Axis Objects

Introduction

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

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

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

| Error Messages |

Methods

Create, Delete, Validate Methods

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

Configuration and Information Methods

mpiAxis**ActualPositionGet**	Get actual position
mpiAxis**ActualPositionGet32**	Gets the lower 32 bits of actual position
mpiAxis**ActualPositionSet**	Set actual position
mpiAxis**ActualVelocity**	Get actual velocity
mpiAxis**ConfigGet**	Get Axis configuration
mpiAxis**ConfigSet**	Set Axis configuration
mpiAxis**CommandPositionGet**	Get command position
mpiAxis**CommandPositionGet32**	Gets the lower 32 bits of command position
mpiAxis**CommandPositionSet**	Set command position
mpiAxis**FlashConfigGet**	Get Axis flash config
mpiAxis**FlashConfigSet**	Set Axis flash config
mpiAxis**FlashOriginGet**	
mpiAxis**FlashOriginSet**	
meiAxis**FrameBufferStatus**	
mpiAxis**OriginGet**	Get Axis origin

mpiAxis**OriginSet**	Set Axis origin
mpiAxis**PositionError**	Get position error of an Axis
mpiAxis**Status**	Get Axis status
mpiAxis**Trajectory**	Get Axis trajectory

Event Methods

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

Memory Methods

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

Relational Methods

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

Data Types

MPIAxis**Config** / MEIAxis**Config**
MPIAxis**EstopModify**
MEIAxis**FrameBufferStatus**
MPIAxis**InPosition**
MPIAxis**Master**
MPIAxis**MasterType**
MPIAxis**Message**

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

Constants

MEI**PreFilterCoeffsMAX**

MEI**PreFilterCountMAX**

mpiAxisCreate

Declaration

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

Required Header: stdmpi.h

Description

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

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

Remarks

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

Return Values	
handle	to an Axis object
MPIHandleVOID	

See Also

mpiAxisDelete | mpiAxisValidate

mpiAxisDelete

Declaration

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

Required Header: stdmpi.h

Description

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

axis	the Axis handle to be deleted

Remarks

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

Return Values	
MPIMessageOK	

See Also

mpiAxisCreate | mpiAxisValidate

mpiAxisValidate

Declaration

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

Required Header: stdmpi.h

Description

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

axis	a handle to the Axis object to be validated

Return Values	
MPIMessageOK	

See Also

mpiAxisCreate | mpiAxisDelete

mpiAxisActualPositionGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

AxisCommandPositionSet | Using the Origin Variable

mpiAxisActualPositionGet32

Declaration

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

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

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisActualPositionGet | AxisCommandPositionSet | Using the Origin Variable

mpiAxisActualPositionSet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

AxisCommandPositionSet | Using the Origin Variable | Controller Positions

mpiAxisActualVelocity

Declaration

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

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiAxisConfigGet

Declaration

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

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

Description

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

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

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

Return Values	
MPIMessageOK	

Remarks

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

Sample Code

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

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

See Also

MPIAxisConfig | mpiAxisConfigSet | MEIAxisConfig

mpiAxisConfigSet

Declaration

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

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

Description

mpiAxisConfigSet sets the configuration of an Axis (***axis***) using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

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

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

Return Values	
MPIMessageOK	

Remarks

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

Sample Code

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

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

See Also

mpiAxisConfigGet | MEIAdcConfig | MEIAxisConfig

mpiAxisCommandPositionGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisCommandPositionSet | Controller Positions

mpiAxisCommandPositionGet32

Declaration

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

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

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisCommandPositionGet | mpiAxisCommandPositionSet | Controller Positions

mpiAxisCommandPositionSet

Declaration

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

Required Header: stdmpi.h

Description

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

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

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

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

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

Remarks

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

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

See Also

MEIMotorDisableAction | AxisActualPositionSet | AxisCommandPositionSet | MPIAxisMessage | Controller Positions

mpiAxisFlashConfigGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

MEIFlash | mpiAxisFlashConfigSet | MEIAxisConfig

mpiAxisFlashConfigSet

Declaration

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

Required Header: stdmpi.h

Description

mpiAxisFlashConfigSet sets the flash configuration for for an Axis (***axis***) using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

MEIFlash | mpiAxisFlashConfigGet | MEIAxisConfig

mpiAxisFlashOriginGet

Declaration

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

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

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisFlashOriginSet | mpiAxisOriginGet | Using the Origin Variable | Controller Positions

mpiAxisFlashOriginSet

Declaration

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

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

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisFlashOriginGet | mpiAxisOriginSet | Using the Origin Variable | Controller Positions

meiAxisFrameBufferStatus

Declaration

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

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

Description

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

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

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessageHANDLE_INVALID	

Sample Code

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

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

See Also

MEIAxisFrameBufferStatus

mpiAxisOriginGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisOriginSet | Using the Origin Variable | Controller Positions

mpiAxisOriginSet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisOriginGet | Using the Origin Variable | Controller Positions

mpiAxisPositionError

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisCommandPositionGet | mpiAxisActualPositionGet | Controller Positions

mpiAxisStatus

Declaration

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

Required Header: stdmpi.h

Description

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

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

Remarks

external should always be set to NULL.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiAxisCommandPositionGet | mpiAxisActualPositionGet | Controller Positions

mpiAxisTrajectory

Declaration

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

Required Header: stdmpi.h

Description

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

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

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

Remarks

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

Sample Code

```
MPITrajectory trajectory;

    mpiAxisTrajectory(axis, &trajectory);

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

Return Values	
MPIMessageOK	

See Also

mpiMotionStart | mpiMotionModify | MPITrajectory

mpiAxisEventNotifyGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

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

Remarks

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

The **MEIEventNotifyData{}** structure is an array of firmware addresses, whose contents are placed into the **MEIEventStatusInfo{}** structure (of all events generated by this object).

Return Values	
MPIMessageOK	

See Also

MEIEventNotifyData | MEIEventStatusInfo | mpiAxisEventNotifySet

mpiAxisEventNotifySet

Declaration

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

Required Header: stdmpi.h

Description

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

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

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

Remarks

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

The **MEIEventNotifyData{}** structure is an array of firmware addresses, whose contents are placed into the **MEIEventStatusInfo{}** structure (of all events generated by this object).

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

Return Values	
MPIMessageOK	

See Also

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

mpiAxisEventReset

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

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

Event Notification Methods

mpiAxisMemory

Declaration

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

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiAxisMemoryGet | mpiAxisMemorySet

mpiAxisMemoryGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisMemory | mpiAxisMemorySet

mpiAxisMemorySet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisMemory | mpiAxisMemoryGet

mpiAxisControl

Declaration

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

Required Header: stdmpi.h

Description

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

axis	a handle to an Axis object.

Return Values	
MPIHandleVOID	

See Also

mpiAxisFilterMapGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

mpiAxisFilterMapSet

mpiAxisFilterMapSet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

mpiAxisFilterMapGet | MPIObjectMap

mpiAxisMotorMapGet

Declaration

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

Required Header: stdmpi.h

Description

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

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

Remarks

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

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

Return Values	
MPIMessageOK	

See Also

mpiAxisFilterMapGet | MPIObjectMap

mpiAxisNumber

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

MPIAxisConfig / MEIAxisConfig

Definition: MPIAxisConfig

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

Change History: Modified in the 03.03.00.

Description

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

Definition: MEIAxisConfig

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

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

Description

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

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

PreFilter

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

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

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

Example:

```
MEIXmpData         *firmware;
MEIXmpBufferData   *bufferData;

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

- **Ratio.A** - numerator of multiplier
- **Ratio.B** - denominator of multiplier

- **Ratio.Old** -
- **Ratio.Remainder** -
- **Position** - final geared position

GantryType -

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

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

See Also

mpiAxisConfigGet | mpiAxisConfigSet | MPIAxisInPosition

MPIAxisEstopModify

Definition

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

Change History: Added in the 03.03.00

Description

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

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

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

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

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

See Also

MPIAxisConfig | mpiMotorEventConfigSet | mpiMotorEventConfigGet | MPIAction

MEIAxisFrameBufferStatus

Definition

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

Change History: Added in the 03.04.00.

Description

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

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

See Also

meiAxisFrameBufferStatus | mpiControlConfigGet | mpiControlConfigSet

MPIAxisInPosition

Definition

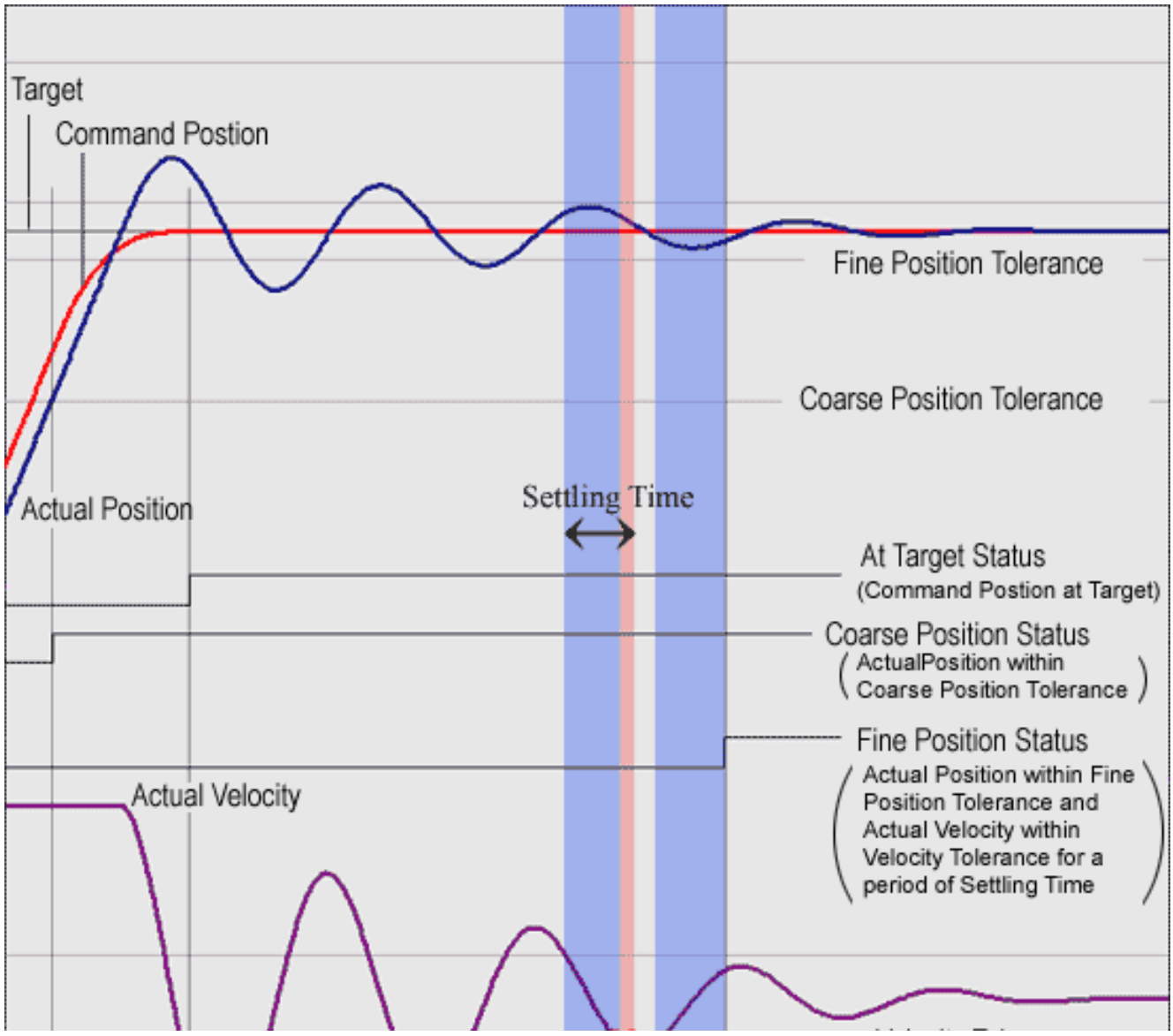
```
typedef struct MPIAxisInPosition {
    struct {
        float   positionFine;
        long    positionCoarse;
        float   velocity;
    } tolerance;
        float   settlingTime;  /* seconds */
        long    settleOnStop;
        long    settleOnEstop
        long    settleOnEstopCmdEqAct;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

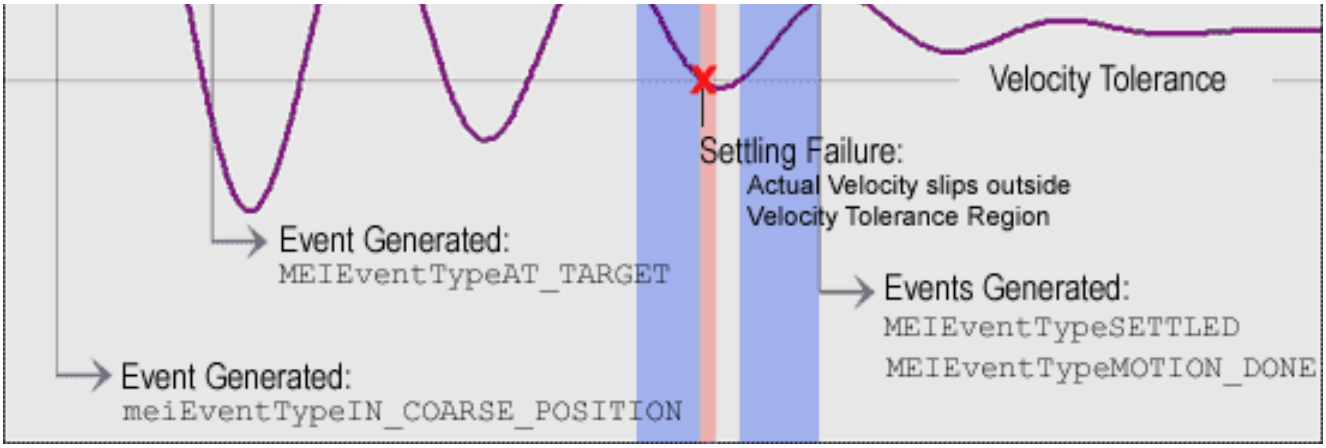
Description

tolerance	Includes the following 3 elements that determine settling tolerances for an axis.
positionFine	Value, in counts, from the move target position at which the controller sets the "in fine position" status flag. This parameter is used as part of the Axis settling criteria to determine when a point-to-point motion is complete and when MPIEventTypeMOTION_DONE and MEIEventTypeSETTLEDevents are generated.
positionCoarse	Value, in counts, from a move target position at which the controller sets the "in coarse position" status flag. This value does not affect the settling time status.
velocity	Value, in counts/second, from the final move velocity at which the controller sets the "at velocity" status flag. This parameter is used as part of the Axis settling criteria to determine when: • a position-based move is complete and an MPIEventTypeMOTION_DONE event is generated. • a velocity move is complete and an MPIEventTypeMOTION_AT_VELOCITY event is generated. • an axis is settled and an MPIEventTypeSETTLED event is generated. **NOTE**: Always enter a Tolerance Velocity value that is a multiple of the controller sample rate. The controller will then receive velocity in counts/ controller sample. 1 Counts/second = (1 counts/second) * (1/sample rate(Hz)) = (1/sample rate) counts/controller sample

	NOTE: Value is truncated to the next smallest integer. **Example**: With a sample rate of 2000Hz, • a Tolerance Velocity value of 500 counts/second = 0.25 = 0 count/ controller sample (after truncation). • a Tolerance Velocity value of 2000 counts/second = 1 count/ controller sample.
settlingTime	Duration in seconds that an axis must satisfy the positionFine and/or velocity tolerance, before the respective status flag is set.
settleOnStop	If TRUE, the controller will use settle on stop mode. If FALSE, the controller will not use the settle on stop mode. When in settleOnStop mode and a STOP event has occurred, the axis will stay in an MPIStateSTOPPING state until: • The settling criteria are satisfied AND. • The stop duration for the axis' Motion Supervisor has elapsed. • This state can be read with mpiAxisStatus(MPIAxis axis, MPIStatus *status, void *external). The value to look for is (MPIState) status.state. If settleOnStop = FALSE, the axis will stay in an MPIStateSTOPPING state only until the stop duration for the axis' Motion Supervisor has elapsed.
settleOnEstop	If TRUE, the controller will use settle on Estop mode. If FALSE, the controller will not use the settle on Estop mode. When in settleOnEstop mode and a ESTOP event has occurred, the axis will stay in an MPIStateSTOPPING_ERROR state until: • The settling criteria are satisfied AND. • The Estop duration for the axis' Motion Supervisor has elapsed. • This state can be read with mpiAxisStatus(MPIAxis axis, MPIStatus *status, void *external). The value to look for is (MPIState) status.state. If settleOnEStop = FALSE, the axis will stay in an MPIStateSTOPPING_ERROR state only until the Estop duration for the axis' Motion Supervisor has elapsed.

settleOnEstopCmdEqAct	If TRUE, the controller will use settle on EstopCmdEqAct mode. If FALSE, the controller will not use the settle on EstopCmdEqAct mode. ***settleOnEstopCmdEqAct mode is not recommended*** SettleOnEstopCmdEqAct is an alternative to Estop mode. When this mode is enabled, the following things happen: • During normal motion, there is no difference. • During an Estop, Cmd Eq Act action, the command position is set equal to the actual position from the previous servo sample. This can have a damping effect in some systems with some tuning parameters, causing the stage to slow. The behavior of the stage in this mode can be vastly different than in normal servoing mode. Approach this mode with great caution. The axis will stay in this mode for the amount of time that the Axis' Motion Supervisor Estop time. • After the Estop time elapses, the axis' motors will disable the amplifiers.





Sample Code

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

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

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

    return returnValue;
}
```

See Also

MPIAxisConfig | MPIAction

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

Configuration of IN_POSITION and Done Events after STOP or E_STOP Events

MPIAxisMaster

Definition

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

Description

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

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

MPIMasterType	Number	Address
MPIAxisMasterTypeMOTOR_FEEDBACK_PRIMARY	motor number	Not used
MPIAxisMasterTypeMOTOR_FEEDBACK_SECONDARY	motor number	Not used
MPIAxisMasterTypeAXIS_COMMANDED_POSITION	Axis number	Not used
MPIAxisMasterTypeAXIS_ACTUAL_POSITION	Axis number	Not used
MPIAxisMasterTypeADDRESS	Not used	Any controller address
MPIAxisMasterTypeNONE	Not used	Not used

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

See Also

MPIAxisMasterType

MPIAxisMasterType

Definition

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

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

Description

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

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

See Also

MPIAxisMaster

MPIAxisMessage

Definition

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

Description

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

MPIAxisMessageAXIS_INVALID

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

MPIAxisMessageCOMMAND_NOT_SET

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

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

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

MPIAxisMessageNOT_MAPPED_TO_MS

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

See Also

MEIPreFilter

Definition

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

Change History: Added in the 03.04.00.

Description

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

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

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

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

See Also

MEIPreFilterForm | MEIControlConfig

MEIPreFilterForm

Definition

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

} MEIPreFilterForm;
```

Change History: Added in the 03.04.00.

Description

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

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

See Also

MEIPreFilter

MEIPreFilterCoeffsMAX

Definition

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

Change History: Added in the 03.04.00.

Description

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

See Also

MEIPreFilterCountMAX

MEIPreFilterCountMAX

Definition

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

Change History: Added in the 03.04.00.

Description

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

See Also

MEIPreFilterCoeffsMAX

CAN Objects

Introduction

The CAN object allow the user easy access to the I/O nodes connected to a controller's CANOpen interface.

If a controller does not support the CANOpen interface, the meiCanValidate function will return MEICanMessageINTERFACE_NOT_FOUND.

The CAN system uses the MEICanConfig and MEICanNodeConfig structures to hold all of the user configurable quantities. These structures are stored in non-volatile flash memory. When the XMP is released from reset (normally soon after the host powers up or after a call to mpiControlReset), the CAN Processor will initialize itself with data from MEICanConfig and MEICanNodeConfig before starting to scanning the network for nodes.

The functions meiCanConfigGet, meiCanConfigSet, meiCanNodeConfigGet and meiCanNodeConfigSet allow the user to modify the current configuration of the CAN Processor. meiCanFlashConfigGet and meiCanFlashConfigSet functions allow the user to modify the configuration that the CAN system will use after the next reset.

The MEICanVersion structure returns the version information about the CAN system on a controller.

After the CAN processor has finished scanning the network, it will have completed the MEICanNodeInfo structures for each node. The user can call the meiCanNodeInfo function to query this initial configuration for each of the nodes.

Bit Rate | Transmission Types | Bus State | CAN Hardware | Node Health |
Emergency Messages | Handling Events | CAN Hardware on the XMP | CAN Analog Values

| Error Messages |

Methods

Create, Delete, Validate Methods

meiCan**Create**	Create Can object
meiCan**Delete**	Delete Can object
meiCan**Validate**	Validate Can object

Configuration and Information Methods

meiCan**ConfigGet**	Get Can's configuration
meiCan**ConfigSet**	Set Can's configuration
meiCan**FlashConfigGet**	Get Can's flash configuration
meiCan**FlashConfigSet**	Set Can's flash configuration
meiCan**Status**	Get status of the CAN controller.
meiCan**Version**	Returns the version information about a controller's CAN system.
meiCan**Command**	Get Can's flash configuration
meiCan**NodeConfigGet**	Return a copy of the current configuration

meiCan**NodeConfigSet**	Update the current configuration that the specified CAN node is using.
meiCan**NodeFlashConfigGet**	Get the flash configuration of the Can node
meiCan**NodeFlashConfigSet**	Set the flash configuration of the Can node
meiCan**NodeStatus**	Get the instantaneous state of the local CAN interface.
meiCan**NodeInfo**	Return the node information after the XMP finishes scanning the network.

I/O Methods

- meiCan**NodeAnalogIn**
- meiCan**NodeAnalogOutGet**
- meiCan**NodeAnalogOutSet**
- meiCan**NodeDigitalIn**
- meiCan**NodeDigitalOutGet**
- meiCan**NodeDigitalOutSet**

Event Methods

meiCan**EventNotifyGet**	Get event mask of events for which host notification has been requested
meiCan**EventNotifySet**	Set event mask of events for which host notification will be requested

Firmware Methods

meiCan**FirmwareDownload**	Downloads firmware to the Can controller
meiCan**FirmwareErase**	Erases firmware on the Can controller
meiCan**FirmwareUpload**	Uploads firmware from the Can controller

Memory Methods

meiCan**Memory**	Get address to Can's memory
meiCan**MemoryGet**	Copy data from Can memory to application memory
meiCan**MemorySet**	Copy data from application memory to Recorder memory

Action Methods

- meiCan**Init**

Relational Methods

- meiCan**Control**
- meiCan**Number**

Data Types

- MEICan**BitRate**
- MEICan**BusState**

MEICan**Callback**
MEICan**Command**
MEICan**CommandType**
MEICan**Config**
MEICan**HealthType**
MEICan**Message**
MEICan**NodeConfig**
MEICan**NodeInfo**
MEICan**NodeInfoProductCode**
MEICan**NodeInfoVendor**
MEICan**NodeStatus**
MEICan**NodeType**
MEICan**NMTState**
MEICan**Status**
MEICan**TransmissionType**
MEICan**Version**

Constants

MEICan**NetworkMAX**

meiCanCreate

Declaration

```
MEICan meiCanCreate(MPIControl  control,
                    long        number);
```

Required Header: stdmei.h
Change History: Modified in the 03.02.00

Description

meiCanCreate creates a CAN object handle that is used subsequently to address the CAN network on this controller. You will need a valid CAN handle to use the MPI's CANOpen functionality.

control	a handle to the controller object that contains the CAN object.
number	the number of the CAN network on the specified controller. For most controllers with a single CAN network interface this will be zero. Network numbers are zero based.

Return Values	
handle	Handle to the CAN object created or MPIHandleVOID.
MPIHandleVOID	if the object could not be created

Sample Code

The following sample code shows the creation and destruction of a valid CAN handle.

```
MPIControl ControlHandle;
 MEICan CANHandle;
 long Result;

 /* Create, validate and initalise a handle to the controller. */
 ControlHandle = mpiControlCreate( MPIControlTypeDEFAULT, NULL );
      Result = mpiControlValidate( ControlHandle );
                           assert( Result == MPIMessageOK );

 Result = mpiControlInit( ControlHandle );
                  assert( Result == MPIMessageOK );

 /* Create and validate a handle to the CAN object. */
```

```
CANHandle = meiCanCreate( ControlHandle, 0 );
 Result = meiCanValidate( CANHandle );
                  assert( Result == MPIMessageOK );

/* Use the CAN object here */

/* Delete the CAN and Controller objects */
Result = meiCanDelete( CANHandle );
                assert( Result == MPIMessageOK );
Result = mpiControlDelete( ControlHandle );
                  assert( Result == MPIMessageOK );
```

See Also

mpiCanDelete | mpiCanValidate

meiCanDelete

Declaration

```
long meiCanDelete(MEICan  can);
```

Required Header: stdmei.h

Description

meiCanDelete deletes the specified CAN object.

can	handle to the CAN object to delete.

Return Values	
MPIMessageOK	

Sample Code

```
See meiCanCreate for an example of how to use meiCanDelete.
```

See Also

meiCanCreate | meiCanValidate

meiCanValidate

Declaration

```
long meiCanValidate(MEICan   can);
```

Required Header: stdmei.h

Description

meiCanValidate validates the specified CAN handle.

can	handle to the CAN object

Return Values	
MPIMessageOK	
MPIMessageUNSUPPORTED	

Sample Code

```
See meiCanCreate for an example of how to use meiCanValidate.
```

See Also

meiCanNodeInfo | meiCanNodeStatus

meiCanConfigGet

Declaration

```
long meiCanConfigGet(MEICan           can,
                     MEICanConfig*    config);
```

Required Header: stdmei.h

Description

meiCanConfigGet returns a copy of the current configuration of the CAN controller.

can	a handle to the CAN object
config	a pointer to the CAN configuration structure that will be filled in by this function..

Return Values	
MPIMessageOK	

See Also

meiCanConfigSet

meiCanConfigSet

Declaration

```
long meiCanConfigSet(MEICan           can,
                     MEICanConfig*    config);
```

Required Header: stdmei.h

Description

meiCanConfigSet updates the current configuration of the CAN controller.

can	a handle to the CAN object
config	a pointer to the CAN configuration structure containing the new configuration.

Return Values	
MPIMessageOK	

See Also

meiCanConfigGet

meiCanFlashConfigGet

Declaration

```
long meiCanFlashConfigGet(MEICan          can,
                          void*           flash,
                          MEICanConfig*   config);
```

Required Header: stdmei.h

Description

meiCanFlashConfigGet returns a copy of the current flash configuration that the CAN controller is using.

can	handle to the CAN object
flash	normally NULL
config	a pointer to the CAN configuration structure that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiCanFlashConfigSet

meiCanFlashConfigSet

Declaration

```
long meiCanFlashConfigSet(MEICan          can,
                          void*           flash,
                          MEICanConfig*   config);
```

Required Header: stdmei.h

Description

meiCanFlashConfigSet updates the current flash configuration that the CAN controller is using.

can	handle to the CAN object
flash	normally NULL
config	a pointer to the CAN configuration structure that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiCanFlashConfigGet

meiCanStatus

Declaration

```
long meiCanStatus(MEICan          can,
                  MEICanStatus*   status);
```

Required Header: stdmei.h

Description

meiCanStatus gets the instantaneous state of the local CAN interface to the CAN network.

can	handle to the CAN object
node	the node number of the CANOpen node.
status	a pointer to where this function will put the status.

Return Values	
MPIMessageOK	

See Also

meiCanNodeInfo | meiCanNodeStatus

meiCanVersion

Declaration

```
long meiCanVersion(MEICan           can,
                   MEICanVersion*   version);
```

Required Header: stdmei.h

Description

meiCanVersion returns the version of the firmware being used by the CAN controller.

can	handle to the CAN object
version	a pointer to where this function will put the version information.

Return Values	
MPIMessageOK	

See Also

meiCanCommand

Declaration

```
long meiCanCommand(MEICan            can,
                   MEICanCommand*    command);
```

Required Header: stdmei.h

Description

meiCanCommand allows a set of basic commands to be performed. The ***type*** field of the MEICanCommand structure specifies the type of command to perform.

can	a handle to the CAN object
command	a pointer to a structure which contains the details of the command to be issued. On the functions return, it will contain the result of the requested command.

Return Values	
MPIMessageOK	

See Also

MEICanCommand

meiCanNodeConfigGet

Declaration

```
long meiCanNodeConfigGet(MEICan              can,
                         long                node,
                         MEICanNodeConfig*   nodeConfig);
```

Required Header: stdmei.h

Description

meiCanNodeConfigGet returns a copy of the current configuration that the specified CAN node is using.

can	a handle to the CAN object
node	the node number of the CANOpen node
nodeConfig	a pointer to the CAN node configuration structure that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiCanNodeConfigSet | meiCanConfigGet | meiCanConfigSet

meiCanNodeConfigSet

Declaration

```
long meiCanNodeConfigSet(MEICan              can,
                         long                node,
                         MEICanNodeConfig*   nodeConfig);
```

Required Header: stdmei.h

Description

meiCanNodeConfigSet updates the current configuration that the specified CAN node is using.

can	a handle to the CAN object
node	the node number of the CANOpen node
nodeConfig	a pointer to the CAN node configuration structure containing the new configuration.

Return Values	
MPIMessageOK	

See Also

meiCanNodeConfigGet | meiCanConfigGet | meiCanConfigSet

meiCanNodeFlashConfigGet

Declaration

```
long meiCanNodeFlashConfigGet(MEICan             can,
                              void*              flash,
                              long               node,
                              MEICanNodeConfig*  nodeConfig);
```

Required Header: stdmei.h

Description

meiCanNodeFlashConfigGet returns a copy of the current flash configuration of the CAN controller.

can	a handle to the CAN object
flash	normally NULL
node	the node number of the CANOpen node
nodeConfig	a pointer to the CAN node configuration structure that will be filled in by this function

Return Values	
MPIMessageOK	

See Also

meiCanNodeFlashConfigSet

meiCanNodeFlashConfigSet

Declaration

```
long meiCanNodeFlashConfigSet(MEICan            can,
                              void*             flash,
                              long              node,
                              MEICanNodeConfig* nodeConfig);
```

Required Header: stdmei.h

Description

meiCanNodeFlashConfigSet updates the current flash configuration for the node.

can	a handle to the CAN object
flash	normally NULL
node	the node number of the CANOpen node
nodeConfig	a pointer to the CAN node configuration structure containing the new configuration.

Return Values	
MPIMessageOK	

See Also

meiCanNodeFlashConfigGet

meiCanNodeStatus

Declaration

```
long meiCanNodeStatus(MEICan              can,
                      long                node,
                      MEICanNodeStatus*   nodeStatus);
```

Required Header: stdmei.h

Description

meiCanNodeStatus gets the instantaneous state of the specified node on the CAN network.

can	handle to the CAN object
node	the node number of the CANOpen node.
nodeStatus	a pointer to where this function will put the node status.

Return Values	
MPIMessageOK	

See Also

meiCanNodeInfo | meiCanStatus

meiCanNodeInfo

Declaration

```
long meiCanNodeInfo(MEICan            can,
                    long              node,
                    MEICanNodeInfo*   nodeInfo);
```

Required Header: stdmei.h

Description

meiCanNodeInfo returns the node information for the specified node on the CAN network that was generated when the XMP/ZMP finished scanning the network.

can	handle to the CAN object
node	the filename of the CAN controller firmware (*.out file).
nodeInfo	a pointer to where this function will put the node information.

Return Values	
MPIMessageOK	

See Also

meiCanNodeStatus | meiCanStatus

meiCanNodeAnalogIn

Declaration

```
long meiCanNodeAnalogIn(MEICan        can,
                        long          node,
                        long          channel,
                        long          *state);
```

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

Description

meiCanNodeAnalogIn gets the current state of an analog input on the specified CAN node.

can	handle to the CAN object
node	the node number of the CANOpen node.
channel	the index of the analog input.
***state**	a pointer to where the current state of the input is written to by this function. See CAN Analog Values.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to get the state of analog input 3 on node 5.

