See Increasing the Maximum Cam Table Size.

Definition: MEIMotionMessage

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

Required Header: stdmei.h
Change History: 03.03.00

Description

MEIMotionMessage is an enumeration of Motion error messages that can be returned by the MPI library.

MEIMotionMessageRESERVED0

Reserved for specialized use.

MEIMotionMessageRESERVED1

Reserved for specialized use.

MEIMotionMessageRESERVED2

Reserved for specialized use.

MEIMotionMessageNO_AXES_MAPPED

The motion object has no axes. This message code is returned by mpiMotionStart(...), mpiMotionModify(...) or mpiMotionAction(...) if there are no axes mapped to the motion object. To correct this problem, make sure there is at least one axis object associated with the motion object before commanding any motion or motion actions.

Possible Causes:
No axes are mapped to the motion supervisor that the move was commanded on.

MEIMotionMessageBAD_PATH_DATA

If any of the motion parameters passed to the MPI for path motion are infinite or NAN, the MPI will return MEIMotionMessageBAD_PATH_DATA and will not load the move to the controller.

See Also

MPIMotionType | MPIMotionAttrMask | mpiMotionModify | mpiMotionAction
mpiMotionStart

MPIMotionParams / MEIMotionParams

Definition: MPIMotionParams

```
typedef struct MPIMotionParams {

    MPIMotionPT         pt;
    MPIMotionPTF        ptf;
    MPIMotionPVT        pvt;
    MPIMotionPVTF       pvtf;
    MPIMotionSPLINE     spline;
    MPIMotionBESSEL     bessel;
    MPIMotionBSPLINE    bspline;

    MPIMotionSCurve         sCurve;
    MPIMotionSCurve         sCurveJerk;
    MPIMotionTrapezoidal    trapezoidal;

    MPIMotionVelocity    velocity;
    MPIMotionVelocity    velocityJerk;

    MPIMotionCam     cam;

    MPIMotionAttributes     attributes;

    void                *external;
} MPIMotionParams;
```

Description

MPIMotionParams contains the motion trajectory parameters for each motion type and the motion attributes.

pt	This structure contains the parameters for a PT motion type. Please see MPIMotionPT data type for more information.
ptf	This structure contains the parameters for a PTF (position, time, and feedforward) motion type. Please see MPIMotionPTF data type for more information.
pvt	This structure contains the parameters for a PVT motion type. Please see MPIMotionPVT data type for more information.
pvtf	This structure contains the parameters for a PTF (position, velocity, time, and feedforward) motion type. Please see MPIMotionPVTF 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.
sCurveJerk	This structure contains the parameters for an S-Curve Jerk point to point motion type. Please see MPIMotionSCurveJerk data type for more information.
trapezoidal	This structure contains the parameters for a Trapezoidal point to point motion type. Please see MPIMotionTrapezoidal 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.
cam	This structure contains the parameters for a cam motion type. Please see MPIMotionCAM 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.

Definition: 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

PT and PVT Path Motion

MPIMotionPoint

Definition

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

Change History: Modified in the 03.03.00

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 controller's buffer to trigger an E_STOP action and a MOTION_OUT_OF_FRAMES event. If ALL the points for a move fit into the controller's buffer, the emptyCount can be disabled with a value of 0. If ALL the points for a move do NOT fit into the controller's buffer, you must set the emptyCount to a non-zero value. Any points that do not fit into the controller's buffer will be streamed from the host to the controller via the eventMgr. The controller monitors the emptyCount to determine if it will run out of points. If the number of frames left in the controller's buffer is less than the emptyCount, an E_STOP action will occur for all axes mapped to the Motion Supervisor. An E_STOP will decelerate the axes to a stop along the path of the controller's remaining points. The emptyCount must be set to a large enough value to allow the E_STOP to stop motion BEFORE the end of the point list is reached. For example, if each point takes 5 milliseconds to execute and an E_STOP is configured for 20 milliseconds, then emptyCount should be 4 (or higher). The valid range is 0 to the controller's axis frame buffer size.

See Also

mpiMotionModify | MPIMotionAttrMaskAPPEND | MPIMotionPT

MPIMotionPT

Definition

```
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 previous 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 | Streaming Point Motion Type Calculations

PT and PVT Path Motion

MPIMotionPTF

Definition

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

Description

MPIMotionPTF contains the motion parameters for the MPIMotionTypePTF.

The feedforward values are interpolated linearly over the PT or PVT time intervals. The feedfoward values correspond to the P or PV values. (i.e. When the motion reaches a specified position (PTF) or position and velocity (PVTF), the interpolated feedforward value will be equal to what is specified in the motion parameters.)

The feedforward values are not set to zero at the beginning of the move; they retain the last value that is specified in the PTF or PVTF motion parameters.

The feedforward values are not zeroed by non-PTF or PVTF moves (i.e. PT, PVT, Spline, S-Curve, etc.).

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.
***feedforward**	This array stores the feedforward values for the motion profile. There is one feedforward value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The feedforward values are interleaved in the array by the axis index. The units are raw DAC counts (range -32768 to +32767).
***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 previous 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

MPIMotionParams | MPIMotionType | MPIMotionPoint | mpiMotionStart | mpiMotionModify

MPIMotionPVT

Definition