```
long analog3;
meiCanNodeAnalogIn( can, 5, 3, &analog3 );
```

See Also

meiCanNodeAnalogOutSet | meiCanNodeAnalogOutGet | CAN Analog Values

meiCanNodeAnalogOutGet

Declaration

```
long meiCanNodeAnalogOutGet(MEICan        can,
                            long          node,
                            long          channel,
                            long          *state);
```

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

Description

meiCanNodeAnalogOutGet gets the current state of an analog output on the specified CAN node.

can	handle to the CAN object
node	the node number of the CANOpen node.
channel	the index of the analog output.
***state**	a pointer to where the current state of the output is written to by this function. See CAN Analog Values.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to get the state of analog output 3 on node 5.

```
long analog3;
meiCanNodeAnalogOutGet( can, 5, 3, &analog3 );
```

See Also

meiCanNodeAnalogIn | meiCanNodeAnalogOutSet | CAN Analog Values

meiCanNodeAnalogOutSet

Declaration

```
long meiCanNodeAnalogOutSet(MEICan        can,
                            long          node,
                            long          channel,
                            long          state,
                            long          wait);
```

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

Description

meiCanNodeAnalogOutSet changes the state of an analog output on the specified CAN node.

can	handle to the CAN object
node	the node number of the CANOpen node.
channel	the index of the analog output.
state	the new state of the analog output. See CAN Analog Values.
wait	a Boolean flag that indicates if the new output state is immediately applied or a *wait* is inserted so that any previously set output is transmitted over CAN first before applying the new output state.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to change the state of analog output 3 on node 5 to the maximum value 7FFFh.

```
meiCanNodeAnalogOutGet( can, 5, 3, 0x7FFF, 1 );
```

See Also

meiCanNodeAnalogIn | meiCanNodeAnalogOutGet | CAN Analog Values

meiCanNodeDigitalIn

Declaration

```
long meiCanNodeDigitalIn(MEICan         can,
                         long           node,
                         long           bitStart,
                         long           bitCount,
                         unsigned long *state);
```

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

Description

meiCanNodeDigitalIn gets the current state of one or multiple digital inputs on the specified CAN node.

can	handle to the CAN object
node	the node number of the CANOpen node.
bitSmart	the first input bit on the CAN node that will be returned by this function.
bitCount	the number of bits that will be returned by the function.
***state**	the address of the current state of the input(s) that is returned.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to get the state of controller input 1.

```
unsigned long input3;
meiCanDigitalIn( can, 5, 3, 1, &input3 );
```

See Also

meiCanNodeDigitalOutSet | meiCanNodeDigitalOutGet

meiCanNodeDigitalOutGet

Declaration

```
long meiCanNodeDigitalOutGet(MEICan        can,
                             long          node,
                             long          bitStart,
                             long          bitCount,
                             unsigned long *state);
```

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

Description

meiCanNodeDigitalOutGet gets the current state of one or multiple digital outputs on the specified CAN node.

can	handle to the CAN object
node	the node number of the CANOpen node.
bitSmart	the first output bit on the CAN node that will be returned by this function.
bitCount	the number of bits that will be returned by the function.
***state**	the address of the current state of the output(s) that is returned.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to get the state of digital output 3 on node 5.

```
unsigned long output3;
meiCanDigitalOutGet( can, 5, 3, 1, &output3 );
```

See Also

meiCanNodeDigitalOutSet | meiCanNodeDigitalIn

meiCanNodeDigitalOutSet

Declaration

```
long meiCanNodeDigitalOutSet(MEICan        can,
                             long          node,
                             long          bitStart,
                             long          bitCount,
                             unsigned long state,
                             MPI_BOOL      wait);
```

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

Description

meiCanNodeDigitalOutSet changes the state of one or multiple digital outputs on the specified CAN node.

can	handle to the CAN object
node	the node number of the CANOpen node.
bitSmart	the first output bit on the CAN node that will be returned by this function.
bitCount	the number of bits that will be set by the function.
state	the new state of the outputs on the CANOpen node.
wait	a Boolean flag that indicates if the new output state is immediately applied or a wait is inserted so that any previously set output is transmitted over CAN first before applying the new output state.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to set the state of digital output 3 on node 5.

```
meiCanDigitalOutSet( can, 5, 3, 1, 1, 1);
```

See Also

meiCanNodeDigitalOutGet | meiCanNodeDigitalIn

meiCanEventNotifyGet

Declaration

```
long meiCanEventNotifyGet(MEICan          can,
                          MPIEventMask    *eventMask,
                          void            *external);
```

Required Header: stdmei.h

Description

meiCanEventNotifyGet gets the current CAN event mask.

can	handle to the CAN object.
***eventMask**	a pointer to the MPI event mask that will be filled in by this function.
***external**	external points to an implementation specific structure. Since there is currently no implementation specific data, NULL should be used.

Return Values	
MPIMessageOK	

See Also

meiCanNotifySet

meiCanEventNotifySet

Declaration

```
long meiCanEventNotiySet(MEICan          can,
                         MPIEventMask    eventMask,
                         void            *external);
```

Required Header: stdmei.h

Description

meiCanEventNotifySet updates the current CAN event mask.

can	handle to the CAN object.
eventMask	a pointer to the new MPI event mask that will be filled in by this function.
***external**	external points to an implementation specific structure. Since there is currently no implementation specific data, NULL should be used.

Return Values	
MPIMessageOK	

See Also

meiCanEventNotifyGet

meiCanFirmwareDownload

Declaration

```
long meiCanFirmwareDownload(MEICan            can,
                            const  char*      filename,
                            MEICanCallback    callback);
```

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

Description

meiCanFirmwareDownload allows the user to upgrade the CAN controller's firmware.

This operation will take some time (between 10 and 30 seconds) to perform the download process. Therefore, the callback function is provided to allow the current status of the download operation to be reported to the calling application and to also allow the calling application to abort the download if required. The callback function passes the progress of the download process to the calling application. The calling applications normally returns a 0 unless it wants to abort the upgrade. If the upgrade is aborted, it returns a 1.

can	handle to the CAN object
filename	the filename of the CAN controller firmware (*.out file).
callback	a pointer to the call back function. (Pass an address of zero if you do not have a callback function.)

Return Values	
MPIMessageOK	

See Also

meiCanFirmwareErase | meiCanFirmwareUpload

meiCanFirmwareErase

Declaration

```
long meiCanFirmwareErase(MEICan   can);
```

Required Header: stdmei.h

Description

meiCanFirmwareErase allows the user to erase the CAN controllers firmware.

can	handle to the CAN object

Return Values	
MPIMessageOK	

See Also

meiCanFirmwareDownload | meiCanFirmwareUpload

meiCanFirmwareUpload

Declaration

```
long meiCanFirmwareUpload(MEICan            can,
                          const  char*      filename,
                          MEICanCallback    callback);
```

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

Description

meiCanFirmwareUpload allows the user to get a copy of the current CAN controller's firmware.

This operation will take some time (between 10 and 30 seconds) to perform the upload process. Therefore, the callback function is provided to allow the current status of the upload operation to be reported to the calling application and to also allow the calling application to abort the upgrade (if required). The callback function passes the progress of the upgrade process to the calling application. The calling applications normally returns 0 unless it wants to abort the upgrade. If the upgrade is aborted, it returns a 1.

can	handle to the CAN object
filename	the filename of the CAN controller firmware (*.out file).
callback	a pointer to the call back function. (Pass an address of zero if you do not have a callback function.)

Return Values	
MPIMessageOK	

See Also

meiCanFirmwareErase | meiCanFirmwareDownload

meiCanMemory

Declaration

```
long meiCanMemory(MEICan   can,
                  void**   memory);
```

Required Header: stdmei.h

Description

meiCanMemory returns a pointer to the base of the CAN processors DPR. This function is generally not used and is provided for implementing advanced features of the MPI.

can	handle to the CAN object
memory	a pointer to the base of the CAN processors DPR.

Return Values	
MPIMessageOK	

See Also

meiCanMemoryGet | meiCanMemorySet

meiCanMemoryGet

Declaration

```
long meiCanMemoryGet(MEICan        can,
                     void*         dst,
                     const void*   src,
                     long          count);
```

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

Description

meiCanMemoryGet copies the specified number of bytes from controller's memory to the application's memory. This function is generally not used and is provided for implementing advanced features of the MPI.

can	handle to the CAN object
dst	the base address of the destination
src	the base address of the source
count	the number of bytes to copy

Return Values	
MPIMessageOK	

See Also

meiCanMemory | meiCanMemorySet

meiCanMemorySet

Declaration

```
long meiCanMemorySet(MEICan        can,
                     void*         dst,
                     const void*   src,
                     long          count);
```

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

Description

meiCanMemorySet copies the specified number of bytes from the application's memory to the controller's memory. This function is generally not used and is provided for implementing advanced features of the MPI.

can	handle to the CAN object
dst	the base address of the destination
src	the base address of the source
count	the number of bytes to copy

Return Values	
MPIMessageOK	

See Also

meiCanMemory | meiCanMemoryGet

meiCanInit

Declaration

```
long meiCanInit(MEICan    can);
```

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

Description

meiCanInit will reset the CAN network and will not affect the rest of the controller or SynqNet.

can	handle to the CAN object

Return Values	
MPIMessageOK	

See Also

meiCanControl

Declaration

```
MPIControl meiCanControl(MEICan   can);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiCanControl returns a handle to the control object associated with the Can object.

can	a handle to a Can object.

Return Values	
MPIControl	a handle to a control object.
MPIHandleVOID	if the object could not be created

See Also

meiCanCreate | mpiControlCreate

meiCanNumber

Declaration

```
long  meiCanNumber(MEICan   can,
                   long     *number);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiCanNumber reads the index of a Can object and writes it into the contents of a long pointed to by ***number***. Each Can node associated with a controller is indexed by a number (0, 1, 2, etc.).

can	a handle to a Can object.
***number**	a pointer to the index of a Can node.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessageHANDLE_INVALID	

See Also

meiCanNodeInfo

MEICanBitRate

Definition

```
typedef enum {
    MEICanBitRate1000K = 0,
    MEICanBitRate800K,
    MEICanBitRate500K,
    MEICanBitRate250K,
    MEICanBitRate125K,
    MEICanBitRate50K,
    MEICanBitRate20K,
    MEICanBitRate10K
} MEICanBitRate;
```

Description

MEICanBitRate enumerates all the valid bit rates that the CANOpen interface can use. These are the recommended bit rates that the CANOpen standard defines.

For more information see the Bit Rate section.

See Also

MEICanBusState

Definition

```
typedef enum {
    MEICanBusStateOFF,
    MEICanBusStatePASSIVE,
    MEICanBusStateOPERATIONAL
} MEICanBusState;
```

Description

MEICanBusState enumerates the bus states that the controller's CAN interface can take.

To see how the CanBusState is displayed in Motion Console, click here.

See Also

CAN Bus State

MEICanCallback

Definition

```
typedef long (*MEICanCallback)(long percentage);
```

Description

MEICanCallback is the definition of a call back function used during the firmware download.

See Also

MEICanCommand

Definition

```
typedef struct MEICanCommand {
    MEICanCommandType   type;
    long                data[6];
} MEICanCommand;
```

Description

MEICanCommand holds the command request and response for an meiCanCommand.

type	The type of CAN command.
data	Data associated with the command.

See Also

meiCanCommand

MEICanCommandType

Definition

```
typedef enum {
    MEICanCommandTypeSDO_READ,
    MEICanCommandTypeSDO_WRITE,
    MEICanCommandTypeCLEAR_STATUS_BITS,
    MEICanCommandTypeBUS_START,
    MEICanCommandTypeBUS_STOP,
    MEICanCommandTypeNMT_ENTER_PRE_OPERATIONAL,
    MEICanCommandTypeNMT_START_REMOTE_NODE,
    MEICanCommandTypeNMT_STOP_REMOTE_NODE,
    MEICanCommandTypeNMT_RESET_NODE,
    MEICanCommandTypeNMT_RESET_COMMUNICATION,
} MEICanCommandType;
```

Description

MEICanCommandType enumerates the different type of commands that can be used with meiCanCommand.

MEICanCommandTypeSDO_READ

This command reads the remote nodes object dictionary using the SDO protocol.

Command data:
data[0] = Node
data[1] = Index
data[2] = SubIndex
data[3] = Length

Returned data:
data[0] = Error code
data[4] = Low Data word
data[5] = High Data word

MEICanCommandTypeSDO_WRITE

This command writes to a remote nodes object dictionary using the SDO protocol.

Command data:
data[0] = Node
data[1] = Index
data[2] = SubIndex
data[3] = Length
data[4] = Low Data word
data[5] = High Data word

Returned data:
data[0] = Error code

MEICanCommandTypeCLEAR_STATUS_BITS

Clear selected MEICanStatusBits.

Command data:
data[0], Bit map of MEICanStatusBits to clear.

Returned data:
data[0] = Error code

MEICanCommandTypeBUS_START

This puts the CAN bus into operational state if it is Bus off.

Command data:
None

Returned data:
data[0] = Error code

MEICanCommandTypeBUS_STOP

This puts the CAN bus into operational state if it is Bus off.

Command data:
None

Returned data:
data[0] = Error code

MEICanCommandTypeNMT_ENTER_PRE_OPERATIONAL

This issues the CANOpen NMT command "Enter Pre-Operational" to a node.

Command data:
data[0] = Node number, (0 broadcasts to all nodes)

Returned data:
data[0] = Error code

MEICanCommandTypeNMT_START_REMOTE_NODE

This issues the CANOpen NMT command "Start Remote Node" to a node.

Command data:
data[0] = Node number, (0 broadcasts to all nodes)

Returned data:
data[0] = Error code

MEICanCommandTypeNMT_STOP_REMOTE_NODE

This issues the CANOpen NMT command "Stop Remote Node" to a node.

Command data:
data[0] = Node number, (0 broadcasts to all nodes)

Returned data:
data[0] = Error code

MEICanCommandTypeNMT_RESET_NODE

This issues the CANOpen NMT command "Reset Node" to a node.

Command data:
data[0] = Node number, (0 broadcasts to all nodes)

Returned data:
data[0] = Error code

MEICanCommandTypeNMT_RESET_COMMUNICATION

This issues the CANOpen NMT command "Reset Communication" to a node.

Command data:
data[0] = Node number, (0 broadcasts to all nodes)

Returned data:
data[0] = Error code

See Also

meiCanCommand

MEICanConfig

Definition

```
typedef struct MEICanConfig {
    MEICanBitRate           bitRate;
    unsigned        long    cyclicPeriod;
    unsigned        long    healthPeriod;
    unsigned        long    nodeNumber;
    unsigned        long    inhibitTime;
} MEICanConfig;
```

Description

MEICanConfig holds the configuration of the CAN object. The default state for this structure is held in the controller's flash. Use the meiCanConfigGet/Set and meiCanNodeConfigGet/Set to interrogate and change to what the CAN system is currently using or the default.

bitRate	The bit rate the CAN bus uses. See also CAN Bit Rate.
cyclicPeriod	The period (milliseconds) between sending consecutive SYNC messages. A value of zero will disable the SYNC messages from being produced. See also CAN Transmission Types.
healthPeriod	The period (milliseconds) used for checking the health of nodes. A value of zero will disable the health checking protocol. For nodes that use the node guarding protocol, this is the node guarding period. For nodes that use the heartbeating protocol, this is the heartbeat consumer time (the heartbeat producers are half this period). See also CAN Node Health.
nodeNumber	The node number of the controller on the CAN network. CANOpen requires that the master node has a valid node number to implement the heartbeat protocol. See also CAN Node Numbers.
inhibitTime	The minimum time (in milliseconds) that a node on the network will remain silent before transmitting their next event message. See also CAN Transmission Types.

See Also

meiCanConfigGet | meiCanConfigSet | meiCanNodeConfigGet | meiCanNodeConfigSet

MEICanHealthType

Definition

```
typedef enum {
    MEICanHealthTypeNODE_GUARDING,
    MEICanHealthTypeHEART_BEATING
} MEICanHealthType;
```

Description

MEICanHealthType is used to report the health protocol that the XMP is using with each node.

See Also

MEICanMessage

Definition

```
typedef enum {
    MEICanMessageFIRMWARE_INVALID,
    MEICanMessageFIRMWARE_VERSION,
    MEICanMessageNOT_INITALIZED,
    MEICanMessageCAN_INVALID,
    MEICanMessageIO_NOT_SUPPORTED,
    MEICanMessageFILE_FORMAT_ERROR,
    MEICanMessageUSER_ABORT,
    MEICanMessageCOMMAND_PROTOCOL,
    MEICanMessageINTERFACE_NOT_FOUND,
    MEICanMessageNODE_DEAD,
    MEICanMessageSDO_TIMEOUT,
    MEICanMessageSDO_ABORT,
    MEICanMessageSDO_PROTOCOL,
    MEICanMessageTX_OVERFLOW,
    MEICanMessageRTR_TX_OVERFLOW,
    MEICanMessageRX_BUFFER_EMPTY,
    MEICanMessageBUS_OFF,
    MEICanMessageSIGNATURE_INVALID,
} MEICanMessage;
```

Change History: Modified in the 03.02.00

Description

MEICanMessage is an enumeration of Can error messages that can be returned by the MPI library.

MEICanMessageFIRMWARE_INVALID

The CAN firmware is not valid. This message code is returned by meiCanCreate(...) if the CAN hardware bootloader detects no firmware has been loaded or the firmware signature is not recognized. To correct this problem, download valid firmware with meiCanFirmwareDownload(...).

MEICanMessageFIRMWARE_VERSION

The CAN firmware version does not match the software version. This message code is returned by meiCanCreate(...), meiCanFirmwareDownload(...), or meiCanFirmwareUpload(...) if the CAN firmware version is not compatible with the MPI library. To correct this problem, download the proper firmware version with meiCanFirmwareDownload(...).

MEICanMessageNOT_INITIALIZED

The CAN firmware did not initialize. This message code is returned by meiCanCreate(...) if the controller did not copy the configuration structure from flash to memory after power-on or controller reset. To correct this problem, verify the controller firmware is correct and the controller hardware is operating properly.

MEICanMessageCAN_INVALID

The can network number is out of range. This message code is returned by meiCanCreate(...) if the network number is less than zero or greater than or equal to MEICanNetworkMAX.

MEICanMessageIO_NOT_SUPPORTED

The CAN node does not support the specified I/O. This message code is returned by CAN methods that read/write to a digital or analog input/output that is out of range. To prevent this problem, specify a supported I/O bit.

MEICanMessageFILE_FORMAT_ERROR

The CAN firmware file format has an error. This message code is returned by meiCanFirmwareDownload(...) if the specified file has an error in its internal headers. This indicates a corrupted file. To correct this problem, use the original CAN firmware file or reinstall the software distribution.

MEICanMessageUSER_ABORT

The CAN firmware loading was aborted. This message code is returned by meiCanFirmwareDownload(...) or meiCanFirmwareUpload(...) when the firmware loading is aborted by the user via the callback function. This message code is returned for application notification. It is not an error.

MEICanMessageCOMMAND_PROTOCOL

The CAN command failed due to a protocol error. This message code is returned by CAN methods that do not get a valid response from a CAN node. To correct this problem, check your CAN nodes for proper operation.

MPICanMessageINTERFACE_NOT_FOUND

The CAN interface is not available. This message code is returned by meiCanCreate(...) if the specified controller does not support a CAN network interface. To correct this problem, use a controller that has a CAN interface.

MEICanMessageNODE_DEAD

The CAN node does not respond. This message code is returned by CAN methods that read/write from a CAN node and the node fails the health check. This message code indicates a node hardware or network connection problem. To correct this problem, verify the node operation and network connections.

MEICanMessageSDO_TIMEOUT

The CAN command failed due to a timeout. This message code is returned by CAN methods that do not get a response from a CAN node within the timeout period. To correct this problem, check your CAN nodes for proper operation.

MEICanMessageSDO_ABORT

The CAN command failed due to a user abort. This message code is returned by CAN methods when an SDO transaction is aborted.

MEICanMessageSDO_PROTOCOL

The CAN command failed due to an SDO protocol error. This message code is returned by CAN methods when an SDO transaction fails because the node did not conform to the CANOpen protocol.

MEICanMessageTX_OVERFLOW

The controller's transmit buffer overflowed. This message code is returned by CAN methods that failed to transmit a message due to an internal memory buffer overflow.

MEICanMessageRTR_TX_OVERFLOW

The controller's transmit buffer overflowed. This message code is returned by CAN methods that failed to transmit a message due to an internal memory buffer overflow.

MEICanMessageRX_BUFFER_EMPTY

The controller's receive buffer is empty. This message code is returned by CAN methods that expected to get a response from a CAN node, but the controller's receive buffer was empty.

MEICanMessageBUS_OFF

The CAN network bus is in the off state. This message code is returned by CAN methods that are not able to use the CAN network because the bus is off. To correct this problem, verify the node operation and network connections.

MEICanMessageSIGNATURE_INVALID

When initialising the CAN system, some tests are performed to make sure that the CAN processor is returning a valid signature value. If an unexpected signature is returned, this error message is returned. A probable cause for this error is that the bootloader is invalid. To correct this problem, you will need to return the controller to MEI to fix the bootloader.

See Also

MEICanNodeConfig

Definition

```
typedef struct MEICanNodeConfig {
    MEICanTransmissionType digitalOutTransmissionType;
    MEICanTransmissionType analogOutTransmissionType;
    MEICanTransmissionType digitalInTransmissionType;
    MEICanTransmissionType analogInTransmissionType;
} MEICanNodeConfig;
```

Description

MEICanNodeConfig is the configuration of each node on the CAN bus. You can select which type of communication (event or cyclic) is to be used for the different types of IO data that a node supports.

For more information, see the CAN Transmission Types section.

See Also

MEICanTransmissionType | meiCanNodeConfigGet | meiCanNodeConfigSet

MEICanNodeInfo

Definition

```
typedef struct MEICanNodeInfo {
    MEICanNodeType              type;
    unsigned long               digitalInputCount;
    unsigned long               digitalOutputCount;
    unsigned long               analogInputCount;
    unsigned long               analogOutputCount;
    MEICanHealthType            healthType;
    MEICanNodeInfoVendor        vendorID;
    MEICanNodeInfoProductCode   productCode;
    unsigned long               versionNumber;
    unsigned long               serialNumber;
} MEICanNodeInfo;
```

Change History: Modified in the 03.03.00

Description

MEICanNodeInfo describes how many of the different types of I/O are on this node.

type	An enumeration indicating the type of node found at startup, or MEICanNodeTypeNONE if no node was found.
digitalInputCount	The number of digital inputs supported by this node. The CANOpen protocol only allows the number of digital inputs to be interrogated in multiples of eight, i.e. if a node has two digital inputs then digitalInputCount will return eight. MEI CANOpen SLICE nodes support an extension to the CANOpen protocol that allows the exact number of digital inputs to be returned in this field.
digitalOutputCount	The number of digital outputs supported by this node. The CANOpen protocol only allows the number of digital outputs to be interrogated in multiples of eight, i.e. if a node has two digital outputs then digitalOutputCount will return eight. MEI CANOpen SLICE nodes support an extension to the CANOpen protocol that allows the exact number of digital outputs to be returned in this field.
analogInputCount	The number of analog inputs supported by this node.
analogOutputCount	The number of analog outputs supported by this node.
healthType	The type of health checking protocol being used with this node. See also CAN Node Health.

vendorId	This is a number read from the node. Vendor ID numbers are unique numbers allocated to each manufacturer of CANOpen nodes. Not all CANOpen nodes support this feature, in which case, these nodes will return zero for this field. MEI CANOpen nodes always return 0x014F. See also MEICanNodeInfoVendor.
productCode	This is a number read from the node. The product code is made up of numbers allocated by each manufacturer to uniquely identify their different types of nodes. Not all CANOpen nodes support this feature, in which case, these nodes will return zero for this field. MEI CANOpen SLICE nodes always return 0x0204. See also MEICanNodeInfoProductCode.
versionNumber	This is a number read from the node. The version number identify the version of code running on this CANOpen node. Not all CANOpen nodes support this feature, in which case, these nodes will return zero for this field. MEI CANOpen nodes do support this field.
serialNumber	This is a number read from the node. The serial number uniquely identifies each CANOpen node. Not all CANOpen nodes support this feature, in which case, these nodes will return zero for this field. MEI CANOpen SLICE nodes do support this field and the number is also on the side label of the Network adapter.

See Also

MEICanNodeInfoProductCode

Definition

```
typedef enum {
    MEICanNodeInfoProductCodeUNKNOWN = 0,
    MEICanNodeInfoProductCodeMEI_SLICE_IO = 0x0204
} MEICanNodeInfoProductCode;
```

Change History: Modified in the 03.03.00

Description

MEICanNodeInfoProductCode defines the product codes for the MEI manufactured CANOpen nodes. If the node is not manufactured by MEI then the product code may be any non-zero number. A zero product code (UNKNOWN) indicates that the manufacturer does not support the CANOpen method to read this from the node.

See Also

MEICanNodeInfo | Slice-I/O Hardware

MEICanNodeInfoVendor

Definition

```
typedef enum {
   MEICanNodeInfoVendorUNKNOWN = 0,
   MEICanNodeInfoVendorMEI = 0x014F
} MEICanNodeInfoVendor;
```

Change History: Added in the 03.03.00

Description

MEICanNodeInfoVendor defines some vendor Ids for CANOpen nodes. A zero vendor ID (UNKNOWN) indicates that the manufacturer does not support the CANOpen method to read this from the node.

See Also

MEICanNodeInfo

MEICanNodeStatus

Definition

```
typedef struct MEICanNodeStatus {
    unsigned long    live;
    MEICanNMTState   nmtState;
} MEICanNodeStatus;
```

Description

MEICanNodeStatus holds the current status of a node.

live	Set if the node is alive, clear if the node is dead.
nmtState	The current NMT state that the node is reporting.

See Also

CAN Node Health

MEICanNodeType

Definition

```
typedef enum {
    MEICanNodeTypeNONE = 0,
    MEICanNodeTypeIO   = 401
} MEICanNodeType;
```

Description

MEICanNodeType enumerates the different types of nodes that the XMP has detected. MEICanNodeTypeNONE is returned if no node is found or an unsupported node type is detected.

See Also

CAN Node Health

MEICanNMTState

Definition

```
typedef enum {
    MEICanNMTStateBOOT_UP,
    MEICanNMTStateSTOPPED,
    MEICanNMTStateOPERATIONAL,
    MEICanNMTStatePRE_OPERATIONAL,
    MEICanNMTStateUNKNOWN,
} MEICanNMTSTATE;
```

Description

MEICanNMTState enumerates the NMT (network management) states of a node on a CANOpen network. The XMP's CAN controller will automatically put all nodes into the Operational state during the initialization of the network.

See Also

MEICanStatus

Definition

```
typedef struct MEICanStatus {
    MEICanBusState    busState;
    long              transmitErrorCounter;
    long              receiveErrorCounter;
    long              messageRate;
    long              tick;
    long              softwareReceiveOverflow;
    long              hardwareReceiveOverflow;
} MEICanStatus;
```

Description

MEICanStatus holds the current status of the XMP's or ZMP's CAN object.

busState	The current bus state of the XMP's or ZMP's CAN interface.
transmitErrorCounter	The current value of the transmit error counter.
receiveErrorCounter	The current state of the receive error counter.
messageRate	The number of messages received and transmitted per second.
tick	This is incremented every 1ms by the CAN firmware.
softwareReceiveOverflow	This bit will be set if software receive buffer has overflowed. This bit can be cleared by using the CLEAR_STATUS_BITS command.
hardwareReceiveOverflow	This bit will be set if the CAN interface hardware has detected an overflow. This bit can be cleared by using the CLEAR_STATUS_BITS command.

See Also

MEICanTransmissionType

Definition

```
typedef enum {
    MEICanTransmissionTypeCYCLIC = 0,
    MEICanTransmissionTypeEVENT  = 1,
} MEICanTransmissionType;
```

Description

MEICanTransmissionType enumerates the transmission types a node can use.

For more information, see the CAN Transmission Types section.

See Also

MEICanNodeConfig | meiCanNodeConfigGet | meiCanNodeConfigSet

MEICanVersion

Definition

```
typedef struct MEICanVersion {
    long    bootloaderVersion;
    long    firmwareVersion;
    char    firmwareRevision;
    long    firmwareSubRevision;
} MEICanVersion;
```

Description

MEICanVersion holds the version information about the XMP's or ZMP's CAN object.

bootloaderVersion	The version number of the CAN bootloader.
firmwareVersion	The CAN firmware version.
firmwareRevsion	The CAN firmware revision.
firmwareSubRevision	The CAN firmware subrevision.

See Also

MEICanNetworkMAX

Definition

```
#define MEICanNetworkMAX  (1)
```

Change History: Added in the 03.02.00

Description

MEICanNetworkMAX defines the maximum number of Can networks supported by a controller.

See Also

meiCanCreate

CAN Bit Rate

The CANOpen standard defines a set of bit rates that can be supported. Any CANOpen node must support at least one of these bit rates. All the nodes on the CAN network must be operating at the same bit rate. Any of these standard bit rates can be used with the XMP.

Due to the electrical characteristics of a CAN network, the maximum length of a CAN network (and the corresponding drop lengths) is dependent upon the bit rate that is chosen. See the table below.

It is recommended that opto-isolated nodes are used on networks with bus lengths longer than 200m.

CANOpen Bit Rates

Bit Rate	Max Bus Length (m)	Max Drop Length (m)	Max Cumulative Drop Length (m)
1M	25*	2	10
800k	50*	3	15
500k	100	6	30
250k	250	12	60
125k	500	24	120
50k	1000	60	300
20k	2500	150	750
10k	5000	300	1500

* No opto-isolation

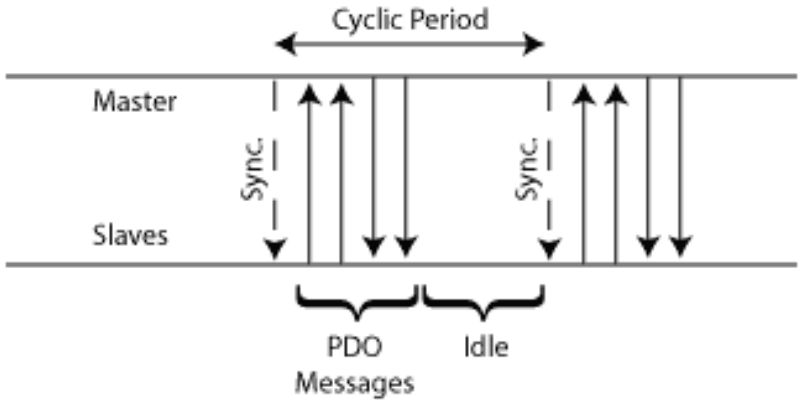
CAN Transmission Types

Introduction

The XMP CANOpen interface uses four messages (serial packets of data on the CAN bus) to pass I/O data between the XMP and an I/O node. Each message contains either the digital input, digital output, analog input, or analog output data. The XMP supports two standard communication methods to transmit I/O data between the XMP and each of the I/O nodes-**cyclic transmission** and **event transmission**. For most applications, cyclic messaging (the default) will be sufficient, but the transmission type fields within the MEICanNodeConfig structure allow the user to select an alternative transmission type for each of the I/O messages going to and from a node.

Cyclic Transmission

The Cyclic Transmission type, transfers I/O data messages between the XMP and the nodes using a cyclic protocol. The trigger for each cycle is a synchronization message that is transmitted at a regular rate by the XMP. When a node receives the synchronization message, it latches and transmits the current state of its inputs. Immediately after receiving the synchronization message, the master also transmits command messages to all the nodes with their new output states, which will get applied on the next synchronization message. An idle period is also needed to allow time for any non-cyclic messages to be transmitted.



The advantage of this scheme is that it generates a predicable loading of data on the bus. The latency on transmitted data is predictable, but the latency is not the absolute minimum that can be achieved.

Cyclic Period

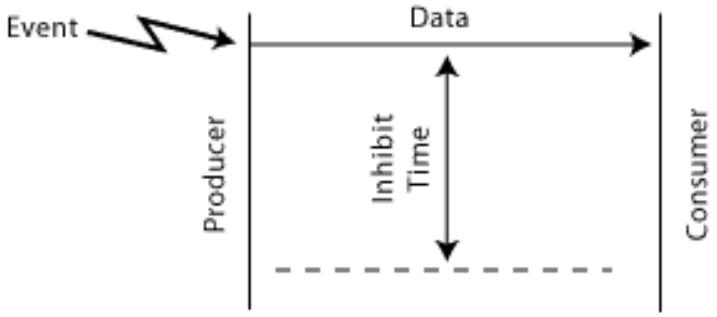
The cyclicPeriod field within the MEICanConfig structure allows the user to specify the period (in milliseconds) that the XMP will use between the successive transmission of synchronization messages. The minimum cyclic period that can be used is dependent upon the chosen bit rate and the number of nodes. Assuming that all the nodes have inputs and outputs that are analog and digital, the minimum cyclic period that can be used is given in the following table.

CANOpen Cyclic Period

Bit Rate	< 5 Nodes	< 10 Nodes	< 50 Nodes	< 128 Nodes
1M	3	5	30	60
800k	3	6	30	80
500k	5	10	50	200
250k	10	18	89	300
125k	19	36	200	500
50k	46	90	500	2000
20k	200	300	2000	3000
10k	300	500	3000	6000

Event Transmission

The Event Transmission type, only transmits I/O data messages when an "event" occurs on the source node (either the XMP or the I/O node) to change the I/O data. The event that forces the transmission is either a new state of an input that is detected on an I/O node or a new output state that is commanded on the XMP.



The advantage of this type of messaging is that short reaction times are attainable, but this is accomplished at the expense of variable network traffic, and the possibility of saturating the network. In many cases, the reaction time is not significant in relation to other time delays in the system (ex: the user's application or delays in task switching).

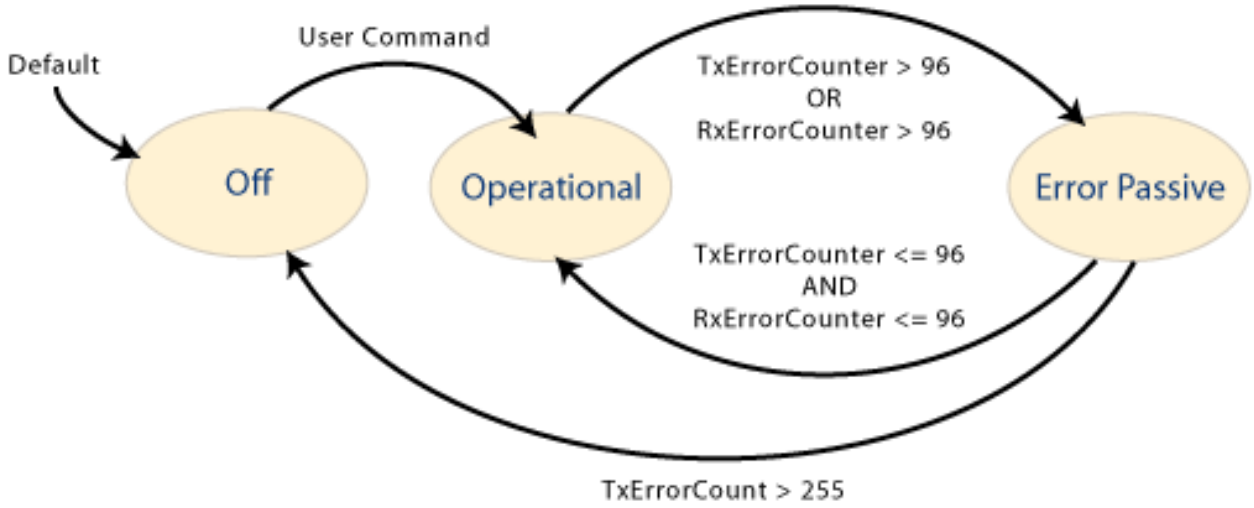
Inhibit Time

If the source node's events occur at a very fast rate, the number of messages generated can swamp the network and consequently block out other messages. To prevent an excess of messages, nodes can optionally support inhibit times for their transmit PDOs. This value defines the minimum time between two successive PDO messages.

The inhibitTime field within the MEICanConfig structure allows the user to specify the period (in milliseconds) that all nodes on the network will use. A reasonable inhibit time is half a cyclic period.

CAN Bus State

All CAN hardware maintains two error counters that are increased when transmit or receive errors are detected, and decreased when successful transmissions or receptions are achieved. In an error free operational system, these counters should be zero. The magnitude of these counters control the following state machine:

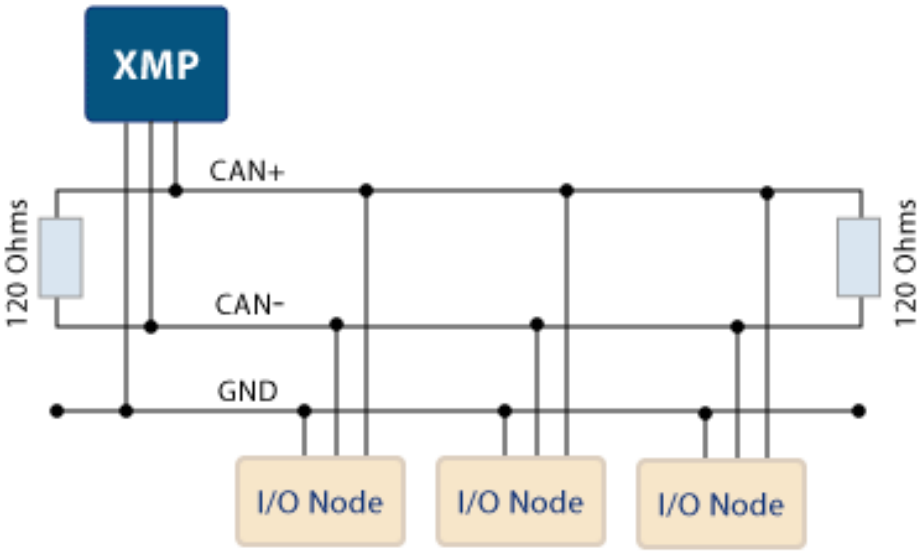


When a node is in the **Operational** state it will participate fully with all communications over the network, as the errors increase the CAN hardware will become **Passive** (detecting errors but not generating error messages), before turning **Off** and isolating the node from the network once the TxErrorCount exceeds 255 error messages. This feature allows nodes that are either malfunctioning or not configured correctly to be isolated for the network, thereby allowing the remaining nodes to successfully communicate.

CAN Hardware

CANOpen is a serial network that uses a bus topology. The CANOpen bus always contains two signal wires, CAN+ and CAN-, which carry the differential serial data and a ground (GND). It is also common for most CANOpen nodes to provide a shield connection.

Similar to most industrial buses, the signal wires need to be terminated. CANOpen requires a 120ohm resistor at both ends of the main bus. If these resistors are not fitted, the network will not function properly. Some node suppliers build the terminating resistor into the node and provide a jumper or switch to enable it. You will need to check your nodes' datasheets for the inclusion of a terminating resistor. The XMP does not have any terminating resistors.



For pinout information, go to the XMP's CAN D-9 connector page.

A CANOpen node either has an opto-isolated or non-isolated interface. The use of optoisolation is primarily provided as an EMC countermeasure and is used to cope with potential differences in the ground. These effects are more pronounced for large machines and cable lengths. Therefore, the use of opto-couplers is recommended for bus lengths greater that 200m. The disadvantage of opto-couplers is that they reduce the maximum permissible bus length for a given bit rate.

The XMP CAN interface is available with or without opto-isolation. This option needs to be specified at the time your XMP is ordered.

Most types of nodes require a separate power supply to drive the local logic and the I/O interfaces. For nodes that use opto-isolated interfaces, a separate supply of +7 to 24V needs to be provided to power the interface circuitry. The user must also supply an external 24V to the XMP (CAN_V+) if the opto-isolated interface option is being used.

Each node on the network must have a unique node number, in the range of 1 to 127. The node number is commonly set with a bank of DIP switches on each node. If two nodes are given the same node number, network errors are generated and unpredictable problems will be encountered. The node number of the XMP can be changed from the factory default of 1 using the meiCanConfigSet

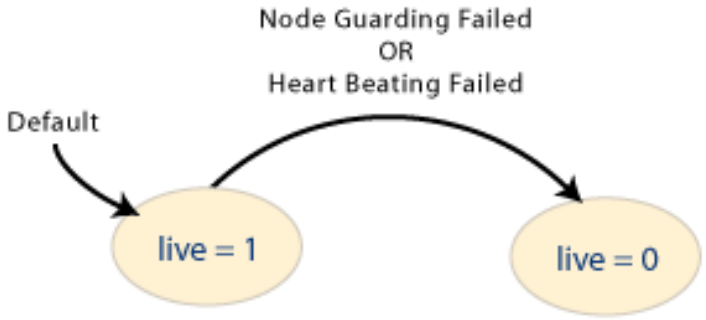
function.

In order for all nodes to communicate they must all use the same bit rate. Normally the bit rate that a node uses is set by DIP switches. If all of the nodes on a CANOpen network do not use the same bit rate then the whole network or some of the nodes on the network will not work properly. The bit rate of the XMP is set via software meiCanConfigSet. See also CAN Bit Rate.

CAN Node Health

All networks including CAN are vulnerable to faults such as breaks in the bus wiring or loss of power by some of the nodes. CANopen defines two methods for the master node (the XMP in our case) to periodically check the presence of nodes on the network-node guarding and heart beating.

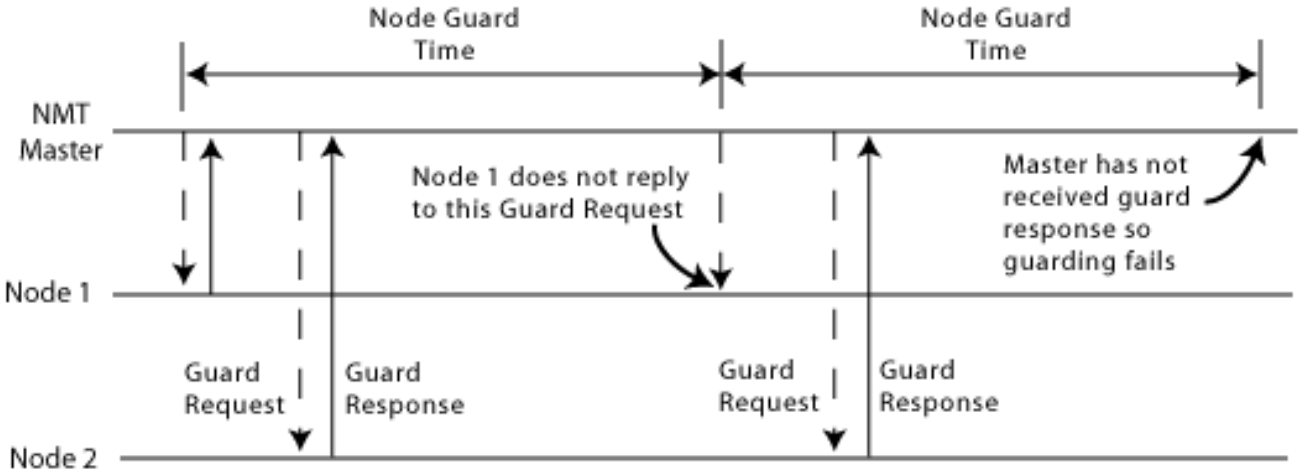
Using these services the XMP can monitor the health of the communications to each of the nodes. The current health of each node is reported in the live field of the MEICANNodeStatus structure.



It is mandatory for a node to either support the node guarding or heart beating protocols, or to support both. The heartbeat protocol has recently been introduced to CANOpen (in June 1999), and will probably NOT be supported on many nodes, but its adoption is recommended for all new nodes. The XMP's implementation will operate with either protocol and will automatically detect the protocol that each node supports and then use the most appropriate protocol for the CAN network. The healthType field of the MEICanNodeInfo structure reports the health checking protocol being used with each node.

Node Guarding protocol

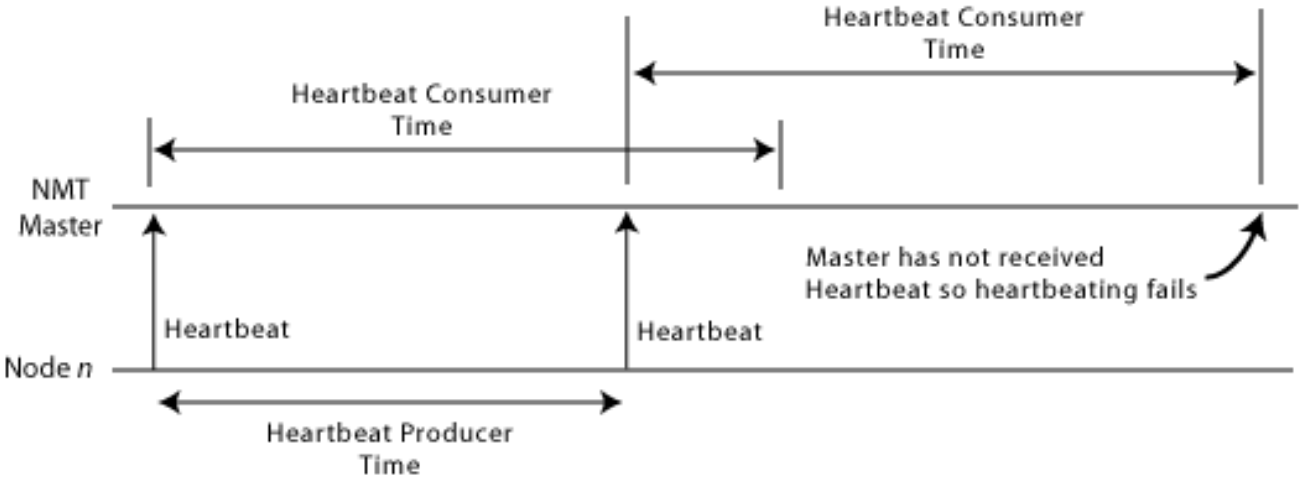
The Node Guarding protocol has the master sending an RTR message to all nodes on the network and checks to see whether a response is received from each of the nodes.



Heart Beating protocol

In the Heart Beating protocol, each node periodically broadcasts a heartbeat message. The period between transmitting the heartbeat messages is half the health period. If the XMP does not receive a message within a specific time window, it generates a heartbeat error for that node.

The advantage of the Heart Beating protocol over the Node Guarding protocol is that the number of messages is reduced in half, thereby freeing up bandwidth for other messages.



Health Period

The healthPeriod field of the MEICanConfig structure allows the user to specify the Node Guard and Heartbeat times for the health protocols according to the following table. The same period is used for all nodes.

Node Health Times

Protocol Times	Value
Node Guard Time	healthPeriod
Heartbeat Producer Time	healthPeriod / 2
Heatbeat Consumer Time	healthPeriod

For most applications it is recommended that the healthPeriod should be set to ten times the cyclic period.

CAN Emergency Messages

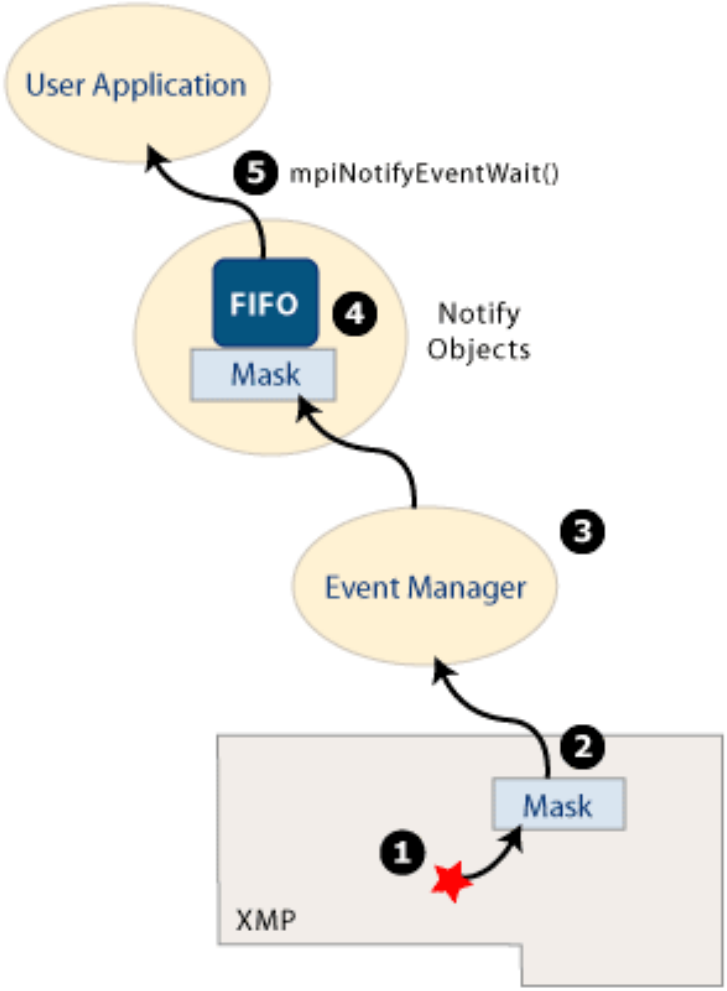
Every type of CANOpen node can transmit an emergency message. These messages are designed to report errors and warnings, as well as fatal problems on a node. The contents of these emergency messages are very dependent upon the node manufacturer and node type. To interpret this data, you will need to refer to the node manufacture's data. If an emergency message is generated by a node, the event handling scheme described in the events section below allows the user's application to receive the emergency message data.

CAN Handling Events

The CAN interface on the XMP generates many different types of asynchronous events such as:

- a change in the XMP's bus state
- a change in a node's health
- a change in the state of an input node's analog or digital inputs
- an emergency message is transmitted by a node
- a boot message is transmitted by a node
- a lost message is detected by the XMP CAN firmware

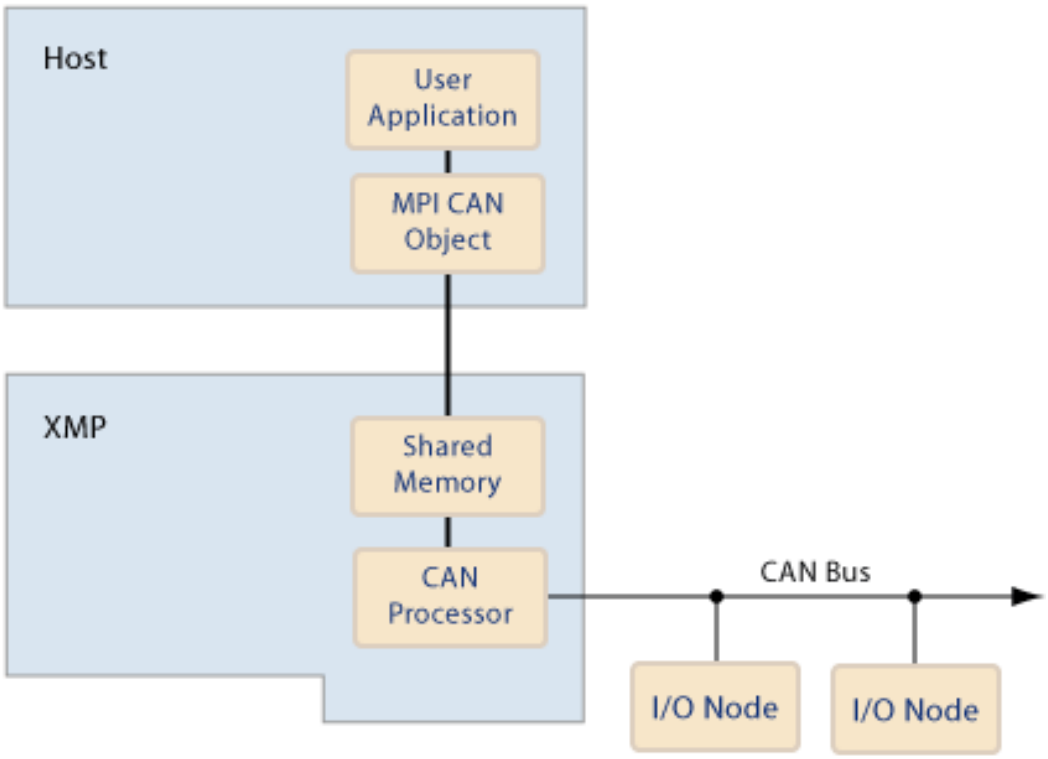
The events above have been appended to the standard MPI event handling scheme in order to provide the user the ability to respond to these events. The diagram below shows an overview of how events are relayed to the user's application.



1. The CANOpen firmware detects one of the CAN events.
2. There is a mask within the XMP firmware that allows only a specified set of events to reach the host. This mask is interrogated and modified with the meiCanEventNotifyGet and meiCanEventNotifySet functions.
3. Like all other events in the MPI, the user must install an Event Manager on the host. You will find the serviceCreate and serviceDelete functions from apputils convenient for installing an Event Manager.
4. For each thread that needs to know about CAN events, the user will need to create a notify object, specifying a mask for the required events.
5. The user's application can use the mpiNotifyEventWait function to either poll or wait for a CAN event to be generated. A valid event returned from mpiNotifyEventWait may also contain extra fields of information relevant to the event produced. (ex: the new bus state or node number).

CAN Hardware on the XMP

In the example below, the XMP uses a dedicated CAN processor to handle the network. This ensures that the motion will not be affected by the CAN network. The XMP operates as a master node on the network with all the I/O nodes being slaves. This arrangement implies that there may only be one XMP on any CAN Network.



The XMP operates as a master node on the network with all the IO nodes being slaves. This arrangement implies that there may only be one XMP on any CAN Network.

Capture Objects

Introduction

A **Capture** object manages a single position capture logic block. It represents the physical hardware capture logic and data. When configured and armed, the capture logic block can latch a motor's position based on one or more source input triggers.

The Capture object's number, motor input trigger sources, edge, type, feedback source, and capture index are all configurable. There are two capture types: Position and Time based. For the Position type, the position counters are latched in the FPGA and are read directly by the controller. This methodology works well for incremental quadrature encoders. For the Time type, the FPGA latches the clock and the controller reads the clock value and position value for that sample period. The controller interpolates the position value from the previous sample's position, the present sample's position, and the clock data. This methodology works very well for cyclic feedback data that is digitally transmitted from the drive to the FPGA. Many drives have a proprietary serial encoder that decodes the encoder position and sends the position information to the FPGA once per sample. In these cases, time-based capture is more accurate than position-based capture.

For the **Position** type, the motor number for the input sources and the feedback motor number must be the same.

For the **Time** type, the motor number and feedback motor number can be different. This makes it possible to use inputs from one node to capture positions on another node.

When using captures, the controller must have enough enabled captures to process the specified capture number. The controller will process the enabled captures (captureCount) every sample period. Since each capture object is configurable, use the minimum number of captures possible for best controller performance. For example, if you want to use 2 captures for motor 0 and motor 3, set the capture count to 2 and use capture number 0 and 1.

NOTE: Time-based capture will only work correctly if the speed of an axis is less than 344 million counts per second.

For an overview of the Capture Engine, see diagram below.

See Also: Overview of Capture

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiCapture**Create**	Create Capture object
mpiCapture**Delete**	Delete Capture object
mpiCapture**Validate**	Validate Capture object

Configuration and Information Methods

mpiCapture**ConfigGet**	Get Capture configuration
mpiCapture**ConfigSet**	Set Capture configuration
mpiCapture**Status**	Get status of Capture
mpiCapture**ConfigReset**	

Action Methods

mpiCapture**Arm**	Arm capture object

Memory Methods

mpiCapture**Memory**	Set address to Capture memory
mpiCapture**MemoryGet**	Copy Capture memory to application memory
mpiCapture**MemorySet**	Copy application memory to Capture memory

Relational Methods

mpiCapture**Control**	
mpiCapture**Number**	Get index of Capture (for Control list)

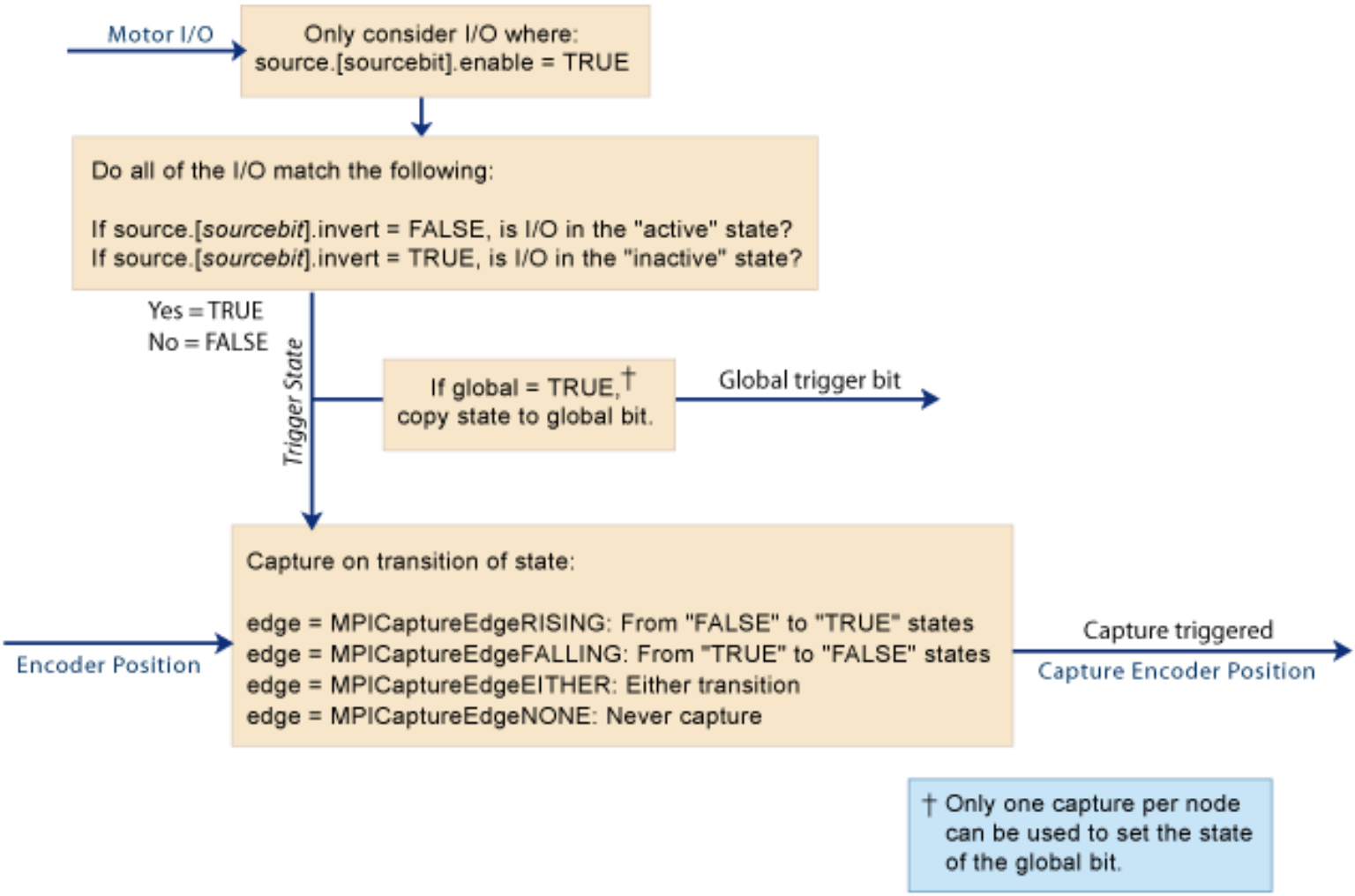
Data Types

MPICapture**Config**
MPICapture**Edge**
MPICapture**Message** / MEICapture**Message**
MPICapture**Source**
MPICapture**State**
MPICapture**Status**
MPICapture**Trigger**
MPICapture**TriggerGlobal**
MPICapture**Type**

Constants

MPICapture**NOT_MAPPED**

Capture Engine Diagram



mpiCaptureCreate

Declaration

```
MPICapture mpiCaptureCreate(MPIControl     control,
                            long           number);
```

Required Header: stdmpi.h

Description

mpiCaptureCreate creates a Capture object. The Capture object is identified by its association with a motor object, the motor's encoder and the encoder's capture number. The maximum number of enabled captures is 32.

CaptureCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	An index to the encoder's capture block.

Return Values	
handle	to a Capture object
MPIHandleVOID	if the object could not be created

See Also

mpiCaptureNumber

mpiCaptureDelete

Declaration

```
long mpiCaptureDelete(MPICapture  capture)
```

Required Header: stdmpi.h

Description

mpiCaptureDelete deletes a Capture object and invalidates its handle (***capture***).

CaptureDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiCaptureCreate | mpiCaptureValidate

mpiCaptureValidate

Declaration

```
long mpiCaptureValidate(MPICapture  capture)
```

Required Header: stdmpi.h

Description

mpiCaptureValidate validates the Capture object and its handle. CaptureValidate should be called immediately after an object is created.

capture	a handle to a Capture object

Return Values	
MPIMessageOK	

See Also

mpiCaptureCreate | mpiCaptureDelete

mpiCaptureConfigGet

Declaration