```
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 velocities for the motion profile. There is one velocity value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The velocities 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 previous 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 | Streaming Point Motion Type Calculations

PT and PVT Path Motion

MPIMotionPVTF

Definition

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

Description

MPIMotionPVTF contains the motion parameters for the MPIMotionTypePVTF.

The feedforward values are interpolated linearly over the PT or PVT time intervals. The feedfoward values correspond to the P or PV values. (i.e. When the motion reaches a specified position (PTF) or position and velocity (PVTF), the interpolated feedforward value will be equal to what is specified in the motion parameters.)

The feedforward values are not set to zero at the beginning of the move; they retain the last value that is specified in the PTF or PVTF motion parameters.

The feedforward values are not zeroed by non-PTF or PVTF moves (i.e. PT, PVT, Spline, S-Curve, etc.).

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 velocities for the motion profile. There is one velocity value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The velocities are interleaved in the array by the axis index.
***feedforward**	This array stores the feedforward values for the motion profile. There is one feedforward value per point, per axis. The length of the array must be equal to pointCount multiplied by the number of axes. The feedforward values are interleaved in the array by the axis index. The units are raw DAC counts (range -32768 to +32767).

***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 previous 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

MPIMotionParams | MPIMotionType | MPIMotionPoint | mpiMotionStart | mpiMotionModify

MPIMotionSCurve

Definition

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

Description

MPIMotionSCurve contains the motion trajectory parameters and final target positions that specify the point-to-point motion profile. Only the velocity, acceleration, deceleration, and jerkPercent trajectory parameters are applicable to S-Curve motion profiles.

***trajectory**	A pointer to an array of trajectory structures. Each trajectory structure contains the motion profile parameters for each axis associated with the motion object. If more than one axis is associated with the motion and neither the MPIMotionAttrMaskSYNC_START nor MPIMotionAttrMaskSYNC_END attributes are specified, then only the first trajectory structure will be applied as the vector trajectory for all the axes.
***position**	A pointer to an array of target positions. Each position value specifies the target for each axis associated with the motion object. Some motion types, like VELOCITY and VELOCITY_JERK do not require target positions and will ignore these values.

See Also

MPITrajectory | MPIMotionType | mpiMotionStart | mpiMotionModify

MPIMotionSCurveJerk

Definition

```
typedef MPIMotionSCurve MPIMotionSCurveJerk;
```

Description

MPIMotionSCurveJerk contains the motion trajectory parameters and final target positions that specify the point-to-point motion profile. Only the velocity, acceleration, deceleration, accelerationJerk, and decelerationJerk trajectory parameters are applicable to S-Curve Jerk motion profiles.

See Also

MPITrajectory | MPIMotionType | mpiMotionStart | mpiMotionModify

MPIMotionSPLINE

Definition

```
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 previous 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

Definition

```
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

Definition

```
typedef MPIMotionSCurve MPIMotionTrapezoidal;
```

Description

MPIMotionTrapezoidal structure contains the motion trajectory parameters and final target positions that specify the point-to-point motion profile. Only the velocity, acceleration, and deceleration trajectory parameters are applicable to Trapezoidal motion profiles.

See Also

MPITrajectory | MPIMotionType | mpiMotionStart | mpiMotionModify

MPIMotionType and MEIMotionType

Definition: MPIMotionType

```
typedef enum {
    MPIMotionTypeINVALID,

    MPIMotionTypePT,
    MPIMotionTypePTF,
    MPIMotionTypePVT,
    MPIMotionTypePVTF,
    MPIMotionTypeSPLINE,
    MPIMotionTypeBESSEL,
    MPIMotionTypeBSPLINE,
    MPIMotionTypeBSPLINE2,

    MPIMotionTypeS_CURVE,
    MPIMotionTypeTRAPEZOIDAL,
    MPIMotionTypeS_CURVE_JERK,

    MPIMotionTypeVELOCITY,
    MPIMotionTypeVELOCITY_JERK,
    MPIMotionTypeCAM_LINEAR,
    MPIMotionTypeCAM_CUBIC,

    MPIMotionTypeMASK  = 0xFF,
} MPIMotionType;
```

Description

MPIMotionType is an enumeration of motion profile types. It specifies the motion profile to be generated with mpiMotionStart(...) and mpiMotionModify(...). The motion type also defines the trajectory parameters that must be passed to mpiMotionStart(...) and mpiMotionModify(...). For each MPIMotionType and MEIMotionType there is a corresponding element in the MPIMotionParams{...} or MEIMotionParams{...} structure. For example, when MPIMotionTypeS_CURVE is specified, the MPIMotionSCurve{...} structure must be filled in to specify the trajectory for the S-Curve profile.