```
long mpiCaptureConfigGet(MPICapture       capture,
                         MPICaptureConfig *config,
                         void             *external)
```

Required Header: stdmpi.h

Description

mpiCaptureConfigGet gets a Capture object's (***capture***) configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

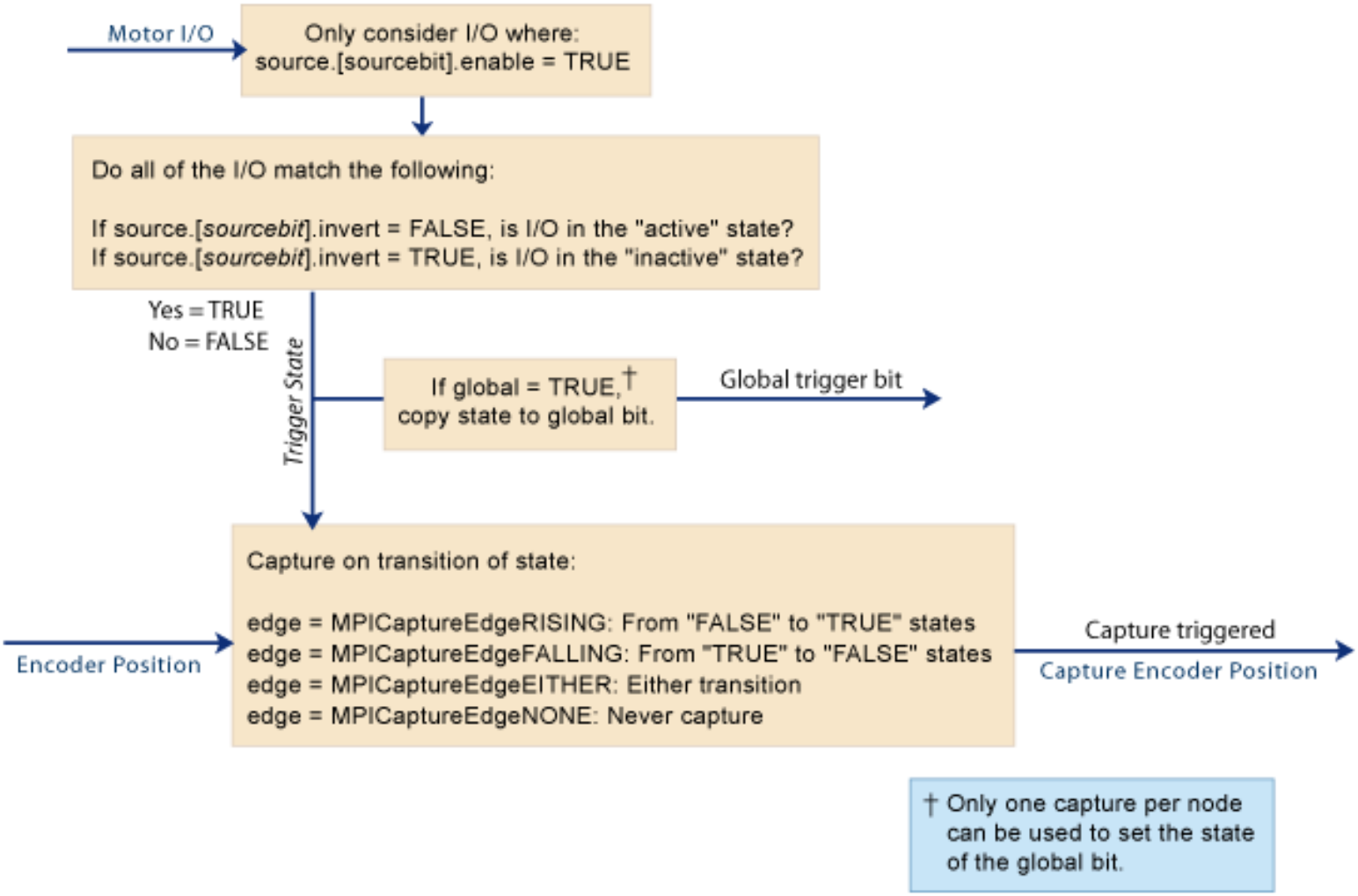
If a capture has been previously configured (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet (...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

Remarks

***external** should be NULL.

Return Values	
MPIMessageOK	

Capture Engine Diagram



See Also

mpiCaptureConfigSet | mpiCaptureConfigReset

mpiCaptureConfigSet

Declaration

```
long mpiCaptureConfigSet(MPICapture       capture,
                         MPICaptureConfig *config,
                         void             *external)
```

Required Header: stdmpi.h

Description

mpiCaptureConfigSet sets a Capture object's (***capture***) configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

If a capture has been previously configured (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet (...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

NOTE:
Don't reconfigure the source (trigger) capture resource with different settings via different MPICapture objects. Time-based capture allows you to capture multiple encoder positions using the same trigger. Currently, each motor only has one trigger resource— i.e. one trigger whose trigger state may be configured. If MPICapture object 0 is configured to trigger off of motor 0's index line, and then MPICapture object 1 is configured to trigger off of motor 0's home input, then only the capture trigger for motor 0 will have been reconfigured. Both
MPICapture object 0 and MPICapture object 1 will now trigger off of motor 0's home input.

Remarks

***external** should be NULL.

Return Values	
MPIMessageOK	

See Also

mpiCaptureConfigGet | mpiCaptureConfigReset

mpiCaptureStatus

Declaration

```
long mpiCaptureStatus(MPICapture          capture,
                      MPICaptureStatus    *status,
                      void                *external)
```

Required Header: stdmpi.h

Description

mpiCaptureStatus writes a Capture object's (***capture***) status into the structure pointed to by ***status***, and also into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

Remarks

external is reserved for future functionality and should always be set to NULL.

capture	a handle to a Capture object
***status**	a pointer to MPIStatus structure
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiCaptureConfigReset

Declaration

```
long mpiCaptureConfigReset(MPICapture   capture);
```

Required Header: stdmpi.h

Description

mpiCaptureConfigGet return the capture object to its unmapped state.

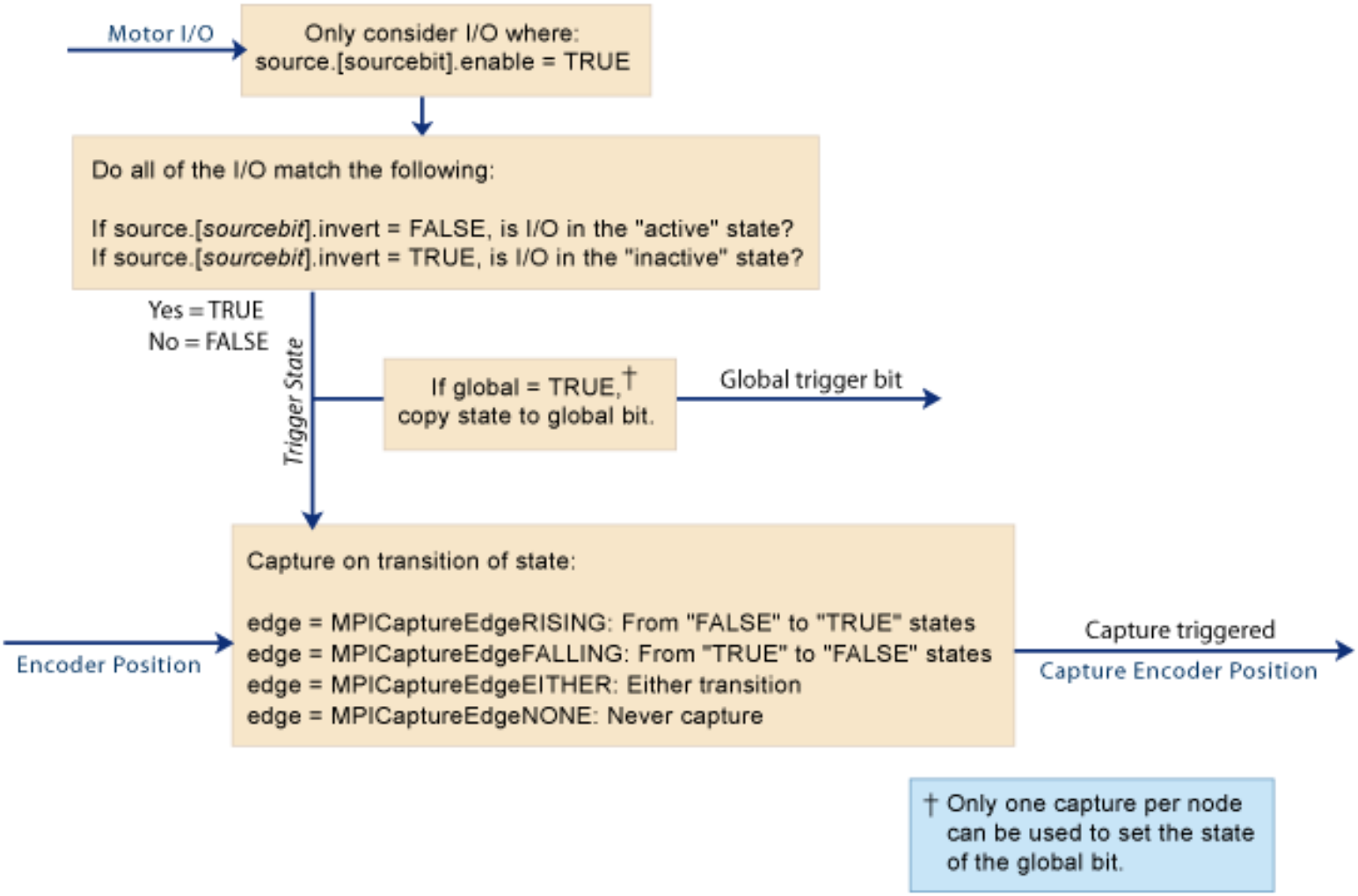
A capture object has no assumed resources, and is unmapped under default conditions. When a capture is first created, its captureMotorNumber and feedbackMotorNumber are unmapped. Once a capture has been configured, the next time that the capture object is created, it will retain the captureMotorNumber and feedbackMotorNumber that was previously assigned. mpiCaptureConfigReset(...) will return the capture object to its unmapped state.

If a capture has been previously configured (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet (...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

capture	a handle to a Capture object

Return Values	
MPIMessageOK	

Capture Engine Diagram



See Also

mpiCaptureConfigGet | mpiCaptureConfigSet | MPICaptureConfig

mpiCaptureArm

Declaration

```
long mpiCaptureArm(MPICapture      capture,
                   MPI_BOOL        arm)
```

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

Description

mpiCaptureArm arms or disarms ***capture***.

Value of "arm"	*Action of mpiCaptureArm*
FALSE	Disarms ***capture*** and sets the state of ***capture*** to MPICaptureStateIDLE
TRUE	Arms ***capture*** and sets the state of ***capture*** to MPICaptureStateARMED

Return Values	
MPIMessageOK	

See Also

MPICaptureState

mpiCaptureMemory

Declaration

```
long mpiCaptureMemory(MPICapture  capture,
                      void        **memory)
```

Required Header: stdmpi.h

Description

mpiCaptureMemory writes an address [which is used to access a Capture object's (***capture***) memory] to the contents of ***memory***. This address, or an address calculated from it, can be passed as the src parameter to mpiCaptureMemoryGet(...) and as the ***dst*** parameter to mpiCaptureMemorySet (...).

Return Values	
MPIMessageOK	

See Also

mpiCaptureMemoryGet | mpiCaptureMemorySet

mpiCaptureMemoryGet

Declaration

```
long mpiCaptureMemoryGet(MPICapture  capture,
                         void        *dst,
                         const void  *src,
                         long        count)
```

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

Description

mpiCaptureMemoryGet copies ***count*** bytes of a Capture object's (***capture***) memory (starting at address ***src***) and writes them into application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiCaptureMemory | mpiCaptureMemorySet

mpiCaptureMemorySet

Declaration

```
long mpiCaptureMemorySet(MPICapture  capture,
                         void        *dst,
                         const void  *src,
                         long        count)
```

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

Description

mpiCaptureMemorySet copies count bytes of application memory (starting at address ***src***) and writes them into a Capture object's (***capture***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiCaptureMemory | mpiCaptureMemoryGet

mpiCaptureControl

Declaration

```
long mpiCaptureControl(MPICapture capture);
```

Required Header: stdmpi.h
Change History: Added in the 03.02.00

Description

mpiCaptureControl returns a handle to the motion controller (Control) with which the Capture is associated.

capture	a handle to a Capture object

Return Values	
MPIControl	Handle to a Control object
MPIHandleVOID	If capture is invalid

See Also

mpiCaptureCreate | mpiControlCreate

mpiCaptureNumber

Declaration

```
long mpiCaptureNumber(MPICapture   capture,
                      long         *number)
```

Required Header: stdmpi.h

Description

mpiCaptureNumber reads the index of the capture block associated with the capture object and writes it into the contents of a long pointed to by encoder.

capture	a handle to a capture object
***number**	pointer to the capture number.

Return Values	
MPIMessageOK	

See Also

mpiCaptureCreate

MPICaptureConfig

Definition

```
typedef struct MPICaptureConfig {
    MPICaptureTrigger          source[MPICaptureSourceCOUNT];
                                    /* use MPICaptureSource to index */
    MPICaptureEdge             edge;
    MPICaptureTriggerGlobal    global;
    MPICaptureType             type;
    long                       captureMotorNumber;
    long                       feedbackMotorNumber; /* the same as
                                   captureMotorNumber for POSITION capture */
    MPIMotorEncoder            encoder;
    long                       captureIndex;    /* 0,1,... */
} MPICaptureConfig;
```

Description

source[MPICaptureSourceCOUNT]	An array of capture trigger source inputs. The capture can be configured to trigger from one or more sources. See MPICaptureTrigger and MPICaptureSourceCOUNT.
edge	An enumerated index to the trigger edge type. The capture can be configured to trigger from a variety of logic. See MPICaptureEdge.
global	A structure to configure the global capture, to chain capture block triggering. See MPICaptureTriggerGlobal.
type	Specifies either postion-based or time-based capture. Use MPICaptureTypePOSITION for position-based capture and MPICaptureTypeTIME for time-based capture.
captureMotorNumber	The number of the motor whose "source" (MPICaptureTrigger) is used to capture position.
feedbackMotorNumber	The number of the motor whose position is being returned from the capture event. (It must be the same as captureMotorNumber for position capture).
encoder	Specifies the encoder feedback being captured.
captureIndex	A zero-based index that specifies which capture resource on an axis is to be associated with the capture object. Each axis on a node has a given number of captures associated with it. An axis may have up to 4 capture resources on it. At present, no vendor provides a node with more than one capture resource, therefore, **captureIndex must be set to zero.**

Remarks

Time-based capture will only work correctly if the speed of an axis is less than 344 million counts per second.

See Also

MPICaptureType | mpiCaptureConfigGet | mpiCaptureConfigSet

MPICaptureEdge

Definition

```
typedef enum MPICaptureEdge {
    MPICaptureEdgeNONE,
    MPICaptureEdgeRISING,
    MPICaptureEdgeFALLING,
    MPICaptureEdgeEITHER,
} MPICaptureEdge;
```

Description

MPICaptureEdge is an enumeration of input trigger edge logic for a capture.

MPICaptureEdgeRISING	Triggers on a 0 to 1 transition.
MPICaptureEdgeFALLING	Triggers on a 1 to 0 transition.
MPICaptureEdgeEITHER	Triggers on either 0 to 1 or 1 to 0 transitions.

See Also

MPICaptureTrigger

MPICaptureMessage / MEICaptureMessage

Definition: MPICaptureMessage

```
typedef enum {
    MPICaptureMessageMOTOR_INVALID,
    MPICaptureMessageCAPTURE_TYPE_INVALID,
    MPICaptureMessageCAPTURE_INVALID,
    MPICaptureMessageENCODER_INVALID,
} MPICaptureMessage;
```

Description

MPICaptureEdge is an enumeration of Capture error messages that can be returned by the MPI library.

MEICaptureMessageMOTOR_INVALID

The capture motor number is not valid. This message code is returned by mpiCaptureConfigSet(...) if the captureMotorNumber does not have node hardware or the value is MPICaptureNOT_MAPPED.

MEICaptureMessageCAPTURE_TYPE_INVALID

The capture type is not valid. This message code is returned by mpiCaptureConfigSet(...) if the type is not one of the values defined by the enum MPICaptureType.

MPICaptureMessageCAPTURE_INVALID

The capture number is out of range. This message code is returned by mpiCaptureCreate(...) if the capture number is less than zero or greater than or equal to MEIXmpMaxCapturesPerMotor.

MPICaptureMessageENCODER_INVALID

The encoder index is out of range. This message code is returned by mpiCaptureCreate(...) if the encoder index is less than MPIMotorEncoderFIRST or greater than or equal to MPIMotorEncoderLAST. See MPIMotorEncoder.

See Also

mpiCaptureCreate | mpiControlConfigSet

Definition: MEICaptureMessage

```
typedef enum {
    MEICaptureMessageINVALID_EDGE,
    MEICaptureMessageGLOBAL_CONFIG_ERR,
    MEICaptureMessageGLOBAL_ALREADY_ENABLED,
    MEICaptureMessageCAPTURE_NOT_ENABLED,
    MEICaptureMessageCAPTURE_STATE_INVALID,
    MEICaptureMessageNOT_MAPPED,
    MEICaptureMessageUNSUPPORTED_PRIMARY,
    MEICaptureMessageUNSUPPORTED_SECONDARY,
    MEICaptureMessageSECONDARY_INDEX_INVALID,
    MEICaptureMessageCAPTURE_ARMED,
} MEICaptureMessage;
```

Change History: Modified in the 03.03.00

Description

MEICaptureMessageINVALID_EDGE

The encoder edge trigger type is not valid. This message code is returned by mpiCaptureConfigSet (...) if the encoder capture edge type is not a member of the MPICaptureEdge enumeration.

MEICaptureMessageGLOBAL_CONFIG_ERR

The global trigger configuration is not valid. This message code is returned by mpiCaptureConfigSet (...) if the capture's trigger source is set to global and the capture's global trigger is enabled simultaneously. To correct this problem, either set the capture's trigger source to global or enable the capture's global trigger (not both).

MEICaptureMessage_GLOBAL_ALREADY_ENABLED

The global trigger is already enabled. This message code is returned by mpiCaptureConfigSet(...) if a global trigger is already enabled on another capture on the same node. Only one global trigger enable is allowed per node. To prevent this problem, do not enable a second global trigger on a single node.

MEICaptureMessageCAPTURE_NOT_ENABLED

This value is returned by mpiCaptureCreate(...) when the capture number specified is greater than the number of captures enabled in firmware. See MPIControlConfig.

MEICaptureMessageCAPTURE_STATE_INVALID

This value is returned by mpiCaptureStatus(...) when the communication between the controller and the capture logic on the node fails resulting in an invalid capture state. See MPICaptureState.

MEICaptureMessageNOT_MAPPED

The capture object's hardware resource is not available. This message code is returned by mpiCaptureCreate(...) if the node hardware for the specified motor and encoder is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor and capture objects. A capture object cannot be created if there is no mapped hardware to support it. To correct this problem, verify that all expected nodes were found. Use meiSynqNetInfo(...) and meiSqNodeInfo(...) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEICaptureMessageUNSUPPORTED_PRIMARY

The capture hardware does not support the primary encoder. This message code is returned by mpiCaptureCreate(...) if the node hardware's primary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageUNSUPPORTED_SECONDARY

The capture hardware does not support the secondary encoder. This message code is returned by mpiCaptureCreate(...) if the node hardware's secondary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageSECONDARY_INDEX_INVALID

This message is returned from MPICaptureConfigSet(...) when the secondary encoder's index is specified as a trigger source in conjunction with other capture sources.

MEICaptureMessageCAPTURE_ARMED

The Capture resource being configured is already armed and cannot be reconfigured until it is disabled or triggered.

See Also

mpiCaptureCreate

MPICaptureSource

Definition

```
typedef enum MPICaptureSource {
    MPICaptureSourceMOTOR_IO_0,
    MPICaptureSourceMOTOR_IO_1,
    MPICaptureSourceMOTOR_IO_2,
    MPICaptureSourceMOTOR_IO_3,
    MPICaptureSourceMOTOR_IO_4,
    MPICaptureSourceMOTOR_IO_5,
    MPICaptureSourceMOTOR_IO_6,
    MPICaptureSourceMOTOR_IO_7,
    MPICaptureSourceHOME,
    MPICaptureSourceINDEX,
    MPICaptureSourceLIMIT_HW_NEG,
    MPICaptureSourceLIMIT_HW_POS,
    MPICaptureSourceGLOBAL,
    MPICaptureSourceINDEX_SECONDARY,
    MPICaptureSourceCOUNT,
} MPICaptureSource;
```

Description

MPICaptureSource is an enumeration of input trigger sources for a capture.

For dedicated input capture sources, (Home, Index, Limits, etc.) use the enums defined in MPICaptureSource.

For general input capture sources, you will need to look up the node specific input enum that matches the MPICaptureSourceMOTOR_IO values. You can determine the appropriate MPICaptureSourceMOTOR_IO by referencing the appropriate node header file. In the node specific header file (*manufacturer_model.h*), look for the NodeMotorIoConfig (replacing "Node" with the node name). The NodeMotorIoConfig contains the indices for the node specific I/O bits that correspond to the MPICaptureSource enums. Use the MPICaptureSource enum that matches the node specific enum to select the capture trigger source.

MPICaptureSourceMOTOR_IO_0	a capture trigger source is the 0 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_1	a capture trigger source is the 1 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_2	a capture trigger source is the 2 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_3	a capture trigger source is the 3 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_4	a capture trigger source is the 4 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_5	a capture trigger source is the 5 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_6	a capture trigger source is the 6 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_7	a capture trigger source is the 7 bit in the motor's configurable I/O.
MPICaptureSourceHOME	a capture trigger source is the HOME input in the dedicated I/O input.
MPICaptureSourceINDEX	a capture trigger source is the encoder INDEX input in the dedicated I/O input.
MPICaptureSourceLIMIT_HW_NEG	a capture trigger source is the Hardware Negative Limit input in the dedicated I/O input.
MPICaptureSourceLIMIT_HW_POS	a capture trigger source is the Hardware Positive Limit input in the dedicated IO word. Please see MPIMotorInfoDedicatedIn.
MPICaptureSourceGLOBAL	a capture trigger source is the Global capture signal found on the node. Please see MPICaptureTriggerGlobal.
MPICaptureSourceINDEX_SECONDARY	A a capture trigger source is the index on the secondary encoder. If position based capture is selected with the feedback source being the secondary encoder, this is the only valid capture source.

MPICaptureSourceCOUNT	Total number of possible input sources for a capture.

Example: RMB-10V
Configure MEI RMB-10V capture for trigger on XCVR_C.

From RMBMotorIoConfig in mei_rmb.h:

```
typedef enum {     /* index values for MEIMotorConfigIo[] */
  RMBMotorIoConfigINVALID = -1,

  RMBMotorIoConfigXCVR_A = MEIMotorIoConfigIndex0,
  RMBMotorIoConfigXCVR_B = MEIMotorIoConfigIndex1,
  RMBMotorIoConfigXCVR_C = MEIMotorIoConfigIndex2,
  RMBMotorIoConfigUSER_0_IN = MEIMotorIoConfigIndex6,
  RMBMotorIoConfigUSER_0_OUT = MEIMotorIoConfigIndex7,
  RMBMotorIoConfigUSER_1_IN = MEIMotorIoConfigIndex8,
  RMBMotorIoConfigUSER_1_OUT = MEIMotorIoConfigIndex9,
  RMBMotorIoConfigUSER_2_IN = MEIMotorIoConfigIndex10,
  RMBMotorIoConfigUSER_2_OUT = MEIMotorIoConfigIndex11,

  RMBMotorIoConfigLAST,
  RMBMotorIoConfigFIRST = RMBMotorIoConfigINVALID + 1
} RMBMotorIoConfig;
```

The matching enumeration value for XCVR_C is RMBMotorIoConfigXCVR_C. RMBMotorIoConfigXCVR_C is defined as index 2. Thus, the MPICaptureSource to use is MPICaptureSourceMOTOR_IO_2 (the third MPICaptureSourceMOTOR_IO value).

```
MPICaptureConfig    captureConfig;

/* enable capture source trigger for XCVR_C on mei_rmb */
captureConfig.source[MPICaptureSourceMOTOR_IO_2].enabled = TRUE;
```

If you want to set the capture source for a HOME, simply use the MPICaptureSourceHOME enum.

Example: Trust TA800
To configure the capture source for hall A on a Trust TA800 node, use MPICaptureSourceMOTOR_IO_0 (matches to node specific enum: TA800MotorIoConfigHALL_A). Remember that you will need to look in *trust_ta800.h* (the node module) to find TA800MotorIoConfigHALL_A.

See Also

MPICaptureTrigger

MPICaptureState

Definition

```
typedef enum {
    MPICaptureStateIDLE,
    MPICaptureStateARMED,
    MPICaptureStateCAPTURED,
    MPICaptureStateCLEAR,
} MPICaptureState;
```

Description

MPICaptureStateIDLE	Capture is not armed. This is the default state.
MPICaptureStateARMED	Capture is armed, but has not triggered yet.
MPICaptureStateCAPTURED	Capture triggered and position data is valid.
MPICaptureStateCLEAR	Capture is not armed, but has not transitioned to the IDLE state yet. This is an internal transitional state between CAPTURED and IDLE. It occurs when a capture is disarmed.

See Also

MPICaptureStatus

MPICaptureStatus

Definition

```
typedef struct MPICaptureStatus {
    MPICaptureState   state;
    double            latchedValue;
} MPICaptureStatus;
```

Description

state	An enumerated value representing the present state of the capture logic
latchedValue	The captured encoder position value. This value is only valid when the state is CAPTURED. The actual position value is the captured encoder position value subtracted by the origin variable. The origin variable can be set/get through mpiAxisOriginSet(...) and mpiAxisOriginGet(...). Recall that encoder positions have no origin adjustment whereas actual positions do have origin adjustment. Please refer to Using the Origin Variable for more information.

Sample Code

```
void displayPosition(MPICapture   capture,
                     MPIAxis      axis)
{
   double             origin;
   MPICaptureStatus   captureStatus;

   /* Check captured position */
   mpiCaptureStatus(capture,
                    &captureStatus,
                    NULL);

   /* Getting origin variable and store it to origin */
   mpiAxisOriginGet(axis, &origin);
```

```
   if (captureStatus.state == MPICaptureStateCAPTURED)
   {

        printf("Latched actual position: %.0lf\n",
                        captureStatus.latchedValue - origin);

        printf("Latched encoder position: %.0lf\n",
                        captureStatus.latchedValue);

    }

}
```

See Also

MPICaptureState

MPICaptureTrigger

Definition

```
typedef struct MPICaptureTrigger {
    MPI_BOOL enabled;
    MPI_BOOL invert;
} MPICaptureTrigger;
```

Change History: Modified in the 03.03.00

Description

The **MPICaptureTrigger** structure specifies the trigger configurations for a capture.

enabled	Enables or disables the trigger. A value of TRUE enables the trigger, FALSE disables the trigger.
invert	Normal or inverted trigger polarity. A value of FALSE indicates normal polarity, TRUE indicates inverted polarity.

See Also

MPICaptureSource

MPICaptureTriggerGlobal

Definition

```
typedef struct MPICaptureTriggerGlobal {
    MPI_BOOL   enabled;
} MPICaptureTriggerGlobal;
```

Change History: Modified in the 03.03.00

Description

The **MPICaptureTriggerGlobal** structure specifies the global input trigger configuration for a capture.

enabled	Enables or disables the global input trigger. A value of TRUE enables the trigger, FALSE disables the trigger.

See Also

MPICaptureConfig

MPICaptureType

Definition

```
typedef enum  {
    MPICaptureTypePOSITION,
    MPICaptureTypeTIME,
} MPICaptureType;
```

Description

MPICaptureTypePOSITION	An actual position is captured by the Node from its feedback source.
MPICaptureTypeTIME	An internal timer is captured by the node and then a captured position is interpolated by the XMP firmware.

Remarks

Time-based capture will only work correctly if the speed of an axis is less than 344 million counts per second.

See Also

MPICaptureConfig

MPICaptureNOT_MAPPED

Definition

```
#define MPICaptureNOT_MAPPED (-1)
```

Description

MPICaptureEdge is an enumeration of input trigger edge logic for a capture.

Capture objects are associated with the controller and are not mapped to any hardware resources under default conditions. MPICaptureNOT_MAPPED will be assigned to:

long captureMotorNumber;
long feedbackMotorNumber;

when mpiCaptureConfigGet is called for the first time on a capture object. After a capture object has been used once, the resource mapping will remain in place until it is reassigned.

See Also

Command Objects

Introduction

The **Command** object specifies one of a variety of program Sequence commands. These include motion, conditional branch, computational, and time delay commands.

Information about the different types of commands can be found on MPICommandType and MPICommandParams.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiCommand**Create**	Create Command object
mpiCommand**Delete**	Delete Command object
mpiCommand**Validate**	Validate Command object

Configuration and Informational Methods

mpiCommand**Label**	Get pointer to Command label
mpiCommand**Params**	Get Command parameters
mpiCommand**Type**	Return Command type

Other Methods

meiCommand**AxisListGet**	Get the axisCount and axisList from a Command object.

Data Types

MPICommand**Address**
MPICommand**Constant**
MPICommand**Expr**
MPICommand**Message**
MPICommand**Motion**
MPICommand**Operator**
MPICommand**Params**
MPICommand**Type**

meiCommandCreate

Declaration

```
MPICommand mpiCommandCreate(MPICommandType   type,
                            MPICommandParams *params,
                            const char       *label)
```

Required Header: stdmei.h

Description

meiCommandCreate creates a Command object. The command type is specified by ***type***. The type-specific parameters are specified by ***params***. If ***label*** is not Null (i.e., something meaningful), then branch commands can call this Command (by using the ***label***).

CommandCreate is the equivalent of a C++ constructor.

Return Values	
handle	to a Command object
MPIHandleVOID	if the object could not be created

See Also

mpiCommandDelete | mpiCommandValidate

mpiCommandDelete

Declaration

```
long mpiCommandDelete(MPICommand  command)
```

Required Header: stdmpi.h

Description

mpiCommandDelete deletes a Command object and invalidates its handle (***command***).

CommandDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiCommandCreate | mpiCommandValidate

mpiCommandValidate

Declaration

```
long mpiCommandValidate(MPICommand  command)
```

Required Header: stdmpi.h

Description

mpiCommandValidate validates the Command object and its handle (***command***).

command	a handle to the Command object

Return Values	
MPIMessageOK	

See Also

mpiCommandCreate | mpiCommandValidate

mpiCommandLabel

Declaration

```
long mpiCommandLabel(MPICommand  command,
                     char        **label)
```

Required Header: stdmpi.h

Description

mpiCommandLabel gets the string from a Command and puts it in the location pointed to by label.

command	a handle to the Command object
label	a pointer to a string returned by the method

Return Values	
MPIMessageOK	
pointer	to a Command's (***command***) label (that is in the location pointed to by ***label***)

See Also

mpiCommandCreate

mpiCommandParams

Declaration

```
long mpiCommandParams (MPICommand        command,
                       MPICommandParams *params)
```

Required Header: stdmpi.h

Description

mpiCommandParams gets the parameters from a Command and puts it in the location pointed to by params.

command	a handle to the Command object
***params**	a pointer to a MPICommandParams structure returned by the method

Return Values	
MPIMessageOK	
Command (*command*) parameters	in the structure pointed to by ***params***

See Also

mpiCommandCreate | MPICommandParams

mpiCommandType

Declaration

```
long mpiCommandType(MPICommand       command,
                    MPICommandType   *type)
```

Required Header: stdmpi.h

Description

mpiCommandType gets the type from a Command and puts it in the location pointed to by type.

command	a handle to the Command object
***type**	a pointer to a MPICommandType returned by the method

Return Values	
MPIMessageOK	
Command (*command*) parameters	in the location pointed to by ***type***

See Also

mpiCommandCreate | MPICommandType

meiCommandAxisListGet

Declaration

```
long meiCommandAxisListGet(MPICommand  command,
                           long        *axisCount
                           MPIAxis     *axisList)
```

Required Header: stdmei.h

Description

meiCommandAxisListGet reads number of axes and the list of axes associated with a motion type Command object (***command***) and writes them into the long pointed to by ***axisCount*** and the array of axis objects pointed to by ***axisList***.

command	a handle to the Command object
***axisCount**	a pointer to a long, representing the number of axes returned by the method
***axisList**	a pointer to an array of axis objects returned by the method

Return Values	
MPIMessageOK	

See Also

MPICommand | MPIAxis | MPIMotion

MPICommandAddress

Definition

```
typedef union {
    long    *l;
    float   *f;
} MPICommandAddress;
```

Description

MPICommandAddress defines a generic pointer that can specify either a long or a float pointer.

***l**	is used to access the long pointer of MPICommandAddress.
***f**	is used to access the float pointer of MPICommandAddress.

See Also

MPICommandConstant

MPICommandConstant

Definition

```
typedef union {
    long    l;
    float   f;
} MPICommandConstant;
```

Description

MPICommandConstant defines a generic variable that can specify either a ***long*** or ***float*** value.

l	is used to access the long value of MPICommandConstant.
f	is used to access the float value of MPICommandConstant.

See Also

MPICommandAddress

MPICommandExpr

Definition

```
typedef struct MPICommandExpr {
    MPICommandOperator   oper;
    MPICommandAddress    address;
 union {
     MPICommandConstant  value;  /* ['address'] 'oper' ['value'] */
     MPICommandAddress   ref;    /* ['address'] 'oper' [*'ref']  */
 } by;
} MPICommandExpr;
```

Description

MPICommandExpr is a structure that represents an expression for an MPICommand object.

The expression is evaluated as either:	
	*address **oper** value
	*address **oper** *ref
depending on the command type.	

See Also

MPICommand | MPICommandParams | MPICommandType

MPICommandMessage

Definition

```
typedef enum {
    MPICommandMessageCOMMAND_INVALID,
    MPICommandMessageTYPE_INVALID,
    MPICommandMessagePARAM_INVALID,
} MPICommandMessage;
```

Description

MPICommandMessageCOMMAND_INVALID

Currently not supported and is reserved for future use.

MPICommandMessageTYPE_INVALID

The command type is not valid. This message code is returned by mpiCommandCreate(...) if the command type is not a member of the MPICommandType enumeration.

MPICommandMessagePARAM_INVALID

Currently not supported and is reserved for future use.

See Also

MPICommandType

MPICommandMotion

Definition

```
typedef enum {
    MPICommandMotionABORT,
    MPICommandMotionE_STOP,
    MPICommandMotionE_STOP_MODIFY,
    MPICommandMotionE_STOP_ABORT,
    MPICommandMotionE_STOP_CMD_EQ_ACT,
    MPICommandMotionMODIFY,
    MPICommandMotionRESET,
    MPICommandMotionRESUME,
    MPICommandMotionSTART,
    MPICommandMotionSTOP,
} MPICommandMotion;
```

Change History: Modified in the 03.03.00

Description

MPICommandMotion is an enumeration of motion specific controller commands that can be used in a program sequence. It specifies a single motion action for the controller to execute. The CommandMotion also defines the command parameters that must be passed to mpiCommandCreate. For MPICommandMotion, there is a corresponding motion{...} structure in the MPICommandParams structure.

MPICommandMotionABORT	Commands an Abort action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionABORT for details.
MPICommandMotionE_STOP	Commands an E-Stop action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionE_STOP for details.
MPICommandMotionE_STOP_MODIFY	Commands an E-Stop Modify action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionE_STOP_MODIFY for details.

MPICommandMotionE_STOP_ABORT	Commands an E-Stop, then Abort action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionE_STOP_ABORT for details.
MPICommandMotionE_STOP_CMD_EQ_ACT	Commands an E-Stop (command position = actual position) action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionE_STOP_CMD_EQ_ACT for details.
MPICommandMotionMODIFY	Commands a Motion Modify on the motion supervisor associated with the motion object. Make sure to specify the MPIMotionType and MPIMotionParams in the MPICommandParams{...} structure. See mpiMotionModify(...) for details.
MPICommandMotionRESET	Commands a Reset action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionRESET for details.
MPICommandMotionRESUME	Commands a Resume action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionRESUME for details.
MPICommandMotionSTART	Commands a Motion Start on the motion supervisor associated with the motion object. Make sure to specify the MPIMotionType and MPIMotionParams in the MPICommandParams{...} structure. See mpiMotionStart(...) for details.
MPICommandMotionSTOP	Commands a Stop action on the motion supervisor associated with the motion object. See mpiMotionAction(...), MPIActionSTOP for details.

See Also

MPIAction | MPICommand | MPICommandParams

MPICommandOperator

Definition

```
typedef enum {
    /* Arithmetic operators */
    MPICommandOperatorADD,
    MPICommandOperatorSUBTRACT,
    MPICommandOperatorMULTIPLY,
    MPICommandOperatorDIVIDE,

    MPICommandOperatorAND,
    MPICommandOperatorOR,
    MPICommandOperatorXOR,

    /* Logical operators */
    MPICommandOperatorALWAYS,

    MPICommandOperatorEQUAL,
    MPICommandOperatorNOT_EQUAL,

    MPICommandOperatorGREATER_OR_EQUAL,
    MPICommandOperatorGREATER,

    MPICommandOperatorLESS_OR_EQUAL,
    MPICommandOperatorLESS,

    MPICommandOperatorBIT_CLEAR,
    MPICommandOperatorBIT_SET,
} MPICommandOperator;
```

Description

The following are operators used by the MPICommand and MPICompare objects.

Arithmetic Operators	
MPICommandOperatorADD	Performs an addition. Equivalent to the C operator (**+**).
MPICommandOperatorSUBTRACT	Performs a subtraction. Equivalent to the C operator (**-**).
MPICommandOperatorMULTIPLY	Performs a multiplication. Equivalent to the C operator (*).
MPICommandOperatorDIVIDE	Performs a divison. Equivalent to the C operator (**/**).
MPICommandOperatorAND	Performs a logical AND. Equivalent to the C operator (&).
MPICommandOperatorOR	Performs a logical OR. Equivalent to the C operator (**|**).
MPICommandOperatorXOR	Performs a logical XOR. Equivalent to the C operator (**^**).

Logical Operators	
MPICommandOperatorALWAYS	Always evaluates TRUE. Equivalent in C to (1) or TRUE.
MPICommandOperatorEQUAL	Performs an equality comparison. Equivalent to the C operator (= =)
MPICommandOperatorGREATER_OR_EQUAL	Performs an inequality comparison. Equivalent to the C operator (!=)
MPICommandOperatorGREATER_OR_EQUAL	Performs a greater than or equal to comparison. Equivalent to the C operator (>=)
MPICommandOperatorGREATER	Performs a greater than comparison. Equivalent to the C operator (**>**)
MPICommandOperatorLESS_OR_EQUAL	Performs a less than or equal to comparison. Equivalent to the C operator (< =)
MPICommandOperatorLESS	Performs a less than comparison. Equivalent to the C operator (**<**)
MPICommandOperatorBIT_CLEAR	Clears specified bits. Equivalent in C to the statement: variable &= ~(bits)
MPICommandOperatorBIT_SET	Sets specified bits. Equivalent in C to the statement: variable	= (bits)

See Also

MPICommand | MPICommandExpr | MPICommandParams

MPICommandParams

Definition

```
typedef union {
    struct {  /* *'dst' = 'value' */
        MPICommandAddress    dst;
        MPICommandConstant   value;
        MPIControl           control;  /* Ignored by Sequence */
    } assign;

    struct {  /* branch to 'label' on 'expr' */
        char            *label;   /* NULL => stop sequence */
        MPICommandExpr  expr;     /* expr.oper => MPICommandOperatorLogical */
        MPIControl      control;  /* Ignored by Sequence */
    } branch;

    struct {  /* branch to 'label' on MPIEventMask('handle') 'oper' 'mask' */
        char                *label; /* NULL => stop sequence */
        MPIHandle           handle; /* [MPIMotor|MPIMotion|...] */
        MPICommandOperator  oper;   /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
        MPIEventMask        mask;   /* MPIEventMask('handle') 'oper' 'mask' */
    } branchEvent;

    struct {  /* branch to 'label' on Io.input 'oper' 'mask' */
        char               *label; /* NULL => stop sequence */
        MPIIoType          type;   /* MOTOR,   USER */
        MPIIoSource        source; /* MPIMotor  index */
        MPICommandOperator oper;   /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
        long               mask;   /* [motor|user]Io.input 'oper' 'mask' */
    } branchIO;

    struct {  /* *'dst' = 'expr' */
        MPICommandAddress   dst;
        MPICommandExpr      expr;    /* expr.oper => MPICommandOperatorArithmetic */
        MPIControl          control; /* Ignored by Sequence */
    } compute;

    struct {  /* Io.output = Io.output 'oper' 'mask' */
        MPIIoType          type;    /* MOTOR,   USER */
        MPIIoSource        source;  /* MPIMotor    index */
        MPICommandOperator oper;    /* AND/OR/XOR */
        long               mask;
    } computeIO;

    struct {  /* memcpy(dst, src, count) */
        void         *dst;
        void         *src;
        long         count;
        MPIControl   control;  /* Ignored by Sequence */
    } copy;

    float  delay;  /* seconds */
```

```
    struct {
        long         value;    /* MPIEventStatus.type    = MPIEventTypeEXTERNAL */
                               /*                .source  = MPISequence/MPIProgram */
                               /*                .info[0] = value */
        MPIEventMgr  eventMgr; /* Ignored by Sequence */
    } event;

    struct {  /* mpiMotion[Abort|EStop|Reset|Resume|Start|Stop](motion[, type,
params]) */
        MPICommandMotion    motionCommand;
        MPIMotion           motion;
        MPIMotionType       type;    /* MPICommandMotionSTART */
        MPIMotionParams     params;  /* MPICommandMotionSTART */
    } motion;

    struct {  /* wait until 'expr' */
        MPICommandExpr      expr;    /* expr.oper => MPICommandOperatorLogical */
        MPIControl          control; /* Ignored by Sequence */
    } wait;

    struct {  /* wait until MPIEventMask('handle') 'oper' 'mask' */
        MPIHandle           handle; /* [MPIMotor|MPIMotion|...] */
        MPICommandOperator  oper;   /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
        MPIEventMask        mask;   /* MPIEventMask('handle') 'oper' 'mask' */
    } waitEvent;

    struct {  /* wait until Io.input 'oper' 'mask' */
        MPIIoType           type;     /* MOTOR,   USER */
        MPIIoSource         source;   /* MPIMotor   index */
        MPICommandOperator  oper;     /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
        long                mask;     /* [motor|user]Io.input 'oper' 'mask' */
    } waitIO;
} MPICommandParams;
```

Description

MPICommandParams holds the parameters used by an MPICommand. Each element in the MPICommandParams union corresponds to different types of commands (specified by the MPICommandType enumeration).

Element	Description	Supported by
assign	Assign a value to a particular controller address: ***dst = value** **assign.control** is currently not supported and is reserved for future use.	MPICommandTypeASSIGN MPICommandTypeASSIGN_FLOAT

branch	Branch to a particular command (similar to a *goto* statement) if a particular comparison evaluates to TRUE: branch to **label** on **expr** If *label* = NULL, then no more commands will be executed if the comparison evaluates to TRUE. **branch.control** is currently not supported and is reserved for future use.	MPICommandTypeBRANCH MPICommandTypeBRANCH_REF MPICommandTypeBRANCH_FLOAT MPICommandTypeBRANCH_FLOAT_REF
branchEvent	Branch to a particular command (similar to a *goto* statement) if a particular event occurs or has occured: branch to ***label*** on **MPIEventMask**(*handle*) **oper** *mask* If *label* = NULL, then no more commands will be executed if a particular event occurs or has occured.	MPICommandTypeBRANCH_EVENT
branchIO	Branch to a particular command (similar to a goto statement) if a particular i/o state matches a specified condition: branch to *label* on *Io.input* **oper** *mask* If *label* = NULL, then no more commands will be executed if a particular i/o state matches a specified condition.	MPICommandTypeBRANCH_IO
compute	perform some computation and place the result at some controller address: ***dst = expr** **compute.control** is currently not supported and is reserved for future use.	MPICommandTypeCOMPUTE MPICommandTypeCOMPUTE_REF MPICommandTypeCOMPUTE_FLOAT MPICommandTypeCOMPUTE_FLOAT_REF
computeIO	Performs a computation on a set of i/o bits: *Io.output* = *Io.output* **oper** *mask*	MPICommandType_IO
copy	Copies controller memory from one place to another: **memcpy**(*dst, src, count*); Remember: ***count*** represents the number of **bytes** copied, NOT the number of controller words. **event.control** is currently not supported and is reserved for future use.	MPICommandTypeCOPY
delay	Delays execution of the next command *delay* seconds.	MPICommandTypeDELAY
event	Generates an event: MPIEventStatus.type = MPIEventTypeEXTERNAL MPIEventStatus.source = MPISequence MPIEventStatus.info[0] = value **event.eventMgr** is currently not supported and is reserved for future use.	MPICommandTypeEVENT
motion	Commands a motion action (See MPICommandMotion): **mpiMotionStart** (*motion, type, params*]); or **mpiMotionAction**(*motion*, MPIAction[ABORT | E_STOP | E_STOP_ABORT | RESET | RESUME | STOP]);	MPICommandTypeMOTION

wait	Delays execution of the next command until a particular comparison evaluates to TRUE: wait until *expr* **wait.control** is currently not supported and is reserved for future use.	MPICommandTypeWAIT MPICommandTypeWAIT_REF MPICommandTypeWAIT_FLOAT MPICommandTypeWAIT_FLOAT_REF
waitEvent	Delays execution of the next command until a particular event occurs: wait until **MPIEventMask** (*handle*) **oper** *mask*	MPICommandTypeWAIT_EVENT
waitIO	Delays execution of the next command until a particular i/o state matches a specified condition: wait until *Io.input* **oper** *mask*	MPICommandTypeWAIT_IO

See Also

MPICommand | MPICommandType | mpiCommandCreate | mpiCommandParams

MPICommandType

Definition

```
typedef enum {
    MPICommandTypeASSIGN,
    MPICommandTypeASSIGN_FLOAT,

    MPICommandTypeBRANCH,
    MPICommandTypeBRANCH_REF,
    MPICommandTypeBRANCH_FLOAT,
    MPICommandTypeBRANCH_FLOAT_REF,
    MPICommandTypeBRANCH_EVENT,
    MPICommandTypeBRANCH_IO,

    MPICommandTypeCOMPUTE,
    MPICommandTypeCOMPUTE_REF,
    MPICommandTypeCOMPUTE_FLOAT,
    MPICommandTypeCOMPUTE_FLOAT_REF,
    MPICommandTypeCOMPUTE_IO,

    MPICommandTypeCOPY,
    MPICommandTypeDELAY,
    MPICommandTypeEVENT,
    MPICommandTypeMOTION,

    MPICommandTypeWAIT,
    MPICommandTypeWAIT_REF,
    MPICommandTypeWAIT_FLOAT,
    MPICommandTypeWAIT_FLOAT_REF,
    MPICommandTypeWAIT_EVENT,
    MPICommandTypeWAIT_IO,

} MPICommandType;
```

Description

MPICommandType is an enumeration of controller commands that can be used in a program sequence. It specifies a single instruction for the controller to execute. The CommandType also defines the command parameters that must be passed to mpiCommandCreate(...). For each MPICommandType there is a corresponding structure in the MPICommandParams{...} union. For example, when the MPICommandTypeASSIGN is specified, the assign{...} structure in MPICommandParams{...} must be filled in to specify the address and value.

Commands must be created with mpiCommandCreate(...) and then added to a sequence using

mpiSequenceCommandAppend(...), mpiSequenceCommandInsert(...), or mpiSequenceCommandListSet(...). Then the command sequence can be loaded into the controller with mpiSequenceLoad(...) and started with mpiSequenceStart(...).

Element	Description	Associated MPICommandParams structure
MPICommandTypeASSIGN	Writes a constant value (long or float) into the controller's memory at the specified address.	assign
MPICommandTypeASSIGN_FLOAT	These commands assign a value to a particular controller address. MPICommandTypeASSIGN assigns a long value while MPICommandTypeASSIGN_FLOAT assigns a float value.	
MPICommandTypeBRANCH	These commands branch to a particular command (similar to a goto statement) if a particular comparison evaluates to TRUE. MPICommandTypeBRANCH compares a controller address to a specified constant long value. MPICommandTypeBRANCH_REF compares a controller address to a long value at a specified controller address.	branch
MPICommandTypeBRANCH_REF	Branch to a particular command if the comparison evaluates to TRUE. Compares a controller address to a long value at a specified controller address.	
MPICommandTypeBRANCH_FLOAT	Compares a controller address to a specified constant float value.	
MPICommandTypeBRANCH_FLOAT_REF	Compares a controller address to a float value at a specified controller address.	
MPICommandTypeBRANCH_EVENT	Branch to a particular command (similar to a goto statement) if a particular event occurs or has occured.	branchEvent
MPICommandTypeBRANCH_IO	Branch to a particular command (similar to a goto statement) if a particular I/O state matches a specified condition.	branchIO

MPICommandTypeCOMPUTE	These commands perform some computation and place the result at some controller address. MPICommandTypeCOMPUTE performs a computation of some controller address and a constant long value.	compute
MPICommandTypeCOMPUTE_REF		
MPICommandTypeCOMPUTE_FLOAT	Performs a computation of some controller address and a constant float value.	
MPICommandTypeCOMPUTE_FLOAT_REF	Performs a computation of some controller address and a float value at a specified controller address.	
MPICommandTypeCOMPUTE_IO	Performs a computation on a set of I/O bits.	computeIO
MPICommandTypeCOPY	Copies controller memory from one place to another.	copy
MPICommandTypeDELAY	Delays execution of the next command.	delay
MPICommandTypeEVENT	Generate an event.	event
MPICommandTypeMOTION	Commands a motion action. See MPICommandMotion.	motion
MPICommandTypeWAIT	These delays execution of the next command until a particular comparison evaluates to TRUE. MPICommandTypeWAIT compares a controller address to a specified constant long value. MPICommandTypeWAIT_REF Compares a controller address to a long value at a specified controller address.	wait
MPICommandTypeWAIT_REF	Compares a controller address to a long value at a specified controller address.	
MPICommandTypeWAIT_FLOAT	Compares a controller address to a specified constant float value.	
MPICommandTypeWAIT_FLOAT_REF	Compares a controller address to a float value at a specified controller address.	

MPICommandTypeWAIT_EVENT	Delays execution of the next command until a particular event occurs.	waitEvent
MPICommandTypeWAIT_IO	Delays execution of the next command until a particular I/O state matches a specified condition.	waitIO

See Also

MPICommand | MPICommandMotion | MPICommandParams | mpiCommandCreate | mpiCommandType | mpiSequenceCommandAppend | mpiSequenceCommandInsert | mpiSequenceCommandListSet | mpiSequenceLoad | mpiSequenceStart

Compensator Objects

Introduction

A **Compensator** object manages a single compensation table. Its primary function is to provide an interface to configure both the compensating axes and the compensated axis. It also provides an interface for loading the on-controller compensation tables. The Compensator object is a host-based object that has a corresponding compensator object embedded on the controller. The embedded compensator handles the real-time issues associated with axis position compensation.

Before creating the MPI Compensator object, the corresponding embedded compensator object on the controller must be enabled. Also, before configuring the MPI Compensator object, the controller's compensation table must be allocated with a sufficient size to hold all required compensation values (or points). Both of these items can be configured using mpiControlConfigGet/Set(...) methods.

NOTE: Configuring the compensator table size using mpiControlConfigSet(...) will reallocate the controller's dynamic memory. Reallocating dynamic memory on the controller affects multiple objects and should only be done at the very beginning of your application.

For more information on determining compensation table size please see Determining Required Compensator Table Size.

See Also:
Configuring the Compensator Objects for Operation
Determining Required Compensator Table Size
Loading the Compensation Table
Setting up an area for 2D Position Compensation

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiCompensator**Create**	Create Compensator object
mpiCompensatorl**Delete**	Delete Compensator object
mpiCompensator**Validate**	Validate Compensator object

Configuration and Information Methods

mpiCompensator**ConfigGet**	Get Compensator configuration
mpiCompensator**ConfigSet**	Set Compensator configuration
meiCompensator**Info**	Get Compensator information
meiCompensator**TableGet**	Get Compensator table
meiCompensator**TableSet**	Set Compensator table

Memory Methods

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

Relational Methods

meiCompensator**Control**	Return handle of Control object associated with Compensator
meiCompensator**Number**	Get number of Compensator

Data Types

- MPICompensator**Config**
- MPICompensator**Dimension**
- MPICompensator**Info**
- MPICompensator**InputAxis**
- MPICompensator**Message**
- MPICompensator**Range**

Constants

- MPICompensator**DimensionsMAX**

mpiCompensatorCreate

Declaration

```
MPICompensator  mpiCompensatorCreate(MPIControl  control,
                                     long        number);
```

Required Header: stdmpi.h

Description

mpiCompensatorCreate creates a Host Compensator object associated with the compensation object identified by ***number*** located on motion controller ***control***. CompensatorCreate is the equivalent of a C++ constructor.

Valid compensator numbers are zero (0) to MPIControlMAX_COMPENSATORS.

Before creating a Compensator object, the controller compensation objects must be enabled using MPIControlConfig.compensatorCount, or the host object will be invalid.

Return Values	
handle	handle to a Compensator object
MPIHandleVOID	if the object could not be created

See Also

mpiCompensatorDelete | mpiCompensatorValidate | MPIControlConfig

mpiCompensatorDelete

Declaration

```
long  mpiCompensatorDelete(MPICompensator  compensator);
```

Required Header: stdmpi.h

Description

mpiCompensatorDelete deletes a host Compensator object (***compensator***) and invalidates its handle.

CompensatorDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiCompensatorCreate | mpiCompensatorValidate

mpiCompensatorValidate

Declaration

```
long  mpiCompensatorValidate(MPICompensator  compensator);
```

Required Header: stdmpi.h

Description

mpiCompensatorValidate validates the Compensator object (***compensator***) and its handle. Always call mpiCompensatorValidate after creating a new Compensator object.

Return Values	
MPIMessageOK	
MPICompensatorMessageCOMPENSATOR_INVALID	
MPICompensatorMessageNOT_ENABLED	

See Also

mpiCompensatorCreate | mpiCompensatorDelete

mpiCompensatorConfigGet

Declaration

```
long  mpiCompensatorConfigGet(MPICompensator         compensator,
                              MPICompensatorConfig   *config,
                              void                   *external);
```

Required Header: stdmpi.h

Description

mpiCompensatorConfigGet gets the configuration of a Compensator object (***compensator***) 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 intended for future use and is not currently used. Set this value to NULL.

Return Values	
MPIMessageOK	
MPIMessagePARAM_INVALID	

See Also

mpiCompensatorConfigSet | MEICompensatorConfig

mpiCompensatorConfigSet

Declaration

```
long  mpiCompensatorConfigSet(MPICompensator        compensator,
                              MPICompensatorConfig  *config,
                              void                  *external);
```

Required Header: stdmpi.h

Description

mpiCompensatorConfigSet sets (writes) the configuration of a Compensator object (***compensator***) 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: ***config*** or ***external*** can be NULL (but both cannot be NULL).

Remarks

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

Return Values	
MPIMessageOK	
MPIMessagePARAM_INVALID	
MPIMessageARG_INVALID	See MPICompensatorConfig.
MPICompensatorMessageDIMENSION_NOT_SUPPORTED	
MPICompensatorMessageAXIS_NOT_ENABLED	
MPICompensatorMessagePOSITION_DELTA_INVALID	
MPICompensatorMessageTABLE_SIZE_ERROR	

See Also

mpiCompensatorConfigGet | MEICompensatorConfig

mpiCompensatorInfo

Declaration

```
long  mpiCompensatorInfo(MPICompensator      compensator,
                         MPICompensatorInfo  *info);
```

Required Header: stdmpi.h

Description

mpiCompensatorInfo reads the static information about the compensator object, and writes it into the structure pointed to by ***info***.

compensator	a handle to the Compensator object.
***info**	a pointer to a compensator information structure.

Return Values	
MPIMessageOK	
MPICompensatorMessageNOT_CONFIGURED	

See Also

MPICompensatorInfo | MPIControlConfig | mpiCompensatorTableGet | mpiCompensatorTableSet | mpiCompensatorInfo

mpiCompensatorTableGet

Declaration

```
long  mpiCompensatorTableGet(MPICompensator   compensator,
                             long             *table);
```

Required Header: stdmpi.h

Description

mpiCompensatorTableGet reads the NxM Compensator table stored on the controller whose dimensions are defined by the values in the MPICompensatorConfig structure. These values are written into the location specified by ***table***.

NOTE: The array pointed to ***table*** must have enough memory allocated to hold the entire size of the configured compensation table.

Return Values	
MPIMessageOK	
MPICompensatorMessageNOT_CONFIGURED	
MPIMessagePARAM_INVALID	

See Also

mpiCompensatorTableSet | MPICompensatorConfig

mpiCompensatorTableSet

Declaration

```
long  mpiCompensatorTableSet(MPICompensator   compensator,
                             long             *table);
```

Required Header: stdmpi.h

Description

mpiCompensatorTableSet writes the values stored in the location specified by ***table** to the Compensator table stored on the controller.

NOTE: The array pointed to ***table** must have a size large enough to fill the configured compensation table size (as defined by the MPICompensatorConfig structure) or memory access violations may occur.

Return Values	
MPIMessageOK	
MPICompensatorMessageNOT_CONFIGURED	
MPIMessagePARAM_INVALID	

See Also

mpiCompensatorTableGet | MPICompensatorConfig

mpiCompensatorMemory

Declaration

```
long  mpiCompensatorMemory(MPICompensator   compensator,
                           void             **memory);
```

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiCompensatorMemoryGet | mpiCompensatorMemorySet

mpiCompensatorMemoryGet

Declaration

```
long  mpiCompensatorMemoryGet(MPICompensator   compensator,
                              void             *dst,
                              const   void     *src,
                              long             count);
```

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

Description

mpiCompensatorMemoryGet copies ***count*** bytes of a Compensator's (***compensator***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiCompensatorMemorySet | mpiCompensatorMemory

mpiCompensatorMemorySet

Declaration

```
long  mpiCompensatorMemorySet(MPICompensator   compensator,
                              void             *dst,
                              const   void     *src,
                              long             count);
```

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

Description

mpiCompensatorMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Compensator's (***compensator***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiCompensatorMemoryGet | mpiCompensatorMemory

mpiCompensatorControl

Declaration

```
MPIControl  mpiCompensatorControl(MPICompensator   compensator);
```

Required Header: stdmpi.h

Description

mpiCompensatorControl returns a handle to the Control object with which the compensator is associated.

compensator	a handle to the Compensator object

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

See Also

mpiCompensatorCreate | mpiControlCreate

mpiCompensatorNumber

Declaration

```
long  mpiCompensatorNumber(MPICompensator   compensator,
                           long             *number);
```

Required Header: stdmpi.h

Description

mpiCompensatorNumber writes the index of a compensation object (object on the motion controller that the Compensator object is associated with) to the contents of ***number***.

Return Values	
MPIMessageOK	

See Also

mpiCompensatorCreate

MPICompensatorConfig

Definition

```
typedef struct MPICompensatorConfig {
    long                    dimensionCount,
    MPICompensatorInputAxis inputAxis[MPICompensatorDimensionMAX],
    long                    outputAxisNumber,
} MPICompensatorConfig;
```

Description

dimensionCount	The input dimension count of the compensation table. Valid values are from zero (0) to MPICompensatorDimensionsMAX. A value of zero (0) effectively disables the compensation object
inputAxis	The substructure used to configure each input dimension of the Compensator object.
outputAxisNumber	This specifies the axis number of the Axis to be compensated by the Compensator object. This number must correspond to a valid (existing) and enabled Axis on the controller.

See Also

MPIControlConfig | mpiCompensatorConfigGet | mpiCompensatorConfigSet | MPICompensatorDimension | MPICompensatorDimensionsMAX

MPICompensatorDimension

Definition

```
typedef struct MPICompensatorDimension {
    MPICompensatorDimensionX,
    MPICompensatorDimensionY,
} MPICompensatorDimension;
```

Description

MPICompensatorDimension an enumeration of valid Compensator dimensions.

MPICompensatorDimensionX	First Compensating Dimension
MPICompensatorDimensionY	Second Compensating Dimension

See Also

MPICompensatorConfig | MPICompensatorInfo

MPICompensatorInfo

Definition

```
typedef struct MPICompensatorInfo {
    long     tableDimensions[MPICompensatorDimensionMAX],
    long     tableSizeBytes,
} MPICompensatorInfo;
```

Description

tableDimensions	The dimensions along each axis of the table. This value is affected by the values set in the MPICompensatorConfig structure.
tableSizeBytes	The size (in Byte) required to store the entire compensation table in a host resident structure or array. This value is affected by the values set in the MPICompensatorConfig structure. This is NOT the amount of memory allocated on the controller by setting the MPIControlConfig. compensatorPointCount value.

See Also

MPICompensatorConfig | MPIControlConfig | MPICompensatorDimension | MPICompensatorDimensionMAX

MPICompensatorInputAxis

Definition

```
typedef struct MPICompensatorInputAxis {
    long                    axisNumber,
    MPICompensatorRange     range,
    long                    positionDelta,
} MPICompensatorInputAxis;
```

Description

axisNumber	This specifies the axis number of the compensating Axis object. The position from this Axis will be used to index a single dimension of the compensation table. This number must correspond to a valid (existing) and enabled Axis on the controller.
range	Used to configure the feedback positions along the compensation axis where compensation will start and end.
positionDelta	Spacing between compensation positions on the compensating axis: **positionDelta** must meet some specifications: • positionDelta must be an exact multiple of the range (i.e. ((range. positionMax – range.positionMin) / positionDelta) must be an integer value). • positionDelta must be greater than zero. • positionDelta must be greater than (range.positionMax - range. positionMin).

See Also

MPICompensatorConfig | mpiCompensatorConfigGet | mpiCompensatorConfigSet

MPICompensatorMessage

Definition

```
typedef enum {
   MPICompensatorMessageCOMPENSATOR_INVALID,
   MPICompensatorMessageNOT_CONFIGURED,
   MPICompensatorMessageNOT_ENABLED,
   MPICompensatorMessageAXIS_NOT_ENABLED,
   MPICompensatorMessageTABLE_SIZE_ERROR,
   MPICompensatorMessagePOSITION_DELTA_INVALID,
   MPICompensatorMessageDIMENSION_NOT_SUPPORTED,
} MPICompensatorMessage;
```

Description

MPICompensatorMessageCOMPENSATOR_INVALID

The compensator number is not valid. This message code is returned by mpiCompensatorCreate(...) if the compensator number is less than 0 or greater than MPIControlMAX_COMPENSATORS.

MPICompensatorMessageNOT_CONFIGURED

MPI Compensator object must be configured before calling mpiCompensatorTableGet/ Set or mpiCompensatorInfo(...).

MPICompensatorMessageNOT_ENABLED

The compensator is not available on the controller. This message code is returned by mpiCompensatorValidate(…) if the compensator number is not within the range of enabled compensators on the controller. To correct the problem, use MPIControlConfig to configure the compensatorCount to be greater than the required compensator number.

MPICompensatorMessageAXIS_NOT_ENABLED

The axis is not available on the controller. This message coder is returned by mpiCompensatorConfigSet(...) if the axisOutNumber or any of the inputAxis[n].axisNumbers are not within the range of enabled axes on the controller. To correct the problem, use MPIControlConfig to configure the axisCount to be greater than the required axis number.

MPICompensatorMessageTABLE_SIZE_ERROR

The host compenstation table will not fit within the controller's configured compensation table. See Determining Required Compensation Table Size.

MPICompensatorMessagePOSITION_DELTA_INVALID

The positionDelta is either out of range or is not a multiple of the range. This message code is returned by mpiCompensatorConfigSet(…). To correct the problem, check the valid range values for MPICompensatorInputAxis.

MPICompensatorMessageDIMENSION_NOT_SUPPORTED

The dimensionCount is out of range. This message code is returned by mpiCompensatorConfigSet(…). To correct the problem, check the valid range values for MPICompensatorConfig.

See Also

MPICompensatorRange

Definition

```
typedef struct MPICompensatorRange {
    double  positionMin,
    double  positionMax,
} MPICompensatorRange;
```

Change History: Modified in the 03.04.00.

Description

positionMin	The minimum feedback position (counts) along the compensation axis where compensation will occur.
positionMax	The maximum feedback position (counts) along the compensation axis where compensation will occur.

See Also

MPICompensatorConfig | mpiCompensatorConfigGet | mpiCompensatorConfigSet

MPICompensatorDimensionsMAX

Definition

```
#define MPICompensatorDimensionsMAX (MPICompensatorDimensionLAST)
```

Description

MPICompensatorDimensionMAX defines the maximum number of dimensions supported by the Compensator object's compensation tables. Currently, the maximum dimension value is 2.

See Also

MPICompensatorDimension | MPICompensatorInfo | MPICompensatorConfig

Determining Required Compensator Table Size

The compensator table size is dependent on the number of dimensions (1D or 2D), the position range, and the resolution (or granularity) of the compensation points. The compensator uses linear interpolation to calculate the compensation values between each distinct compensation point.

For each compensation axis there are three position values: Min, Max, and Delta. The compensating range for an axis is specified by the Min and Max positions along the axis. The range (Max – Min) divided by Delta, determines the number of required points for the compensator. You can calculate the number of required compensator points by using the following equations:

1D Compensation: Points = (positionMax-positionMin) / positionDelta + 1

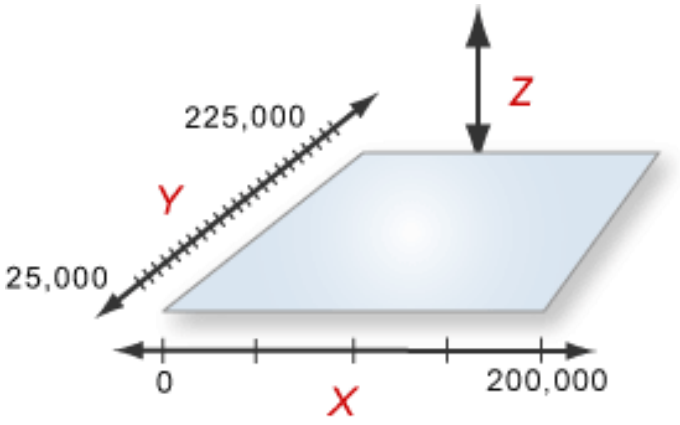
2D Compensation: Points = PointsX * PointsY

NOTE: Delta must be an exact multiple of the difference between Min and Max.

Example: (taken from comp.c sample application)

To compensate a Z (vertical) axis for X-Y surface irregularities, first define the X-Y area to be compensated (Xmin to Xmax, Ymin to Ymax). Then define the spacing of the measuring points (delta) for the X and Y axes to determine the compensation table size.

For the X-Y table diagram below we have:
Xmin = 0, Xmax = 200000, Xdelta = 50000
Ymin = 25000, Ymax = 225000, Ydelta = 10000



For this table our X & Y dimensions are:
X_DIM = (200000-0)/50000 + 1 = 5
Y_DIM = (225000-25000)/10000 +1 = 21

which requires a table point count of:
Points = X_DIM * Y_DIM = 105

With this information we can now configure the size of our compensation table on the controller using MPIControlConfig.compensatorPointCount = 105.

Configuring the Compensator Objects for Operation

After determining the required compensator table size, we need to configure both the embedded compensation tables on controller and the MPI Compensator object.

We will illustrate how to do this using the X-Y-Z system defined in the Determining Required Compensator Table Size section.

Configuring Controller Compensation Table
From our example in the previous section we have calculated that we need at least a point count of 105 to hold all of our measured compensation points (Acquiring and loading compensation points will be described in in the next section). First we need tell the motion controller to allocate memory space to hold the compensation table. We also need to enable a compensator since compensator objects are disabled on the controller by default. For an example, see the code below.