The motion profile characteristics can be specified by bitwise OR-ing the MPIMotionType with one or more motion attribute masks. The MPIMotionAttrMask and MEIMotionAttrMask enumerations contain the attribute masks. Attribute masks can be used to enable special features or configure the motion profile calculation and execution properties.

Internally, the motion profile is calculated and broken up into smaller trajectory segments called frames. The frames contain position, time, velocity, acceleration, and jerk trajectory information. The controller buffers the frames in its memory and executes them by loading the values into its trajectory calculator. For simple MotionTypes, like VELOCITY, TRAPEZOIDAL, and S_CURVE, the frames are calculated by the controller. For complex, multi-point motions, like PT, PVT, SPLINE, etc. the frames are calculated in the MPI Library. MotionTypes that are calculated in the controller can be modified on-the-fly with mpiMotionModify(...).

The move types, MPIMotionTypeS_CURVE_JERK and MPIMotionTypeVELOCITY_JERK are only available with custom firmware (option 21). In the custom firmware, the Velocity/Trapezoidal/S-Curve algorithm is replaced with the S-Curve Jerk algorithm. In the standard firmware, the MPIMotionTypeS_CURVE_JERK and MPIMotionTypeVELOCITY_JERK motion types are not supported and if specified, the MPI will return a "profile not supported" error.

Fore details, see S-Curve Jerk Algorithm and Attributes.

Example
Current Position = 2147483638 (2^{31} - 10)
First Position in Point List = 2147483658 ((2^{31} + 10)
or First Position in Point List = -2147483638 ((2^{31} + 10) - 2^{32})

From the controller's point of view, both first positions are the same value, but from the MPI's point of view, they are 2^{32} apart. With the shortest path algorithm, -2147483638 is converted to 2147483658 so that the motion between the current position and the first position is smooth and does not cause excessive acceleration or velocity.

NOTE: The shortest path algorithm only applies to the delta between the current position and the first point in a points list (path motion). It DOES NOT apply to points within the points list.

MPIMotionTypePT	This type fits constant velocity profile segments through a list of position and time points. Not supported in motion sequences. Please see MPIMotionPT for more information.
MPIMotionTypePTF	PTF motion is identical to PT except host-calculated feedforward values can be specified for each PTF motion segment.
MPIMotionTypePVT	This type fits jerk profile segments through a list of position, velocity and time points. Not supported in motion sequences. Please see MPIMotionPVT for more information.
MPIMotionTypePVTF	PVTF motion is identical to PVT except host-calculated feedforward values can be specified for each PVTF motion segment.
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.
MPIMotionTypeCAM_LINEAR	This type generates linear interpolated cam motion. See Camming.
MPIMotionTypeCAM_CUBIC	This type generates cubic interpolated cam motion. See Camming.

Definition: MEIMotionType

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

Description

MEIMotionType is an enumeration of the controller specific motion profile types. See MPIMotionType for details.

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

See Also

mpiMotionStart | mpiMotionModify | mpiMotionTYPE | MPIMotionParams | MEIMotionParams | MPIMotionAttr | MEIMotionAttr | Streaming Point Motion Type Calculations

MPIMotionVelocity

Definition

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

Description

MPIMotionVelocity contains the motion trajectory parameters that specify velocity motion profiles. Only the velocity, acceleration, jerkPercent, and accelerationJerk trajectory parameters are applicable to velocity and velocity-jerk motion profiles.

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

See Also

MPITrajectory | MPIMotionType | mpiMotionStart | mpiMotionModify

mpiMotionATTR

Declaration

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

Required Header: stdmpi.h

Description

mpiMotionATTR 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

mpiMotionAttrMaskBIT 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

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

See Also

MPIMotionType

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.

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

MEI Motion Attributes

MEIMotionAttrEVENT

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskEVENT attribute. This mask allows the user to specify an MPIEventMask during a motion.

MEIMotionAttrFINAL_VEL

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskFINAL_VEL attribute. This mask allows the user to specify a non-zero target velocity for point to point motion types.

MEIMotionAttrNO_REVERSAL

The MPI has been extended to support mpiMotionStart(...) with the MEIMotionAttrMaskNO_REVERSAL attribute. This mask prevents a motion profile from changing direction.

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 mask allows the user to set or clear bits during a motion. 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.

Return to Motion Objects page