```
MPIControlConfig config;
long returnValue; returnValue =
    mpiControlConfigGet(control,
                        &config,
                        NULL);

if (returnValue == MPIMessageOK) {
    /* configure first compensator table size so our 2D array will fit */
    config.compensatorCount = 1;
    config.compensatorPointCount[0] = 105;

    /*
     * WARNING: this is a low-level configuration that will
     * reinitialize the controller's dynamic memory buffers!
     * Only preform this operation at system initialization.
    */
     returnValue =
         mpiControlConfigSet(control,
                             &config,
                             NULL);
}
```

The comment above reminds us that calling mpiControlConfigSet(...) will reallocate dynamic memory. Reallocation of dynamic memory affects other objects on the controller, so it should only be done during system initialization and not during the execution of a move.

Configuring the MPI Compensator Object
Continuing with our example, we will now assume that our axis numbers for axis X, Y, and Z are 0, 1, & 2 respectively. If we also assume that the MPI Compensator object has already been created, the code to configure the object would look like the following:

```
if (returnValue == MPIMessageOK) {
       MPICompensatorConfig config;

       returnValue =
              mpiCompensatorConfigGet(compensator,
                                      &config,
                                      NULL);
}

       if (returnValue == MPIMessageOK) {
              config.dimensionCount = 2;

              /* configure first compensating (input) axis */
              config.inputAxis[0].axisNumber = 0;
              config.inputAxis[0].range.positionMin = 0;
              config.inputAxis[0].range.positionMax = 250000;
              config.inputAxis[0].positionDelta = 50000;

              /* configure second compensating (input) axis */
              config.inputAxis[1].axisNumber = 1;
              config.inputAxis[1].range.positionMin = 25000;
              config.inputAxis[1].range.positionMax = 225000;
              config.inputAxis[1].positionDelta = 10000;

              /* configure compensated (out) axis */
              config.outputAxisNumber = 2;

              returnValue =
                  mpiCompensatorConfigSet(compensator,
                                          &config,
                                          NULL); }
```

Once we have the Compensation table allocated and have a configured Compensation object, the last step is to Load the Compensation Table.

Loading the Compensation Table

Next we need to somehow acquire high precision distance measurements (via interferometer, etc.) to the surface at each of the X-Y locations in the compensation area, and store the X and Y offset positions.

Once you've obtained these positions, they will need to be loaded into our previously configured compensation table (See Determining Required Compensator Table Size). Continuing with our original example let's assume that our measurements are as defined by the following table below (taken from the comp.c sample application):

```
long compensatorTable[21][5] =
{
      { 0,    0,    0,     0,     0, },
      { 100,  200,  -200,  -100,  0, },
      { 200,  400,  -400,  -200,  0 },
      { 300,  600,  -600,  -300,  0 },
      { 400,  800,  -800,  -400,  0 },
      { 500,  1000, -1000, -500,  0 },
      { 600,  1200, -1200, -600,  0 },
      { 700,  1400, -1400, -700,  0 },
      { 800,  1600, -1600, -800,  0 },
      { 900,  1800, -1800, -900,  0 },
      { 1000, 2000, -2000, -1000, 0 },
      { 900,  1800, -1800, -900,  0 },
      { 800,  1600, -1600, -800,  0 },
      { 700,  1400, -1400, -700,  0 },
      { 600,  1200, -1200, -600,  0 },
      { 500,  1000, -1000, -500,  0 },
      { 400,  800,  -800,  -400,  0 },
      { 300,  600,  -600,  -300,  0 },
      { 200,  400,  -400,  -200,  0 },
      { 100,  200,  -200,  -100,  0 },
      { 0,    0,    0,      0,    0 },
};
```

Interpreting compensator table above:

To compensate a Z (vertical) axis for X-Y surface irregularities, first define the X-Y area to be compensated (Xmin to Xmax, Ymin to Ymax).

Then define the spacing of the measuring points (delta) for the X and Y axes to determine the compensation table size.

For the X-Y table diagram below we have:

Xmin = 0, Xmax = 200000, Xdelta = 50000

Ymin = 25000, Ymax = 225000, Ydelta = 10000

Y Values	X Values				
	0	50000	100000	150000	200000
25000	0	0	0	0	0
35000	100	200	-200	-100	0
45000	200	400	-400	-200	0
55000	300	600	-600	-300	0
65000	400	800	-800	-400	0
75000	500	1000	-1000	-500	0
85000	600	1200	-1200	-600	0
95000	700	1400	-1400	-700	0
105000	800	1600	-1600	-800	0
115000	900	1800	-1800	-900	0
125000	1000	2000	-2000	-1000	0
135000	900	1800	-1800	-900	0
145000	800	1600	-1600	-800	0
155000	700	1400	-1400	-700	0
165000	600	1200	-1200	-600	0
175000	500	1000	-1000	-500	0
185000	400	800	-800	-400	0
195000	300	600	-600	-300	0
205000	200	400	-400	-200	0
215000	100	200	-200	-100	0
225000	0	0	0	0	0

To load the above compensation table, execute the following code:

```
returnValue = mpiCompensatorTableSet(compensator,
                                     (long*)compensatorTable);
```

Once the compensation positions are loaded, the compensation will be applied to the Z-axis' position feedback loop every servo cycle.

NOTE: No more interpolated compensation of the Z-axis will occur outside of the defined compensation range. Therefore, the compensation of the Z-axis will remain fixed outside of this range.

Setting up an area for 2D Position Compensation

The XMP has 2D compensation capabilities. The user-supplied compensation table is downloaded into XMP memory. The XMP automatically applies this compensation information to optimize motion profiles in real time.

Control Objects

Introduction

A **Control** object manages a motion controller device. The device is typically a single board residing in a PC or an embedded system. A control object can read and write device memory through one of a variety of methods: I/O port, memory mapped or device driver.

For the case where the application and the motion controller device exist on two physically separate platforms connected by a LAN or serial line, the application creates a client control object which communicates via remote procedure calls with a server.

Unlike the methods of all other objects in the MPI, Control object methods are not thread-safe.

Are you using TCP/IP and Sockets? If yes, click here.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiControl**Create**	Create Control object
mpiControl**Delete**	Delete Control object
mpiControl**Validate**	Validate Control object

Configuration and Information Methods

mpiControl**Address**	Get original address of Control object (when it was created)
mpiControl**ConfigGet**	Get Control config
mpiControl**ConfigSet**	Set Control config
mpiControl**DigitalIn**	Read the current input state
mpiControl**DigitalOutGet**	Read the current output state
mpiControl**DigitalOutSet**	Change the state of output bit
meiControl**ExtMemAvail**	Gets the amount of external memory available
mpiControl**FlashConfigGet**	Get Control flash config
mpiControl**FlashConfigSet**	Set Control flash config
meiControl**FPGADefaultGet**	Creates a default FPGA filename
meiControl**FPGAFileOverride**	Replace the default FPGA file
meiControl**GateGet**	Get the closed state (TRUE or FALSE)
meiControl**GateSet**	Set the closed state (TRUE or FALSE)
meiControl**Info**	Retrieve information about an MEI motion controller
meiControl**SampleCounter**	Write the number of servo cycles (samples)
meiControl**SampleRate**	Write current sample rate (Hz) of the controller's processor
meiControl**SamplestoSeconds**	Converts samples to seconds
meiControl**SampleWait**	Wait for count samples

meiControl**SecondstoSamples**	Converts seconds to samples
meiControl**StatisticsReset**	
mpiControl**Type**	Get type of Control object (used to create Command object)

I/O Methods

mpiControl**DigitalIn**	Reads the current input state
mpiControl**DigitalOutGet**	Read the current output state
mpiControl**DigitalOutSet**	Change the state of output bit

Event Methods

mpiControl**EventNotifyGet**	Set the control event that will cause the firmware to generate an interrupt
mpiControl**EventNotifySet**	Configure firmware to generate an interrupt
mpiControl**EventReset**	Reset the events

Memory Methods

mpiControl**Memory**	Get address of Control memory
mpiControl**MemoryAlloc**	Allocate bytes of firmware memory
mpiControl**MemoryCount**	Get number of bytes available in firmware
mpiControl**MemoryFree**	Free bytes of firmware memory
mpiControl**MemoryGet**	Copy count bytes of Control memory to application memory
mpiControl**MemorySet**	Copy count bytes of application memory to Control memory
meiControl**MemoryToFile**	

Relational Methods

meiControl**Platform**	Return a handle to the Platform object

Action Methods

mpiControl**CycleWait**	Wait for Control to execute count cycles
mpiControl**Init**	Initialize Control object
mpiControl**InitVerify**	
mpiControl**InterruptEnable**	Enable interrupts to Control object
mpiControl**InterruptWait**	Wait for controller interrupt
mpiControl**InterruptWake**	Wake all threads waiting for controller interrupt
meiControl**RecorderCancel**	Cancel the reservation for an abandoned recorder.
meiControl**RecorderStatus**	Read the recorder's status.
mpiControl**Reset**	Reset controller hardware
meiControl**SampleWait**	Specify how many samples the host waits for, while the controller executes.
meiControl**VersionMismatchOveride**	

Data Types

MPIControl**Address**	

MPIControl**Config** / MEIControl**Config**
MPIControl**FanStatusFlag**
MPIControl**FanStatusMask**
MEIControl**FPGA**
MEIControl**Info**
MEIControl**InfoDriver**
MEIControl**InfoFirmware**
MEIControl**InfoFirmwareZMP**
MEIControl**InfoIo**
MEIControl**InfoIoDigitalIn**
MEIControl**InfoIoDigitalOut**
MEIControl**InfoHardware**
MEIControl**InfoMpi**
MEIControl**InfoPld**
MEIControl**InfoRincon**
MEIControl**Input**
MEIControl**IoBit**
MPIControl**Message** / MEIControl**Message**
MPIControl**MemoryType**
MEIControl**Output**
MPIControl**Status**
MEIControl**Statistics**
MEIControl**Trace**
MPIControl**Type**

Constants

MPIControl**InMAX**
MPIControl**OutMAX**
MPIControl**MAX_AXES**
MPIControl**MAX_COMPENSATORS**
MPIControl**MAX_RECORDERS**
MPIControl**MIN_AXIS_FRAME_COUNT**
MPIControl**MAX_SAMPLE_RATE**
MPIControl**MIN_SAMPLE_RATE**
MPIControl**RECORD_COUNT_DEFAULT**
MEIControl**STRING_MAX**

Macros

mpiControl**FanStatusMaskBIT**

mpiControlCreate

Declaration

```
MPIControl mpiControlCreate(MPIControlType     type,
                            MPIControlAddress  *address)
```

Required Header: stdmpi.h

Description

mpiControlCreate creates a Control object of the specified ***type*** and type-specific ***address***. ControlCreate is the equivalent of a C++ constructor.

The type parameter determines the form of the address parameter:

If the "type" parameter is	*Then the form of the "address" parameter is*
MPIControlTypeDEFAULT	implementation-specific
MPIControlTypeMAPPED	MPIControlAddress.mapped
MPIControlTypeDEVICE	MPIControlAddress.device
MPIControlTypeCLIENT	MPIControlAddress.client

Remarks

This constructor does not reset or initialize the motion control device.

If **MPIControlType** is	And **MPIControlAddress** is	Then the **Board Number** is	And the **"address" parameter** to be used is
DEFAULT	Null address	0 address.number	default address parameter default address parameter
DEVICE	Null address	0 address.number	default device driver address.type.device (if address. type.device is Null, then default device driver)
CLIENT	address	specified by server	address.type.client (**NOTE**: address.number should be set to zero)

1. If the ***type*** is DEFAULT, then the address structure (if supplied) is referenced **only for the board number**. Note that even if the default ***type*** is DEVICE, the default device driver will be used and *address.type.device* will

not be used.

2. If the ***type*** is explicitly DEVICE, and the ***address*** is provided, then address.number will be used. If *address.type.device* is NULL, then the default device driver will be used. If *address.type.device* is not NULL, then the specified driver (DEVICE) will be used.

Return Values	
handle	to a Control object
MPIHandleVOID	if the object could not be created

Sample Code

Example #1
In general, if the caller specifies an explicit type (i.e., not DEFAULT), then the caller must completely fill out the address.type structure.
A simple case that will work for almost anyone who wants to use board #0:

```
mpiControlCreate(MPIControlTypeDEFAULT, NULL);
```

Example #2
A simple case where board #1 is desired is:

```
{
   MPIControl control;
   MPIControlAddress address;

   address.number = 1;
   control = mpiControlCreate(MPIControlTypeDEFAULT, &address);
}
```

Since the default MPIControlType = MPIControlTypeDEVICE, the *address* may be on the stack with garbage for the device driver name. This isn't a problem, however, because the board number is the only field in *address* that will be used when the caller specifies the DEFAULT MPIControlType.

Example #3

```
/* To remotely connect via client / server, you need to run
   server.exe on your PC that has your XMP / ZMP in it.
   If you are using an eXMP, run server on your eXMP.
   This code will connect to that instance of server.exe. */
MPIControl          control;
MPIControlAddress   address;

/* Change this to the ip address of the computer
```

```
   that has your XMP in it. */

address.type.client.server = "10.50.114.200";

/* Port 3300 is the default port for client / server */

address.type.client.port = 3300;

address.number = 0;
control = mpiControlCreate(MPIControlTypeCLIENT, &address);
```

See Also

MPIControl | MPIControlAddress | MPIControlType | mpiControlValidate
mpiControlInit | mpiControlDelete

mpiControlDelete

Declaration

```
long mpiControlDelete(MPIControl control);
```

Required Header: stdmpi.h

Description

mpiControlDelete deletes a control object and invalidates its handle. *ControlDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiControlCreate | mpiControlValidate

mpiControlValidate

Declaration

```
long mpiControlValidate(MPIControl control);
```

Required Header: stdmpi.h

Description

mpiControlValidate validates the control object and its handle.

Return Values	
MPIMessageOK	

See Also

mpiControlCreate | mpiControlDelete

mpiControlAddress

Declaration

```
long mpiControlAddress(MPIControl        control,
                       MPIControlAddress *address)
```

Required Header: stdmpi.h

Description

When a Control object (***control***) is created, an address is used. **mpiControlAddress** writes this address to the contents of ***address***.

Return Values	
MPIMessageOK	

See Also

mpiControlConfigGet

Declaration

```
long mpiControlConfigGet(MPIControl       control,
                         MPIControlConfig *config,
                         void             *external)
```

Required Header: stdmpi.h

Description

mpiControlConfigGet gets the configuration of a Control object (***control***) and writes it into the structure pointed to by **config**, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

Return Values	
MPIMessageOK	

Sample Code

```
/*
  Write a value to element index of the user buffer.
  Make sure to save topology to flash before doing this.
*/
void write2UserBuffer(MPIControl control, long value, long index)
{
    MPIControlConfig config;
       MEIControlConfig external;
       long returnValue;

    if((index < MEIXmpUserDataSize) && (index >= 0))
    {
      /* Make sure to save topology to flash before doing this */
      returnValue = mpiControlConfigGet(control,
           &config,
           &external);
```

```
        msgCHECK(returnValue);

        external.UserBuffer.Data[index] = value;

        returnValue = mpiControlConfigSet(control,
             &config,
             &external);
        msgCHECK(returnValue);
       }
}
```

See Also

mpiControlConfigSet | MEIControlConfig | Dynamic Allocation of External Memory Buffers

mpiControlConfigSet

Declaration

```
long mpiControlConfigSet(MPIControl       control,
                         MPIControlConfig *config,
                         void             *external)
```

Required Header: stdmpi.h

Description

mpiControlConfigSet sets (writes) the Control object's (***control***) configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

WARNING: mpiControlConfigSet(...) is a controller-wide routine that will clear other controller object configurations and potentially force a reset of the SynqNet network. This method should be executed in your application before configuring any other MPI objects. For information regarding which configurations force a reset of the SynqNet network, please see the Warning message for MPIControlConfig / MEIControlConfig.

Remarks

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

Return Values	
MPIMessageOK	
MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE	

Sample Code

```
/*
  Write a value to element index of the user buffer.
  Make sure to save topology to flash before doing this.
*/
void write2UserBuffer(MPIControl control, long value, long index)
{
    MPIControlConfig config;
       MEIControlConfig external;
       long returnValue;

    if((index < MEIXmpUserDataSize) && (index >= 0))
    {
      /* Make sure to save topology to flash before doing this */
      returnValue = mpiControlConfigGet(control,
           &config,
           &external);
      msgCHECK(returnValue);

      external.UserBuffer.Data[index] = value;

      returnValue = mpiControlConfigSet(control,
           &config,
           &external);
      msgCHECK(returnValue);
     }
}
```

See Also

mpiControlConfigGet | MEIControlConfig | Dynamic Allocation of External Memory Buffers

mpiControlDigitalIn

Declaration

```
long  mpiControlDigitalIn(MPIControl     control,
                          long           bitStart,
                          long           bitCount,
                          unsigned long  *state);
```

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

Description

mpiControlDigitalIn reads the current input state of one or more controller inputs.

control	a handle to the Control object
bitStart	the first controller input bit that will be returned by the function.
bitCount	the number of controller bits that will be returned by the function.
***state**	the address of the current state of the inputs that is returned.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to get the state of controller input 1.

```
unsigned long input1;
mpiControlDigitalIn( control, 1, 1, &input1 );
```

See Also

Controller I/O | mpiControlDigitalOutSet | mpiControlDigitalOutGet

mpiControlDigitalOutGet

Declaration

```
long  mpiControlDigitalOutGet(MPIControl     control,
                              long           bitStart,
                              long           bitCount,
                              unsigned long  *state);
```

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

Description

mpiControlDigitalOutGet function reads the current output state of one or more controller output bits.

control	a handle to the Control object
bitStart	the first controller output bit that will be returned by the function.
bitCount	the number of controller output bits that will be returned by the function.
***state**	the address of the current state of the controller output bits that will be returned.

Return Values	
MPIMessageOK	

Sample Code

The following code shows how to get the state of controller input 1.

```
unsigned long output1;
mpiControlDigitalOutGet( control, 1, 1, &output1 );
```

See Also

Controller I/O | mpiControlDigitalOutSet | mpiControlDigitalIn

mpiControlDigitalOutSet

Declaration

```
long  mpiControlDigitalOutSet(MPIControl     control,
                              long           bitStart,
                              long           bitCount,
                              unsigned long  state,
                              MPI_BOOL       wait);
```

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

Description

mpiControlDigitalOutSet function changes the state of one or more general purpose bits.

control	a handle to the Control object
bitStart	the first controller output bit that will be returned by the function.
bitCount	the number of controller output bits that will be returned by the function.
state	the new state of the general purpose bits that will be returned.
wait	a Boolean flag indicating if the new output state is applied immediately or a wait is inserted so that any previously set output is transmitted to the controller hardware before applying the new output state.

Return Values	
MPIMessageOK	

Sample Code

The next piece of code shows how to set three controller outputs (2, 3 and 4):

```
mpiControlDigitalOutSet( control, 2, 3, 7 );
```

See Also

Controller I/O | mpiControlDigitalIn | mpiControlDigitalOutGet

meiControlExtMemAvail

Declaration

```
long meiControlExtMemAvail(MPIControl control,
                           long       *size)
```

Required Header: stdmei.h

Description

meiControlExtMemAvail gets the amount of external memory available on an XMP-Series controller. It puts the number of words (8 bit) in the location pointed to by size. Since the XMP is a 32 bit controller, the number of 32 bit words available is equal to the value of size divided by 4. The value of size is useful for setting things that use the external memory, such as the Recorder.

control	a handle to the Control object
***size**	a pointer to the available memory words returned by the method

Return Values	
MPIMessageOK	

Sample Code

Example:

```
/* Prints the size of the available external memory size */
void printExternalMemorySize(MPIControl control)
{
    long returnValue;
    long size;

    returnValue = meiControlExtMemAvail(control, &size);

    msgCHECK(returnValue);

    printf("size %d (8 bit), %d (32 bit)", size, size / 4);
}
```

Output:

```
C:\out\extmemavail\Debug>extmemavail
size 238008 (8 bit), 59502 (32 bit)
```

See Also

MPIControlConfig

mpiControlFlashConfigGet

Declaration

```
long mpiControlFlashConfigGet(MPIControl        control,
                              void              *flash,
                              MPIControlConfig  *config,
                              void              *external)
```

Required Header: stdmpi.h

Description

mpiControlFlashConfigGet gets the flash configuration of a Control object (control) and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Control's flash configuration information in ***external*** is in addition to the Control'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).

Remarks

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

control	a handle to a Control 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 control object of type MPIControlConfig.
***external**	a pointer to a configuration structure for the control object of type MEIControlConfig.

Return Values	
MPIMessageOK	

Sample Code

```
/*
  Write a value to element index of the user buffer.
  Make sure to save topology to flash before doing this.
*/
void write2UserBufferFlash(MPIControl control, long value, long index)
{
  MPIControlConfig config;
     MEIControlConfig external;
     long returnValue;

     if((index < MEIXmpUserDataSize) && (index >= 0))
     {
         /* Make sure to save topology to flash before doing this */
         returnValue = mpiControlFlashConfigGet(control,
               MPIHandleVOID,
               &config,
               &external);
         msgCHECK(returnValue);

         external.UserBuffer.Data[index] = value;

         returnValue = mpiControlFlashConfigSet(control,
               MPIHandleVOID,
               &config,
               &external);
         msgCHECK(returnValue);
     }
}
```

See Also

MEIFlash | mpiControlFlashConfigSet | | MEIControlConfig

mpiControlFlashConfigSet

Declaration

```
long mpiControlFlashConfigSet(MPIControl       control,
                              void             *flash,
                              MPIControlConfig *config,
                              void             *external)
```

Required Header: stdmpi.h

Description

mpiControlFlashConfigSet sets (writes) the flash configuration of a Control object (***control***), 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 Control's flash configuration information in ***external*** is in addition to the Control'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).

Remarks

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

control	a handle to a Control 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 control object of type MPIControlConfig.
***external**	a pointer to a configuration structure for the control object of type MEIControlConfig.

Return Values	
MPIMessageOK	

Sample Code

```
/*
  Write a value to element index of the user buffer.
  Make sure to save topology to flash before doing this.
*/
void write2UserBufferFlash(MPIControl control, long value, long index)
{
  MPIControlConfig config;
     MEIControlConfig external;
     long returnValue;

     if((index < MEIXmpUserDataSize) && (index >= 0))
     {
         /* Make sure to save topology to flash before doing this */
         returnValue = mpiControlFlashConfigGet(control,
               MPIHandleVOID,
               &config,
               &external);
         msgCHECK(returnValue);

         external.UserBuffer.Data[index] = value;

         returnValue = mpiControlFlashConfigSet(control,
               MPIHandleVOID,
               &config,
               &external);
         msgCHECK(returnValue);
     }
}
```

See Also

MEIFlash | mpiControlFlashConfigGet | | MEIControlConfig

meiControlFPGADefaultGet

Declaration

```
long meiFPGADefaultGet(MPIControl              control,
                       MEIPlatformSocketInfo   *socketInfo,
                       MEIControlFPGA          *fpga)
```

Required Header: stdmei.h

Description

meiControlFPGADefaultGet creates a default FPGA filename based on the ***socketInfo***.

control	a handle to the Control object
***socketInfo**	tells the function which type of FPGA is physically on the board.
***fpga**	a pointer to a MEIControlFPGA object that contains a string that is the filename. To get the proper ***fpga***, pass in ***control*** and valid ***socketInfo***.

Return Values	
MPIMessageOK	

See Also

meiControlFPGADefaultOverride

Declaration

```
long meiFPGADefaultOverride(MPIControl              control,
                            MEIControlFPGA          *fpga,
                            const char              *overrideFile,
                            MEIPlatformSocketInfo   *socketInfo)
```

Required Header: stdmei.h

Description

meiControlFPGADefaultOverride checks to see if the ***socketInfo*** fits the board's physical configuration. If so, the FPGA filename is replaced with the ***overrideFile***. This allows the user to specify FPGA files instead of using the MPI's default FPGA file.

control	a handle to the Control object.
***fpga**	a pointer to MEIControlFPGA struct that contains the current file name string.
***overrideFile**	is a character string that contains a desired filename.
***socketInfo**	is a pointer to valid socket information.

Return Values	
MPIMessageOK	

See Also

meiControlGateGet

Declaration

```
long meiControlGateGet(MPIControl   control,
                       long         gate,
                       MPI_BOOL     *closed)
```

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

Description

meiControlGateGet gets the closed state (TRUE or FALSE) from the specified control gate (0 to 31).

Return Values	
MPIMessageOK	

See Also

meiControlGateSet

meiControlGateSet

Declaration

```
long meiControlGateSet(MPIControl   control,
                       long         gate,
                       MPI_BOOL     closed)
```

Required Header: stdmei.h

Change History: Modified in the 03.03.00

Description

meiControlGateSet sets the closed state (TRUE or FALSE) for the specified control gate (0 to 31).

Return Values	
MPIMessageOK	

See Also

meiControlGateGet

meiControlInfo

Declaration

```
long  meiControlInfo(MPIControl       control,
                     MEIControlInfo   *info);
```

Required Header: stdmei.h

Description

meiControlInfo retrieves information about an MEI motion controller.

control	a handle to the Control object
***info**	a pointer to MEIControlInfo that gets completed with the appropriate controller information.

Return Values	
MPIMessageOK	
MPIHandleVOID	if ***control*** is invalid

See Also

meiControlSampleCounter

Declaration

```
long meiControlSampleCounter(MPIControl   control,
                             long         *sampleCounter)
```

Required Header: stdmei.h

Description

meiControlSampleCounter writes the number of servo cycles (samples) that have occurred since the last sample counter reset/rollover, to the ***sampleCounter***. When the user resets the controller, the sample counter will also be reset. Since the sample counter is a long, if the sample counter is 2147483647 it will roll over on the next servo cycle to -2147483648.

Return Values	
MPIMessageOK	

See Also

meiControlSecondstoSamples | meiControlSamplestoSeconds | meiControlSampleWait

meiControlSampleRate

Declaration

```
long meiControlSampleRate(MPIControl  control,
                          double      *sampleRate)
```

Required Header: stdmei.h

Description

meiControlSampleRate writes the current sample rate (Hz) of the controller's processor to the address pointed to by ***sampleRate***. This is the same value returned in MPIControlConfig.sampleRate after mpiControlConfigGet(...) has been performed, but is also provided as this separate method to avoid the extra processing overhead of mpiControlConfigGet.

control	a handle to the Control object
***sampleRate**	pointer to a double where the current sample rate will be stored.

Return Values	
MPIMessageOK	
MEIControlMessageFIRMWARE_VERSION_NONE	
MEIControlMessageFIRMWARE_VERSION	

See Also

mpiControlConfigGet | MPIControlConfig | MEIControlMessage

meiControlSamplesToSeconds

Declaration

```
long meiControlSamplesToSeconds(MPIControl control,
                                long       samples,
                                float      *seconds)
```

Required Header: stdmei.h

Description

meiControlSamplesToSeconds writes to seconds the number of seconds it takes to process samples number of ***samples*** (at the current sample rate). Use this function to convert samples to ***seconds***.

Return Values	
MPIMessageOK	

See Also

meiControlSecondstoSamples | meiControlSampleCounter

meiControlSampleWait

Declaration

```
long meiControlSampleWait(MPIControl control,
                          long       count)
```

Required Header: stdmei.h

Description

meiControlSampleWait waits for ***count*** samples while the motion controller (associated with ***control***) executes. While the host waits, the host gives up its time slice and continuously verifies that the controller firmware is operational.

Return Values	
MPIMessageOK	

See Also

meiControlSamplestoSeconds | meiControlSecondstoSamples | meiControlSampleCounter

meiControlSecondsToSamples

Declaration

```
long meiControlSecondsToSamples(MPIControl control,
                                float      seconds,
                                long       *samples)
```

Required Header: stdmei.h

Description

meiControlSecondsToSamples writes to samples the number of servo cycles that will take place in seconds number of ***seconds*** (at the current sample rate). Use this function to convert seconds to ***samples***.

Return Values	
MPIMessageOK	

See Also

meiControlSamplestoSeconds | meiControlSampleCounter | meiControlSampleWait

meiControlStatisticsReset

Declaration

```
long  meiControlStatisticsReset(MPIControl   control);
```

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

Description

meiControlStatisticsReset resets the controller's maxBackgroundTime and maxForegroundTime values, and recalculates the controller's statistics.

control	a handle to the Control object.

Return Values	
MPIMessageOK	

See Also

meiControlStatistics | MEIControlStatistics

mpiControlType

Declaration

```
long mpiControlType(MPIControl     control,
                    MPIControlType *type)
```

Required Header: stdmpi.h

Description

When a Control object (***control***) is created, a type is used. **mpiControlType** writes this type to the contents of ***type***.

Return Values	
MPIMessageOK	

See Also

mpiControlEventNotifyGet

Declaration

```
long  mpiControlEventNotifyGet(MPIControl     control,
                               MPIEventMask   *eventMask,
                               void           *external);
```

Required Header: stdmpi.h
Change History: Added in the 03.02.00

Description

mpiControlEventNotifyGet fills in the ***eventMask*** with the data indicating which control events will cause the firmware to generate an interrupt. If ***external*** is not NULL (it should be a pointer to a user supplied MEIEventNotifyData structure), then the function will fill out the structure with data from the firmware's control object.

control	a handle to the Control object
***eventMask**	pointer to MPIEventMask structure.
***external**	pointer to MEIEventNotifyData structure or NULL.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiControlEventNotifySet | MEIEventNotifyData

mpiControlEventNotifySet

Declaration

```
long  mpiControlEventNotifySet(MPIControl     control,
                               MPIEventMask   eventMask,
                               void           *external);
```

Required Header: stdmpi.h
Change History: Added in the 03.02.00

Description

mpiControEventNotifySet configures the firmware to generate interrupts based on the control events indicated in the ***eventMask***. If ***external*** is not NULL (it should be a pointer to a user supplied MEIEventNotifyData structure), then the data in the structure is written to the firmware's control object.

control	a handle to the Control object
eventMask	MPIEventMask structure.
***external**	pointer to MEIEventNotifyData structure or NULL.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiControlEventNotifyGet | MEIEventNotifyData

mpiControlEventReset

Declaration

```
long  mpiControlEventReset(MPIControl     control,
                           MPIEventMask   eventMask);
```

Required Header: stdmpi.h
Change History: Added in the 03.02.00

Description

mpiControlEventReset resets (clears) the events indicated in the ***eventMask*** from the firmware's ***control*** object. Once cleared, the events can cause the firmware to generate an interrupt.

control	a handle to the Control object
eventMask	MPIEventMask structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiControlEventNotifyGet | mpiControlEventNotifySet | MPIEventMask | mpiMotionEventReset | mpiMotorEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | meiSqNodeEventReset | mpiAxisEventReset

mpiControlMemory

Declaration

```
long mpiControlMemory(MPIControl control,
                      void       **memory,
                      void       **external)
```

Required Header: stdmpi.h

Description

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

If ***external*** is not NULL, the contents of ***external*** are set to an implementation-specific address that typically points to a different section or type of Control memory other than ***memory*** (e.g., to external or off-chip memory). These addresses (or addresses calculated from them) are passed as the src argument to mpiControlMemoryGet(...) and the dst argument to mpiControlMemorySet(...).

Return Values	
MPIMessageOK	

Sample Code

```
/* Simple code to increment userbuffer[0] */
   MEIXmpData       *firmware;
   MEIXmpBufferData *buffer;

   long returnValue, tempBuffer;

   /* Get memory pointers */
   returnValue =
      mpiControlMemory(control,
      &firmware,
      &buffer);
   msgCHECK(returnValue);

   returnValue = mpiControlMemoryGet(control,
      &tempBuffer,
      &buffer->UserBuffer.Data[0],
      sizeof(buffer->UserBuffer.Data[0]));
   msgCHECK(returnValue);
```

```
   tempBuffer++;

   returnValue = mpiControlMemorySet(control,
      &buffer->UserBuffer.Data[0],
      &tempBuffer,
      sizeof(buffer->UserBuffer.Data[0]));
   msgCHECK(returnValue);
```

See Also

mpiControlMemoryGet | mpiControlMemorySet | mpiControlMemoryAlloc | mpiControlMemoryCount | mpiControlMemoryFree

mpiControlMemoryAlloc

Declaration

```
long mpiControlMemoryAlloc(MPIControl           control,
                           MPIControlMemoryType type,
                           long                 count,
                           void                 **memory)
```

Required Header: stdmpi.h

Description

mpiControlMemoryAlloc allocates ***count*** bytes of firmware memory [of type ***type*** on a Control object (***control***)] and writes the host address (of the allocated firmware memory) to the location pointed to by ***memory***.

Return Values	
MPIMessageOK	

See Also

mpiControlMemoryGet | mpiControlMemorySet | mpiControlMemory | mpiControlMemoryCount | mpiControlMemoryFree

mpiControlMemoryCount

Declaration

```
long mpiControlMemoryCount(MPIControl           control,
                           MPIControlMemoryType type,
                           long                 *count)
```

Required Header: stdmpi.h

Description

mpiControlMemoryCount writes the number of bytes of firmware memory [on a Control object (***control***, of type ***type***) that are available to be allocated] to the location pointed to by ***count***.

Return Values	
MPIMessageOK	

See Also

mpiControlMemoryFree

Declaration

```
long mpiControlMemoryFree(MPIControl           control,
                          MPIControlMemoryType type,
                          long                 count,
                          void                 *memory)
```

Required Header: stdmpi.h

Description

mpiControlMemoryFree frees ***count*** bytes of firmware memory on a Control object (***control***, of type ***type***) starting at host address ***memory***.

Return Values	
MPIMessageOK	

See Also

mpiControlMemoryGet | mpiControlMemorySet | mpiControlMemoryAlloc | mpiControlMemoryCount | mpiControlMemory

mpiControlMemoryGet

Declaration

```
long mpiControlMemoryGet(MPIControl  control,
                         void         *dst,
                         const  void  *src,
                         long         count)
```

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

Description

mpiControlMemoryGet gets ***count*** bytes of ***control*** memory (starting at address ***src***) and puts (writes) them in application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

Sample Code

```
/* Simple code to increment userbuffer[0] */
  MEIXmpData        *firmware;
  MEIXmpBufferData  *buffer;

  long returnValue, tempBuffer;

  /* Get memory pointers */
  returnValue =
      mpiControlMemory(control,
      &firmware,
      &buffer);
  msgCHECK(returnValue);

  returnValue = mpiControlMemoryGet(control,
      &tempBuffer,
      &buffer->UserBuffer.Data[0],
      sizeof(buffer->UserBuffer.Data[0]));
  msgCHECK(returnValue);

  tempBuffer++;

  returnValue = mpiControlMemorySet(control,
```

```
        &buffer->UserBuffer.Data[0],
        &tempBuffer,
        sizeof(buffer->UserBuffer.Data[0]));
    msgCHECK(returnValue);
```

See Also

mpiControlMemorySet | mpiControlMemory | mpiControlMemoryAlloc | mpiControlMemoryCount | mpiControlMemoryFree

mpiControlMemorySet

Declaration

```
long mpiControlMemorySet(MPIControl   control,
                         void         *dst,
                         const  void  *src,
                         long         count)
```

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

Description

mpiControlMemorySet sets (writes) ***count*** bytes of application memory (starting at address ***src***) to ***control*** memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

Sample Code

```
/* Simple code to increment userbuffer[0] */
  MEIXmpData    *firmware;
  MEIXmpBufferData *buffer;

  long returnValue, tempBuffer;

  /* Get memory pointers */
  returnValue =
      mpiControlMemory(control,
      &firmware,
      &buffer);
  msgCHECK(returnValue);

  returnValue = mpiControlMemoryGet(control,
      &tempBuffer,
      &buffer->UserBuffer.Data[0],
      sizeof(buffer->UserBuffer.Data[0]));
  msgCHECK(returnValue);

  tempBuffer++;

  returnValue = mpiControlMemorySet(control,
```

```
        &buffer->UserBuffer.Data[0],
        &tempBuffer,
        sizeof(buffer->UserBuffer.Data[0]));
    msgCHECK(returnValue);
```

See Also

mpiControlMemoryGet | mpiControlMemory | mpiControlMemoryAlloc | mpiControlMemoryCount | mpiControlMemoryFree

meiControlMemoryToFile

Declaration

```
long  meiControlMemoryToFile(MPIControl    control,
                             const  char   *fileName);
```

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

Description

mpiControlMemoryToFile creates a file with a copy of the current controller memory. The contents of this file may then later be viewed using the VM3 utility. This is often useful for helping troubleshoot a problem that is difficult to understand.

control	a handle to the Control object.
***filename**	The name of the file to be created which will hold the contents of controller memory.

Return Values	
MPIMessageOK	

Sample Code

```
long result =
    meiControlMemoryToFile(control, “controllerMemory.mem”);
msgCHECK(result);
```

See Also

meiControlPlatform

Declaration

```
MEIPlatform  meiControlPlatform(MPIControl    control)
```

Required Header: stdmei.h

Description

meiControlPlatform returns a handle to the Platform object with which the control is associated.

control	a handle to the Control object

Return Values	
MPIPlatform	handle to a Platform object
MPIHandleVOID	if ***control*** is invalid

See Also

mpiControlCreate

meiControlCycleWait

Declaration

```
long meiControlCycleWait(MPIControl control,
                         long       count)
```

Required Header: stdmei.h

Description

meiControlCycleWait waits for the XMP motion controller (***control***) to execute for count background cycles. The host will continuously verify that the XMP firmware is operational, and the host will give up its time slice as it waits (for the controller to execute the background cycles).

Return Values	
MPIMessageOK	

See Also

mpiControlInit / meiControlInit

Declaration: mpiControlInit

```
long mpiControlInit(MPIControl   control)
```

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

Description

mpiControlInit initializes the control object. ControlInit must be called after mpiControlCreate(...) and before any other MPI calls in your application. ControlInit establishes communication with the motion controller hardware and initializes any SynqNet networks connected to the controller. Controller communication can occur through direct memory access, device driver, or remote via client/server.

control	a handle to the Control object

Return Values	
MPIMessageOK	
MPIControlMessageLIBRARY_VERSION	
MPIControlMessageADDRESS_INVALID	
MPIControlMessageCONTROL_INVALID	
MPIControlMessageTYPE_INVALID	
MPIControlMessageCONTROL_NUMBER_INVALID	
MEIControlMessageFIRMWARE_INVALID	
MEIControlMessageSYNQNET_STATE	
MEIPacketMessageADDRESS_INVALID	
MEIPlatformMessageDEVICE_INVALID	
MEIPlatformMessageDEVICE_MAP_ERROR	
MEISynqNetMessageSTATE_ERROR	
MEISynqNetMessageTOPOLOGY_MISMATCH	
MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH	

MEISynqNetMessageNODE_LATENCY_EXCEEDED	
MEISynqNetMessageNODE_FPGA_VERSION	
MEISynqNetMessageNODE_MAC_VERSION	
MEISynqNetMessageNODE_INIT_FAIL	

Sample Code

```
MPIControl   control; /* motion controller object handle */

long         result;

control =
   mpiControlCreate(MPIControlTypeDEFAULT, NULL);
result =
   mpiControlValidate(control);
msgCHECK(result);

/* Initialize motion controller */
returnValue =
   mpiControlInit(control);
msgCHECK(result);
```

Declaration: meiControlInit

```
long meiControlInit(MPIControl   control
                    const  char  *mpiVersion)
```

Required Header: stdmpi.h

Description

meiControlInit

control	a handle to the Control object.
mpiVersion	Should always be **MPI_INTERFACE_VERSION.**

Return Values	
MPIMessageOK	
MPIControlMessageLIBRARY_VERSION	
MPIControlMessageADDRESS_INVALID	
MPIControlMessageCONTROL_INVALID	
MPIControlMessageTYPE_INVALID	
MPIControlMessageCONTROL_NUMBER_INVALID	
MEIControlMessageFIRMWARE_INVALID	
MEIControlMessageSYNQNET_STATE	
MEIPacketMessageADDRESS_INVALID	
MEIPlatformMessageDEVICE_INVALID	
MEIPlatformMessageDEVICE_MAP_ERROR	
MEISynqNetMessageSTATE_ERROR	

Sample Code

```
MPIControl   control; /* motion controller object handle */

long         result;

control =
   mpiControlCreate(MPIControlTypeDEFAULT, NULL);
result =
   mpiControlValidate(control);
msgCHECK(result);

/* Initialize motion controller, but not SynqNet */
returnValue =
   meiControlInit(control, MPI_INTERFACE_VERSION);
msgCHECK(result);
```

See Also

mpiControlCreate | mpiControlDelete | MPI_INTERFACE_VERSION

mpiControlInitVerify

Declaration

```
long mpiControlInitVerify(MPIControl   control,
                          long         methodVersion,
                          ...);
```

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

Description

mpiControlInitVerify ...

control	a handle to a Control object
methodVersion	

Return Values	
MPIMessageOK	

See Also

mpiControlInit

mpiControlInterruptEnable

Declaration

```
long mpiControlInterruptEnable(MPIControl control,
                               long       enable)
```

Required Header: stdmpi.h

Description

If "enable" is **TRUE**, then **mpiControlInterruptEnable** enables interrupts from the motion controller.

If "enable" is **FALSE**, then **mpiControlInterruptEnable** disables interrupts from the motion controller.

Return Values	
MPIMessageOK	

See Also

mpiControlInteruptWait | mpiControlInteruptWake

mpiControlInterruptWait

Declaration

```
long mpiControlInterruptWait(MPIControl  control,
                             MPI_BOOL    *interrupted,
                             MPIWait     timeout)
```

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

Description

mpiControlInterruptWait waits for an interrupt from the motion controller if interrupts are enabled. After the ControlInterruptWait method returns, if the location pointed to by ***interrupted*** contains **TRUE**, then an interrupt has occurred. After the ControlInterruptWait method returns, if the location pointed to by ***interrupted*** contains **FALSE**, then no interrupt has occurred, and the return of ControlInterruptWait was caused either by a call to **mpiControlInterruptWake(...)**.

If ***timeout*** is **MPIWaitPOLL (0)**, then *ControlInterruptWait* will return immediately.

If ***timeout*** is **MPIWaitFOREVER (-1)**, then *ControlInterruptWait* will wait forever for an interrupt.

Otherwise, *ControlInterruptWait* will wait ***timeout*** milliseconds for an interrupt.

NOTE: For Windows operating systems, only **MPIWaitPOLL** and **MPIWaitFOREVER** are valid timeout values.

Return Values	
MPIMessageOK	
MPIMessageTIMEOUT	

See Also

mpiControlInterruptWake | mpiControlInteruptEnable

mpiControlInterruptWake

Declaration

```
long mpiControlInterruptWake(MPIControl control)
```

Required Header: stdmpi.h

Description

mpiControlInterruptWake wakes all threads waiting for an interrupt from the motion controller ***control*** [as a result of a call to mpiControlInterruptWait(...)]. The waking thread(s) will return from the call with no interrupt indicated.

Return Values	
MPIMessageOK	

See Also

mpiControlInterruptWait | mpiControlInteruptEnable

meiControlRecorderCancel

Declaration

```
long  meiControlRecorderCancel(MPIControl   control,
                               long         recorderNumber);
```

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

Description

meiControlRecorderCancel provides a way to cancel the reservation for an abandoned recorder.

It is possible for a fatal error to occur in your application where mpiRecorderDelete(...) is not called, which will leave your recorders abandoned.

An abandoned recorder number cannot be reused until the recorder's reservation is canceled or the reservation is explicitly overwritten by specifying the recorder number (i.e. a number other than -1) when calling mpiRecorderCreate(...).

Use meiControlRecorderStatus(...) to make sure you have no reason to believe a recorder is being used before canceling the recorders reservation.

control	a handle to the Control object.
recorderNumber	the index of the abandoned recorder object.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	If *recorderNumber* is less than 0 or greater than MPIRecorderRECORDERS_MAX

See Also

mpiRecorderStatus | mpiRecorderCreate | mpiRecorderDelete | meiControlRecorderStatus

recorderinuse.c

meiControlRecorderStatus

Declaration

```
long  meiControlRecorderStatus(MPIControl          control,
                               long                recorderNumber,
                               MPIRecorderStatus   *status);
```

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

Description

meiControlRecorderStatus allows the recorder status to be read without actually creating a recorder object.

This is useful to help determine whether or not a recorder is abandoned. An abandoned recorder will usually not be running, yet will still be reserved. Another hint that a recorder is abandoned, is an enabled recorder that is full. This sort of behavior is what happens when a program crashes that has a recorder operating.

control	a handle to the Control object.
recorderNumber	the index of the recorder object.
***status**	a pointer to the recorder's status structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	If *recorderNumber* is less than 0, greater than MPIRecorderRECORDERS_MAX, or if *status == NULL.

See Also

meiControlRecorderCancel | mpiRecorderDelete | mpiRecorderCreate

recorderinuse.c

mpiControlReset

Declaration

```
long mpiControlReset(MPIControl control)
```

Required Header: stdmpi.h

Description

mpiControlReset resets the motion controller (***control***) board.

Return Values	
MPIMessageOK	

See Also

mpiControlVersionMismatchOveride

Declaration

```
long meiControlVersionMismatchOveride(MPIControl control);
```

Required Header: stdmei.h

Description

mpiControlValidate overrides the version mismatch between the MPI and the controller.

This function is reserved for MEI use only and should not be used by a customer.

Return Values	
MPIMessageOK	

See Also

MPIControlAddress

Definition

```
typedef struct MPIControlAddress {
    long    number;   /* controller number */

    union {
        void                    *mapped;  /* memory address */
        unsigned long           ioPort;   /* I/O port number */
        const  char             *device;  /* device driver name */
        struct {
            const  char         *name;    /* image file name */
            MPIControlFileType  type;     /* image file type */
        } file;
        struct {
            const  char         *server;
                                /* IP address: host.domain.com */
            long                port;     /* socket number */
        } client;
    } type;
} MPIControlAddress;
```

Change History: Modified in the 03.03.00

Description

MPIControlAddress is a structure that specifies the location of the controller that to be accessed when mpiControlCreate(...) is called. Please refer to the documentation for mpiControlCreate(...) to see how to use this structure.

number	The controller number in the computer
type	A union that holds information about controllers on non-local computers.

See Also

MPIControl | MPIControlType | mpiControlCreate

MPIControlConfig / MEIControlConfig

Definition: MPIControlConfig

```
typedef struct MPIControlConfig {
    long   axisCount;
    long   axisFrameCount[MPIControlMAX_AXES];
    long   captureCount;
    long   compareCount;
    long   compensatorCount;
    long   compensatorPointCount[MPIControlMAX_COMPENSATORS];
    long   filterCount;
    long   motionCount;
    long   motorCount;
    long   recorderCount;
    long   recordCount[MPIControlMAX_RECORDERS];
    long   sequenceCount;
    long   userVersion;
    long   sampleRate;

} MPIControlConfig;
```

Change History: Modified in the 03.04.00.

Description

MPIControlConfig is a structure that specifies the controller configurations. It allocates the number of resources and configurations for the controller's operation. The controller's performance is inversely related to the DSP's load. The controller configuration structure allows the user to disable/enable objects for optimum performance.

WARNING!
mpiControlConfigSet(...) should ONLY be called during application initialization and NOT during motion. If the sampleRate or TxTime is changed, the SynqNet network will be shutdown and re-initialized with the new sampleRate and/or TxTime. If the axisCount, axisFrameCount[], compensatorCount, compensatorPointCount[], recorderCount, or recordCount[] is changed, then the controller's dynamic memory will be cleared and re-allocated with the new configuration. During the re-allocation, compensators, recorders, and axes are not available for application use.

axisCount	Number of axis objects enabled for the controller. The controller's axis object handles the trajectory calculations for command position. For simple systems, set the *axisCount* equal to the *motorCount*.
axisFrameCount	An array containing the number of frames for each axis frame buffer. Each frame is the size of MEIXmpFrame{}. The controller's frame buffers are dynamically allocated by changing the axisFrameCount[]. A larger frame buffer may be required for long multi-point or cam motion profiles. Frame buffer counts must be a power of 2 in size (i.e. 128, 256, ...). Axes mapped to the same motion object MUST have the same frame buffer size. The default axis frame buffer size is 128. The valid range is from 128 to the available memory. Use meiControlExtMemAvail(...) to determine the controller's available memory. Be sure to leave some free memory for potential future features.
captureCount	Number of capture objects enabled for the controller. The controller supports up to 32 captures. The controller's capture object manages the hardware resources to latch a motor's position feedback, triggered by a motor's input.
compareCount	Number of compare objects enabled for the controller. The controller's compare object manages the hardware resources to trigger a motor's output, triggered by a comparison between the motor's feedback and a pre-loaded position value.
compensatorCount	This value defines the number of enabled compensators.
compensatorPointCount	The number of points in the compensation table for each compensator. Compensator tables get allocated on a per-compensator basis. Each compensator can have a different compensation table size as specified by the compensatorPointCount[n] value. See Determining Required Compensator Table Size for more information. An array of the number of points in the compensation table for each compensator. Each point is 32bits. The controller's compensation tables are dynamically allocated by changing the *compensatorPointCount*. When using compensator objects, see Determining Required Compensator Table Size for more information on a proper value for the point count.
filterCount	Number of filter objects enabled for a controller. The filter object handles the closed-loop servo calculations to control the motor. For simple systems, set the filterCount equal to the motorCount.
motionCount	Number of motion supervisor obejcts enabled for a controller. The controller's motion supervisor handles coordination of motion and events for an axis or group of axes. For simple systems, set the motionCount equal to the axisCount.

motorCount	Number of motor objects enabled for a controller. The controller's motor object handles the interface to the servo or stepper drive, dedicated I/O and general purpose motor related I/O. For simple systems, the motorCount should equal the number of physical motors connected to the controller (either directly or via SynqNet).
recorderCount	Number of data recorder objects enabled for a controller. The controller's recorder object handles collecting and buffering any data in controller memory. The enabled data recorders can collect up to a total of 32 addresses each sample. The valid range for the recordCount is 0 to 32.
recordCount	An array of the number of records for each data recorder buffer. Each data record is 32 bits. The controller's data recorder buffers can be dynamically allocated by changing the recordCount. A larger data recorder buffer may be required for higher sample rates, slow host computers, when running via client/server, or when a large number of data fields are being recorded. The valid range is 0 to the available memory. Use meiControlExtMemAvail(...) to determine the controller's available external memory. meiControlExtMemAvail() measures the available memory in 8 bit bytes, so divide the size by 4 to get the number of 32 bit words that the record buffer can be increased by.
sequenceCount	Number of sequence objects enabled for the controller. The controller's sequence object executes and manages a sequence of pre-compiled controller commands.
userVersion	A 32 bit user defined field. The userVersion can be used to mark a firmware image with an identifier. This is useful if multiple controller firmware images are saved to a file.
sampleRate	Number of controller foreground update cycles per second. For SynqNet controllers, this is also the cyclic update rate for the SynqNet network. During the controller's foreground cycle, the axis trajectories are calculated, the filters (closed-loop servo control) are calculated, motion is coordinated, the SynqNet data buffers are updated, and other time critical operations are performed.The default sample rate is 2000 (period = 500 microseconds). The minimum sampleRate for SynqNet systems is 1000 (period = 1 millisecond). The maximum is dependent on the controller hardware and processing load. There are several factors that must be considered to find an appropriate sampleRate for a system. The servo performance, the motion profile accuracy, the SynqNet network cyclic rate, the SynqNet drive update rates, controller background cycle update rate, and controller/application performance. For SynqNet systems, select a sampleRate that is a common multiple of the SynqNet drives connected to the network. For example, if the drive update rate is 8kHz, then appropriate controller sample rates are: 16000, 8000, 5333, 4000, 3200, 2667, 2286, 2000, 1778, 1600, 1455, 1333, 1231, 1067, and 1000

	See Also: Sample Rate SynqNet Controller Performance

Definition: MEIControlConfig

```
typedef struct MEIControlConfig {
    char                userLabel[MEIObjectLabelCharMAX+1];
                           /* +1 for NULL terminator */
    long                preFilterCount;
    long                TxTime;
    long                syncInterruptPeriod;
    MEIPreFilter        preFilter[MEIXmpMAX_PreFilters];
    MEIXmpUserBuffer    UserBuffer;
} MEIControlConfig;
```

Change History: Modified in the 03.04.00. Modified in the 03.03.00. Modified in the 03.02.00.

Description

userLabel	This value consists of 16 characters and is used to label the control object for user identification purposes. The userLabel field is NOT used by the controller.
preFilterCount	This value defines the number of enabled pre-filters.
TxTime	This value determines the controller's transmit time for the SynqNet data. The units are a percentage of the sample period. The default is 75%. Smaller TxTime values will reduce the latency between when the controller receives the data, calculates the outputs, and transmits the data. If the TxTime is too small, the data will be sent before the controller updates the buffer, which will cause a TX_FAILURE event. **See Also:** Sample Rate SynqNet Controller Performance
syncInterruptPeriod	samples/interrupt. Configures the controller to send a hardware interrupt to host computer every *n* controller samples. 0 = disabled, 1 = every sample, 2 = every other sample, etc...
preFilter	This array defines the configuration for each pre-filter.

UserBuffer	This structure defines the controller's user buffer. This is used for custom features that require a controller data buffer.

Sample Code

```
/*
  Write a value to element index of the user buffer.
  Make sure to save topology to flash before doing this.
*/
void write2UserBufferFlash(MPIControl control, long value, long index)
{
  MPIControlConfig config;
     MEIControlConfig external;
     long returnValue;

     if((index < MEIXmpUserDataSize) && (index >= 0))
     {
         /* Make sure to save topology to flash before doing this */
         returnValue = mpiControlFlashConfigGet(control,
               MPIHandleVOID,
               &config,
               &external);
         msgCHECK(returnValue);

         external.UserBuffer.Data[index] = value;

         returnValue = mpiControlFlashConfigSet(control,
               MPIHandleVOID,
               &config,
               &external);
         msgCHECK(returnValue);
     }
}
```

See Also

mpiControlConfigGet | mpiControlConfigSet | meiControlExtMemAvail | MEIPreFilter

Dynamic Allocation of External Memory Buffers

MPIControlFanStatusFlag

Definition

```
typedef enum {
    MPIControlFanStatusFlagSTATUS_NOT_AVAILABLE,   /* 0 */
    MPIControlFanStatusFlagFAN_OK,                 /* 1 */
    MPIControlFanStatusFlagFAN_ERROR,              /* 2 */
    MPIControlFanStatusFlagOVER_TEMP_LIMIT,        /* 3 */
} MPIControlFanStatusFlag;
```

Change History: Added in the 03.02.00

Description

MPIControlFanStatusFlag is an enumeration of fan status bit for use in the MPIControlFanStatusMask. The status bits represent the present status condition(s) for the fan controller on a given Control object.

MPIControlFanStatusFlagSTATUS_NOT_AVAILABLE	Value specifies that the fan status is not available for your controller.
MPIControlFanStatusFlagFAN_OK	Value specifies that the fan is fine.
MPIControlFanStatusFlagFAN_ERROR	Value specifies there is a fan error.
MPIControlFanStatusFlagOVER_TEMP_LIMIT	Value specifies there is an over temperature error.

See Also

MPIControl | MPIControlFanStatus | MPIControlFanStatusMask

MPIControlFanStatusMask

Definition

```
typedef enum {

   MPIControlFanStatusMaskNONE  = 0x0,
   MPIControlFanStatusMaskSTATUS_NOT_AVAILABLE =
      mpiControlFanStatusMaskBIT(MPIControlFanStatusFlagSTATUS_NOT_AVAILABLE),
      /* 0x00000001 */
   MPIControlFanStatusMaskFAN_OK =
      mpiControlFanStatusMaskBIT(MPIControlFanStatusFlagFAN_OK),
      /* 0x00000002 */
   MPIControlFanStatusMaskFAN_ERROR =
      mpiControlFanStatusMaskBIT(MPIControlFanStatusFlagFAN_ERROR),
      /* 0x00000004 */
   MPIControlFanStatusMaskOVER_TEMP_LIMIT =
      mpiControlFanStatusMaskBIT(MPIControlFanStatusFlagOVER_TEMP_LIMIT),
      /* 0x00000008 */
   MPIControlFanStatusMaskALL =
      mpiControlFanStatusMaskBIT(MPIControlFanStatusFlagLAST) - 1
      /* 0x0000000F */

} MPIControlFanStatusMask;
```

Change History: Added in the 03.02.00

Description

MPIControlFanStatusMask is an enumeration of bit masks for the MPIControlFanStatusFlags. The status masks represent the present condition for a Control object.

MPIControlFanStatusMaskNONE	Bit mask containing none of the ControlStatusFlags set.
MPIControlFanStatusMaskSTATUS_NOT_AVAILABLE	Fan status is not available or supported by hardware.
MPIControlFanStatusMaskFAN_OK	Fan status is supported and there are no fan errors or temperature over limits.
MPIControlFanStatusMaskFAN_ERROR	The fan or on-board fan controller has failed. This error indicates a serious problem with the fan or fan controller. This provides an early warning of a possible future over temperature error. If this error occurs, then the fan hardware should be examined and serviced by MEI. Please contact MEI for details. The cause of a FAN_ERROR is hardware dependent. **For ZMP-Series using an ADM1030 fan controller, possible causes are:** ALARM_SPEED,

	FAN_FAULT, and/or REMOTE_DIODE_ERROR Flags are set. Please refer to the ADM1030 specifications for more information.
MPIControlFanStatusMaskOVER_TEMP_LIMIT	The temperature limit has been exceeded. This error indicates the controller processor is too hot. If the controller is operated at excessive temperatures, unknown behavior can result. MEI recommends that the application should be shutdown and the controller should be examined. Excessive temperature could be caused by insufficient air flow or by an improperly operating fan. The cause of an OVER_TEMP_LIMIT is hardware dependent. **For ZMP-Series using an ADM1030 fan controller, possible causes are:** REMOTE_TEMP_HIGH, LOCAL_TEMP_HIGH, and/or OVER_TEMP_LIMIT Flags are set. Please refer to the ADM1030 specifications for more information.
MPIControlFanStatusMaskALL	Bit mask containing all of the ControlStatusFlags set.

See Also

MPIControl | MPIControlFanStatus | MPIControlFanStatusFlag

MEIControlFPGA

Definition

```
typedef struct MEIControlFPGA {
    char  FileName[MEIFlashFileMaxChars]
} MEIControlFPGA;
```

Change History: Modified in the 03.02.00

Description

MEIControlFPGA is a structure containing a ***FileName*** character array. The character array is used to define which FPGA file is to be loaded on the controller. This is usually used internally by the MPI.

Filename	character array

See Also

meiControlFPGADefaultGet | meiControlFPGAFileOverride

MEIControlInfo

Definition

```
typedef struct MEIControlInfo {
    MEIControlInfoMpi        mpi;
    MEIControlInfoFirmware   firmware;
    MEIControlInfoPld        pld;
    MEIControlInfoRincon     rincon;
    MEIControlInfoHardware   hardware;
    MEIControlInfoDriver     driver;
    MEIControlInfoIo         io;
}MEIControlInfo;
```

Change History: Modified in the 03.03.00

Description

MEIControlInfo contains the information about the motion controller being used.

mpi	Information about the MPI software located on the host computer.
firmware	Information about the Firmware running on the controller.
pld	Information about the PLD located in the controller.
rincon	Information about the Rincon FPGA located on the controller.
hardware	Production information about the hardware stored in the controller.
driver	Information about the Driver, running on the host, used to interface with the controller.
io	Information about the I/O directly connected to the controller.

See Also

MEIControlInfoDriver

Definition

```
typedef struct MEIControlInfoDriver {
   char   version[MEIControlSTRING_MAX];
} MEIControlInfoDriver;
```

Description

MEIControlInfoDriver is a structure that contains the version information of the connected hardware.

version	The version of the Driver the host uses to interface with the controller.

See Also

MEIControlInfoFirmware

Definition

```
typedef struct MEIControlInfoFirmware {
   long   version;      /* MEIXmpVERSION_EXTRACT(SoftwareID) */
   long   option;       /* MEIXmpOPTION_EXTRACT(Option) */
   char   revision;     /* ('A' - 1) + MEIXmpREVISION_EXTRACT(SoftwareID)*/
   long   subRevision;  /* MEIXmpSUB_REV_EXTRACT(Option) */
   long   branchId;
   MEIControlInfoFirmwareZMP   zmp;
} MEIControlInfoFirmware;
```

Change History: Modified in the 03.02.00

Description

MEIControlInfoFirmware is a structure that contains read-only version information for the firmware running in the controller.

version	The major version number for the controller's firmware. To be compatible with the MPI library, this number must match the fwVersion in the MEIControlInfoMpi structure.
option	The firmware option number. Special or custom firmware is given a unique option number. An application or user can identify optional firmware from this value.
revision	The minor version number for the controller's firmware. Indicates a minor change or bug fix to the firmware code.
subRevision	The micro version value for the controller's firmware. Indicates a very minor change or bug fix to the firmware code.
branchId	Identifies an intermediate branch software revision. The branch value is represented as a hex number between 0x00000000 and 0xFFFFFFFF. Each digit represents an instance of a branch (0x1 to 0xF). A single digit represents a single branch from a specific version, two digits represent a branch of a branch, three digits represent a branch of a branch of a branch, etc.
zmp	ZMP-only information. Contains versions and revision info for boot0 and zboot code.

See Also

MEIControlInfoMPI

MEIControlInfoFirmwareZMP

Definition

```
typedef struct MEIControlInfoFirmwareZMP {
   long     boot0Version;
   long     boot0Revision;
   long     zbootVersion;
   long     zbootRevision;
} MEIControlInfoFirmwareZMP;
```

Change History: Added in the 03.02.00

Description

MEIControlInfoFirmwareZMP is a structure containing version information about the boot0 and zboot code on a ZMP. Boot0 is the bootstrap code and should rarely need updating (updating is done at MEI). Zboot is the initialization code and will get updated every time the firmware is loaded.

NOTE: This information is displayed by the Version utility.

boot0Version	Version of boot0 code.
boot0Revision	Revision of boot0 code.
zbootVersion	Version of zboot code.
zbootRevision	Revision of zboot code.

See Also

MEIControlInfoIo

Definition

```
typedef struct MEIControlInfoIo {
    MEIControlInfoIoDigitalIn   digitalIn[MPIControlInMAX+1];
    MEIControlInfoIoDigitalOut  digitalOut[MPIControlOutMAX+1];
} MEIControlInfoIo;
```

Change History: Added in the 03.03.00

Description

MEIControlInfoIo contains information about the I/O directly connected to the controller.

digitalIn	information about the digital inputs.
digitalOut	information about the digital outputs.

See Also

Controller I/O | meiControlInfo | MEIControlInfo

MEIControlInfoIoDigitalIn

Definition

```
typedef struct MEIControlInfoIoDigitalIn {
    MPI_BOOL    supported;
    const char  *name;
} MEIControlInfoIoDigitalIn;
```

Change History: Added in the 03.03.00

Description

MEIControlInfoIoDigitalIn contains information about the digital inputs supported by this controller.

supported	a Boolean flag indicating whether or not the input is supported.
***name**	a string that gives a name for this input.

See Also

Controller I/O | meiControlInfo | MEIControlInfoIo

MEIControlInfoIoDigitalOut

Definition

```
typedef struct MEIControlInfoIoDigitalOut {
    MPI_BOOL    supported;
    const char  *name;
} MEIControlInfoIoDigitalOut;
```

Change History: Added in the 03.03.00

Description

MEIControlInfoIoDigitalOut contains information about the digital outputs supported by this controller.

supported	a Boolean flag indicating whether or not the output is supported.
***name**	a string that gives a name for this output.

See Also

Controller I/O | meiControlInfo | MEIControlInfoIo

MEIControlInfoHardware

Definition

```
typedef struct MEIControlInfoHardware {
   char   modelNumber[MEIControlSTRING_MAX];
   char   serialNumber[MEIControlSTRING_MAX];
   char   type[MEIControlSTRING_MAX];
} MEIControlInfoHardware;
```

Description

MEIControlInfoHardware is a structure that contains the version information of the connected hardware.

modelNumber	The Controller's model number or t-level number (ex: T001-0001) which is stored on the hardware.
seriallNumber	The Controller's serial number, which is unique to each controller.
type	The type of Controller (XMP or ZMP).

See Also

MEIControlInfoMpi

Definition

```
typedef struct MEIControlInfoMpi {
   char      version[MEIControlSTRING_MAX+1];
                 /* +1 for null termination character */
   long      fwVersion;
   long      fwOption;
} MEIControlInfoMpi;
```

Change History: Modified in the 03.03.00

Description

MEIControlInfoMpi is a structure that contains read-only version information for the MPI.

version	A string representing the version of the MPI. The version of the MPI is broken down by date, branch, and revision (MPIVersion.branch.revision). For ex: 20021220.1.2 means MPI version 20021220, branch 1, revision 2.
fwVersion	The firmware version information that the current version of the MPI will work with. A new field has been added to the XMP's firmware to identify and differentiate between intermediate branch software revisions. The branch value is represented as a hex number between 0x00000000 and 0xFFFFFFFF. Each digit represents an instance of a branch (0x1 to 0xF). A single digit represents a single branch from a specific version, two digits represent a branch of a branch, three digits represent a branch of a branch of a branch, etc.
fwOption	The firmware option number. Special or custom firmware is given a unique option number. An MPI library that requires optional firmware will have a value that must match the firmware's option number.

See Also

MEIControlInfoFirmware | MEIControlInfo

MEIControlInfoPld

Definition

```
typedef struct MEIControlInfoPld {
   char   version[MEIControlSTRING_MAX];
   char   option[MEIControlSTRING_MAX];
} MEIControlInfoPld;
```

Description

MEIControlInfoPld is a read-only structure that contains PLD version information. The PLD is a hardware component that contains logic to handle the controller's internal operation.

version	This is an 8-bit value in the hardware. The version string for the PLD. The PLD image is downloaded to the controller during manufacturing.
option	This is a 16-bit value (actually 2 8 bit values) in the hardware. The build option string for the PLD. The PLD option number is a coded value that describes the PLD image build type and target component. For XMP controllers, the option field has bits defining various features on the PCB - for example, the presence of the CAN interface, or the type of FPGA on the PCB.

See Also

MEIControlInfo

MEIControlInfoRincon

Definition

```
typedef struct MEIControlInfoRincon {
   char   version[MEIControlSTRING_MAX];
   char   package[MEIControlSTRING_MAX];
} MEIControlInfoRincon;
```

Description

MEIControlInfoRincon is a structure that contains read-only version information for the controller's Rincon image. The Rincon image contains the logic to operate a controller's SynqNet interface.

version	This is a 16-bit value in the hardware. The version string for the Rincon image on the controller.
package	This is a 16-bit value in the hardware. The package string identification for the Rincon. The package string is a coded value that describes the Rincon image build type and target component. Existing types are: **9201** - Rincon for XMP, XC2S100, PQ208 package **9601** - Rincon for XMP, XC2S100, FG256 package **A102** - RinconZ for ZMP, XC2S300E, FT256 package **A301** - RinconZ for ZMP, XC3S200, FT256 package The package and version data can be used to create the FPGA filename. For example, 221_9201.fpg is Rincon type 9201, version 221.

See Also

MEIControlInfo

MEIControlInput

Definition

```
typedef enum {
    MEIControlInputUSER_0   = MEIXmpControlIOMaskUSER0_IN,
    MEIControlInputUSER_1   = MEIXmpControlIOMaskUSER1_IN,
    MEIControlInputUSER_2   = MEIXmpControlIOMaskUSER2_IN,
    MEIControlInputUSER_3   = MEIXmpControlIOMaskUSER3_IN,
    MEIControlInputUSER_4   = MEIXmpControlIOMaskUSER4_IN,
    MEIControlInputUSER_5   = MEIXmpControlIOMaskUSER5_IN,
    MEIControlInputXESTOP   = MEIXmpControlIOMaskXESTOP,
} MEIControlInput;
```

Description

MEIControlInput is an enumeration of a controller's local digital input bit masks. Each mask represents a discrete input.

See Also

MEIControlOutput

MEIControlIoBit

Definition

```
typedef enum {

    MEIControlIoBitUSER_0_IN,
    MEIControlIoBitUSER_1_IN,
    MEIControlIoBitUSER_2_IN,
    MEIControlIoBitUSER_3_IN,
    MEIControlIoBitUSER_4_IN,
    MEIControlIoBitUSER_5_IN,
    MEIControlIoBitXESTOP,
    MEIControlIoBitUSER_0_OUT,
    MEIControlIoBitUSER_1_OUT,
    MEIControlIoBitUSER_2_OUT,
    MEIControlIoBitUSER_3_OUT,
    MEIControlIoBitUSER_4_OUT,
    MEIControlIoBitUSER_5_OUT,
} MEIControlIoBit;
```

Change History: Modified in the 03.02.00

Description

MEIControlIoBit is an enumeration of a controller's local digital I/O bit numbers.

MEIControlIoBitUSER_0_IN	controller's local input, bit number 0
MEIControlIoBitUSER_1_IN	controller's local input, bit number 1
MEIControlIoBitUSER_2_IN	controller's local input, bit number 2
MEIControlIoBitUSER_3_IN	controller's local input, bit number 3
MEIControlIoBitUSER_4_IN	controller's local input, bit number 4
MEIControlIoBitUSER_5_IN	controller's local input, bit number 5
MEIControlIoBitXESTOP	controller’s local input, External Emergency Stop Input. **NOTE**: The XESTOP bit does not have any special functionality. The bit number and name were kept for backwards compatibility.

MEIControlIoBitUSER_0_OUT	controller's local output, bit number 0
MEIControlIoBitUSER_1_OUT	controller's local output, bit number 1
MEIControlIoBitUSER_2_OUT	controller's local output, bit number 2
MEIControlIoBitUSER_3_OUT	controller's local output, bit number 3
MEIControlIoBitUSER_4_OUT	controller's local output, bit number 4
MEIControlIoBitUSER_5_OUT	controller's local output, bit number 5

See Also

meiControlIoBitGet

MPIControlMessage / MEIControlMessage

Definition: MPIControlMessage

```
typedef enum {
    MPIControlMessageLIBRARY_VERSION,
    MPIControlMessageADDRESS_INVALID,
    MPIControlMessageCONTROL_INVALID,
    MPIControlMessageCONTROL_NUMBER_INVALID,
    MPIControlMessageTYPE_INVALID,
    MPIControlMessageINTERRUPTS_DISABLED,
    MPIControlMessageEXTERNAL_MEMORY_OVERFLOW,
    MPIControlMessageADC_COUNT_INVALID,
    MPIControlMessageAXIS_COUNT_INVALID,
    MPIControlMessageAXIS_FRAME_COUNT_INVALID,
    MPIControlMessageCAPTURE_COUNT_INVALID,
    MPIControlMessageCOMPARE_COUNT_INVALID,
    MPIControlMessageFILTER_COUNT_INVALID,
    MPIControlMessageMOTION_COUNT_INVALID,
    MPIControlMessageMOTOR_COUNT_INVALID,
    MPIControlMessageSAMPLE_RATE_TO_LOW,
    MPIControlMessageSAMPLE_RATE_TO_HIGH,
    MPIControlMessageRECORDER_COUNT_INVALID,
    MPIControlMessageCOMPENSATOR_COUNT_INVALID,
    MPIControlMessageAXIS_RUNNING,
    MPIControlMessageRECORDER_RUNNING,
    MPIControlMessagePACK_ALIGNMENT,
} MPIControlMessage;
```

Change History: Modified in the 03.04.00.

Description

MPIControlMessage is an enumeration of Control error messages that can be returned by the MPI library.

MPIControlMessageLIBRARY_VERSION

The MPI Library does not match the application. This message code is returned by mpiControlInit(…) if the MPI's library (DLL) version does not match the MPI header files that were compiled with the application. To correct this problem, the application must be recompiled using the same MPI software installation version that the application uses at run-time.

MPIControlMessageADDRESS_INVALID

The controller address is not valid. This message code is returned by mpiControlInit(…) if the controller address is not within a valid memory range. mpiControlInit(…) only requires memory addresses for certain operating systems. To correct this problem, verify the controller memory address.

MPIControlMessageCONTROL_INVALID

Currently not supported.

MPIControlMessageCONTROL_NUMBER_INVALID

The controller number is out of range. This message code is returned by mpiControlInit(…) if the controller number is less than zero or greater than or equal to MaxBoards(8).

MPIControlMessageTYPE_INVALID

The controller type is not valid. This message code is returned by mpiControlInit(…) if the controller type is not a member of the MPIControlType enumeration.

MPIControlMessageINTERRUPTS_DISABLED

The controller interrupt is disabled. This message code is returned by mpiControlInterruptWait(…) if the controller's interrupt is not enabled. This prevents an application from waiting for an interrupt that will never be generated. To correct this problem, enable controller interrupts with mpiControlInterruptEnable(…) before waiting for an interrupt.

MPIControlMessageEXTERNAL_MEMORY_OVERFLOW

The controller's external memory will overflow. This message code is returned by mpiControlConfigSet(…) if the dynamic memory allocation exceeds the external memory available on the controller. To correct the problem, reduce the number/size of control configuration resources or use a controller model with a larger static memory component.

MPIControlMessageADC_COUNT_INVALID

The ADC count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of ADCs is greater than MEIXmpMAX_ADCs.

MPIControlMessageAXIS_COUNT_INVALID

The axis count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of axes is greater than MEIXmpMAX_Axes.

MPIControlMessageAXIS_FRAME_COUNT_INVALID

This message is returned from mpiControlConfigSet(...) if the value for MPIControlConfig. axisFrameCount is not a power of two or if axisFrameCount is less than MPIControlMIN_AXIS_FRAME_COUNT.

MPIControlMessageCAPTURE_COUNT_INVALID

The capture count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of captures is greater than MEIXmpMAX_Captures.

MPIControlMessageCOMPARE_COUNT_INVALID

The compare count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of compares is greater than MEIXmpMAX_Compares.

MPIControlMessageFILTER_COUNT_INVALID

The filter count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of filters is greater than MEIXmpMAX_Filters.

MPIControlMessageMOTION_COUNT_INVALID

The motion count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of motions is greater than MEIXmpMAX_MSs.

MPIControlMessageMOTOR_COUNT_INVALID

The motor count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of motors is greater than MEIXmpMAX_Motors.

MPIControlMessageSAMPLE_RATE_TO_LOW

The controller sample rate is too small. This message code is returned by mpiControlConfigSet(…) if the sample rate is less than MPIControlMIN_SAMPLE_RATE (1kHz). SynqNet does not support cyclic data rates below 1kHz. The controller's sample rate specifies the SynqNet cyclic rate.

MPIControlMessageSAMPLE_RATE_TO_HIGH

The controller sample rate is too big. This message code is returned by mpiControlConfigSet(…) if the sample rate is greater than MPIControlMAX_SAMPLE_RATE (100kHz).

MPIControlMessageRECORDER_COUNT_INVALID

The recorder count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of recorders is greater than MEIXmpMAX_Recorders.

MPIControlMessageCOMPENSATOR_COUNT_INVALID

The compensator count is not valid. This message code is returned by mpiControlConfigSet(…) if the number of compensators is greater than MPIControlMAX_COMPENSATORS.

MPIControlMessageAXIS_RUNNING

Attempting to configure the control object while axes are running. It is recommended that all configuration of the control object occur prior to commanding motion.

MPIControlMessageRECORDER_RUNNING

Attempting to configure the control object while a recorder is running. It is recommended that all configuration of the control object occur prior to operation of any recorder objects.

MPIControlMessagePACK_ALIGNMENT

The application was compiled with a packing alignment that is different from the MPI library.

For Windows, use the MSVC “/Zp8” compiler option or surround the MPI header files with the following #pragma pack statements:

```
#pragma  pack(push, 8)
#include "stdmpi.h"
#include "stdmei.h"
#include "apputil.h"
#pragma  pack(pop)
```

For Linux, use the “-malign-double” compiler option.

Definition: MEIControlMessage

```
typedef enum {
    MEIControlMessageFIRMWARE_INVALID,
    MEIControlMessageFIRMWARE_VERSION_NONE,
    MEIControlMessageFIRMWARE_VERSION,
    MEIControlMessageFPGA_SOCKETS,
    MEIControlMessageBAD_FPGA_SOCKET_DATA,
    MEIControlMessageNO_FPGA_SOCKET,
    MEIControlMessageINVALID_BLOCK_COUNT,
    MEIControlMessageSYNQNET_OBJECTS,
    MEIControlMessageSYNQNET_STATE,
    MEIControlMessageIO_BIT_INVALID,
} MEIControlMessage;
```

Description

MEIControlMessage is an enumeration of Control error messages that can be returned by the MPI library.

MEIControlMessageFIRMWARE_INVALID

The controller firmware is not valid. This message code is returned by mpiControlInit(…) if the MPI library does not recognize the controller signature. After power-up or reset, the controller loads the firmware from flash memory. When the firmware executes, it writes a signature value into external memory. If mpiControlInit(…) does not recognize the signature, then the firmware did not execute properly. To correct this problem, download firmware and verify the controller hardware is working properly.

MEIControlMessageFIRMWARE_VERSION_NONE

The controller firmware version is zero. This message code is returned by control methods do not find a firmware version. This indicates the firmware did not execute at controller power-up or reset. To correct this problem, download firmware and verify the controller hardware is working properly.

MEIControlMessageFIRMWARE_VERSION

The controller firmware version does not match the software version. This message code is returned by control methods if the firmware version is not compatible with the MPI library. To correct this problem, either download compatible firmware or install a compatible MPI run-tim library.

MEIControlMessageFPGA_SOCKETS

The maximum number of FPGA socket types has been exceeded. This message code is returned by meiFlashMemoryFromFile(…) if the controller has more FPGA types than the controller has flash memory space to support them.

MEIControlMessageBAD_FPGA_SOCKET_DATA

Currently not supported.

MEIControlMessageNO_FPGA_SOCKET

The FPGA socket type does not exist. This message code is returned by meiFlashMemoryFromFile(…) if the controller does not support the FPGA type that was specified in the FPGA image file. To correct this problem, use a different FPGA image that is compatible with the controller.

MEIControlMessageINVALID_BLOCK_COUNT

Currently not supported.

MEIControlMessageSYNQNET_OBJECTS

Currently not supported.

MEIControlMessageSYNQNET_STATE

The controller's SynqNet state is not expected. This message code is returned by mpiControlInit(…), mpiControlReset(…) and mpiControlConfigSet(…) if the SynqNet network initialization fails to reach the SYNQ state. To correct this problem, check your node hardware and network connections.

MEIControlMessageIO_BIT_INVALID

The controller I/O bit is not valid. This message code is returned by meiControlIoGet(...) and meiControlIoSet(...) if the controller I/O bit is not a member of the MEIControlIoBit enumeration.

See Also

MPIControlMemoryType

Definition

```
typedef enum {
    MPIControlMemoryTypeUSER,
    MPIControlMemoryTypeDEFAULT = MPIControlMemoryTypeUSER
} MPIControlMemoryType;
```

Description

MPIControlMemoryType is an enumeration of controller memory types. The controller memory contains static and dynamic regions. The controller firmware defines the regions and the MPI configures the dynamic memory.

MPIControlMemoryTypeUSER	The dynamic portion of the controller's external memory that is not in use by the controller.
MPIControlMemoryTypeDEFAULT	Defined as MPIControlMemoryTypeUSER.

See Also

mpiControlMemoryAlloc | mpiControlMemoryCount | mpiControlMemoryFree | mpiControlConfigGet | mpiControlConfigSet

MEIControlOutput

Definition

```
typedef enum {
    MEIControlOutputUSER_0  = MEIXmpControlIOMaskUSER0_OUT,
    MEIControlOutputUSER_1  = MEIXmpControlIOMaskUSER1_OUT,
    MEIControlOutputUSER_2  = MEIXmpControlIOMaskUSER2_OUT,
    MEIControlOutputUSER_3  = MEIXmpControlIOMaskUSER3_OUT,
    MEIControlOutputUSER_4  = MEIXmpControlIOMaskUSER4_OUT,
    MEIControlOutputUSER_5  = MEIXmpControlIOMaskUSER5_OUT,
} MEIControlOutput;
```

Description

MEIControlOutput is an enumeration of a controller's local digital output bit masks. Each mask represents a discrete output.

See Also

MEIControlInput

MPIControlStatus

Definition

```
typedef struct MPIControlStatus {
    MPIEventMask                eventMask;
    MPIControlFanStatusMask     fanStatus;
} MPIControlStatus;
```

Change History: Added in the 03.02.00

Description

MPIControlStatus is an MPI structure that is used to describe the current state of the controller's object. The XMP-Series controllers do not have fans and therefore do not support fanStatus. The ZMP-Series controllers have an optional processor cooling fan and support fanStatus.

eventMask	Array that defines the event mask bits. The controller event bits are defined in the MEIEventType enumeration.
fanStatus	Value is an enumeration of bit masks for the MPIControlFanStatusFlags. The status mask represents the present condition of the fan controller.

See Also

MPIControlFanStatusFlags | MPIControl | MPIControlStatus | MPIControlFanStatusMask | MEIEventType | meiEventMaskCONTROL

MEIControlStatistics

Definition

```
typedef struct MEIControlStatistics {
   double   maxForegroundTime;
   double   maxBackgroundTime;
   double   maxDelta;
   double   avgBackgroundRate;
   double   avgBackgroundTime;
   double   backgroundTime;
} MEIControlStatistics;
```

Change History: Added in the 03.04.00

Description

MEIControlStatistics contains read only statistics of the controller's processor load. It is useful for monitoring the foreground and background task execution times.

For more information about controller processor load, see SynqNet Controller Performance.

maxForegroundTime	Maximum amount of time to execute the controller's foreground task in microseconds (ms).
maxBackgroundTime	Maximum amount of time to execute the controller's background task in microseconds (ms).
maxDelta	Maximum number of times a full background task is interrupted by the foreground task. A value of zero means the background task executed to completion without being interrupted by the foreground task.
avgBackgroundRate	Average interrupted background cycle rate (cycles/sec).
avgBackgroundTime	Average interrupted background cycle time in microseconds (ms).
backgroundTime	Uninterrupted background cycle time in microseconds (ms).

See Also

meiControlStatistics | meiControlStatisticsReset

MEIControlTrace

Definition

```
typedef enum {
    MEIControlTraceDYNA_ALLOC  = MEIControlTraceFIRST << 0,
} MEIControlTrace;
```

Description

MEIControlTrace is an enumeration of control object trace bits to enable debug tracing.

MEIControlTraceDYNA_ALLOC	This trace bit enables tracing for calls that dynamically allocate controller memory.

See Also

MPIControlType

Definition

```
typedef enum {
    MPIControlTypeDEFAULT,
    MPIControlTypeMAPPED,
    MPIControlTypeIOPORT,
    MPIControlTypeDEVICE,
    MPIControlTypeCLIENT,
    MPIControlTypeFILE,
} MPIControlType;
```

Description

MPIControlType is an enumeration that specifies the type of controller that needs to be accessed when mpiControlCreate(...) is called. Please refer to the documentation for mpiControlCreate(...) to see how to use this enumeration.

See Also

MPIControl | mpiControlCreate | mpiControlType

MPIControlInMAX

Definition

```
#define  MPIControlInMAX  6
```

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

Description

MPIControlInMAX defines the maximum number of digital inputs a controller could support. The meiControlInfo(...) function returns details about the actual inputs that the current controller supports.

See Also

MPIControlOutMAX | meiControlInfo

MPIControlOutMAX

Definition

```
#define  MPIControlOutMAX  5
```

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

Description

MPIControlOutMAX defines the maximum number of digital outputs a controller could support. The meiControlInfo(...) function returns details about the actual outputs that the current controller supports.

See Also

MPIControlInMAX | meiControlInfo

MPIControlMAX_AXES

Definition

```
#define MPIControlMAX_AXES (32)
```

Description

MPIControlMAX_AXES defines the maximum number of axes available on one controller.

See Also

MPIAxis | mpiControlConfigGet | mpiControlConfigSet

MPIControlMAX_COMPENSATORS

Definition

```
#define MPIControlMAX_COMPENSATORS (12)
```

Change History: Modified in the 03.04.00.

Description

MPIControlMAX_COMPENSATORS defines the maximum number of compensator objects available on one controller.

See Also

MPICompensator | mpiControlConfigGet | mpiControlConfigSet

MPIControlMAX_RECORDERS

Definition

```
#define MPIControlMAX_RECORDERS (32)
```

Description

MPIControlMAX_RECORDERS defines the maximum number of recorder objects available on one controller.

See Also

MPIControlMIN_AXIS_FRAME_COUNT

Definition

```
#define MPIControlMIN_AXIS_FRAME_COUNT (128)
```

Required Header: stdmpi.h

Description

MPIControlMIN_AXIS_FRAME_COUNT defines the minimum allowed value for which MPIControlConfig.axisFrameCount can be set.

See Also

MPIControlConfig | mpiControlConfigGet | mpiControlConfigSet

MPIControlMAX_SAMPLE_RATE

Definition

```
#define MPIControlMAX_SAMPLE_RATE (100000)
```

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

Description

MPIControlMAX_SAMPLE_RATE defines the maximum allowed value for which MPIControlConfig. sampleRate can be set.

See Also

MPIControlConfig | mpiControlConfigGet | mpiControlConfigSet | MPIControlMIN_SAMPLE_RATE

MPIControlMIN_SAMPLE_RATE

Definition

```
#define MPIControlMIN_SAMPLE_RATE (1000)
```

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

Description

MPIControlMIN_SAMPLE_RATE defines the minimum allowed value for which MPIControlConfig. sampleRate can be set.

See Also

MPIControlConfig | mpiControlConfigGet | mpiControlConfigSet | MPIControlMAX_SAMPLE_RATE

MPIControlRECORD_COUNT_DEFAULT

Definition

```
#define  MPIControlRECORD_COUNT_DEFAULT  (16384)
```

Change History: Added in the 03.04.00

Description

MPIControlRECORD_COUNT_DEFAULT defines the default number of recorder counts available on one controller.

See Also

MEIControlSTRING_MAX

Definition

```
#define MEIControlSTRING_MAX (128)
```

Change History: Modified in the 03.03.00

Description

MEIControlSTRING_MAX defines the maximum number of characters in MEIControlInfo strings.

See Also

MEIControlInfo | MEIControlInfoHardware

mpiControlFanStatusMaskBIT

Declaration

```
#define  mpiControlFanStatusMaskBIT(flag)  (int)(0x1 << (flag))
```

Required Header: stdmpi.h
Change History: Added in the 03.02.00

Description

mpiControlFanStatusMaskBIT is a utility macro to convert MPIControlFanStatusFlag defines to MPIControlFanStatusMask values.

An application should use the MPIControlFanStatusMask values instead of this macro.

See Also

MPIControlStatus | MPIControlFanStatusFlag | MPIControlFanStatusMask

TCP/IP and Sockets for Control Objects

The MPI implements network functionality as client/server. The xmp\util\server.c program implements a basic server. You just create a Control object of type MPIControlTypeCLIENT and specify the server's host in the MPIControlAddress{}.client{} structure.

You can try "MPI networking" on a single machine by starting up the server program in a DOS window, and then running a sample application in another DOS window. Note that you can specify the host name and port of the server as command line arguments to all sample applications and utilities.

The way the MPI client/server works internally is that low-level mpiControlMemory and mpiControlInterrupt methods are intercepted just before they read/write XMP memory. The methods are packaged up as remote procedure calls and sent to the server for execution. The server sends the results back to the client.

There are 2 channels of communication - one channel to wait for interrupts, and another channel to do everything else. All MPI methods that communicate with the XMP do so by calling (eventually) the low level mpiControlMemory methods, so no application code needs to be changed other than the initial call to mpiControlCreate. This is all implemented on WinNT using WinSock.

Note that it would be possible to implement the client/server scenario above using an RS-232 line rather than TCP/IP WinSock. The MPI's client/server protocol only requires a reliable transport mechanism (WinSock, RS-232) between a client and server.

Return to Control Objects page

DriveMap Objects

Introduction

A **DriveMap** object is used to access information in a drive map file. For more information, see SynqNet Drive Parameters.

| Error Messages |

Methods

Create, Delete, Validate Methods

meiDriveMap**Create**

meiDriveMap**Delete**

meiDriveMap**Validate**

Configuration and Information Methods

meiDriveMap**ParamCount**

meiDriveMap**ParamList**

meiDriveMap**ConfigCount**

meiDriveMap**ConfigList**

Data Types

MEIDriveMap**Message**

MEIDriveMap**ParamAccess**

MEIDriveMap**ParamInfo**

MEIDriveMap**ParamType**

MEIDriveMap**ParamValue**

Constants

MEIDriveMap**ParamInfoMAX_STRING_LENGTH**

MEIDriveMap**ParamMAX_STRING_LENGTH**

meiDriveMapCreate

Declaration

```
meiDriveMapCreate (char  *fileName)
```

Required Header: stdmei.h

Description

meiDriveMapCreate creates a DriveMap object associated with the file specified by ***fileName***.

DriveMapCreate is the equivalent of a C++ constructor.

***filename**	the drive map file.

Return Values	
handle	to a DriveMap object. After creating a DriveMap object, it must be validated using meiDriveMapValidate(...).
MPIHandleVOID	if the object could not be created.

See Also

meiDriveMapDelete | meiDriveMapValidate

meiDriveMapDelete

Declaration

```
meiDriveMapDelete(MEIDriveMap  driveMap);
```

Required Header: stdmei.h

Description

meiDriveMapDelete deletes a DriveMap object and invalidates its handle.

DriveMapDelete is the equivalent of a C++ destructor.

driveMap	a handle of the DriveMap object to delete in the reverse order to avoid memory leaks.

Return Values	
MPIMessageOK	

See Also

meiDriveMapCreate | meiDriveMapValidate

meiDriveMapValidate

Declaration

```
long meiDriveMapValidate(MEIDriveMap  driveMap);
```

Required Header: stdmei.h

Description

meiDriveMapValidate validates a DriveMap object and its handle.

DriveMapValidate is the equivalent of a C++ constructor.

driveMap	a handle to a DriveMap object.

Return Values	
MPIMessageOK	

See Also

meiDriveMapCreate | meiDriveMapDelete

meiDriveMapParamCount

Declaration

```
long meiDriveMapParamCount (MEIDriveMap   driveMap,
                            char          *nodeName,
                            char          *firmwareVersion,
                            long          *paramsCount);
```

Required Header: stdmei.h

Description

meiDriveMapParamCount scans the drive map file for a drive entry that matches a particular drive. If an entry is found, then this function returns the number of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiDriveMapParamList(...) function. First, this function is called in order to get the size of the drive parameter list. Then the user can use this size to allocate enough memory to hold the complete parameter list before calling meiDriveMapParamList to fill in the list.

driveMap	a handle to a DriveMap object.
nodeName	the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retreived by calling meiSqNodeInfo. See MEISqNodeInfo.
firmwareVersion	The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See MEISqNodeDriveInfo.
***paramsCount**	pointer to the variable that will be set by this function.

Return Values	
MPIMessageOK	

See Also

meiDriveMapParamList

meiDriveMapParamList

Declaration

```
long meiDriveMapParamList (MEIDriveMap          driveMap,
                           char                 *nodeName,
                           char                 *firmwareVersion,
                           long                 paramCount,
                           MEIDriveParamInfo    *driveParamInfo);
```

Required Header: stdmei.h

Description

meiDriveMapParamList scans the drive map file for an entry that matches a particular drive. If a drive entry is found, this function writes the drive parameter information about each of the drive parameters to the ***driveParamInfo*** list.

This function is normally used with the meiDriveMapParamCount(...) function. The meiDriveMapParamCount function is called first to get the size of the parameter list, the user can then use this size to allocate enough memory to hold the complete parameter list before calling this function to fill in the parameter list.

driveMap	a handle to a DriveMap object.
***nodeName**	the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retreived by calling meiSqNodeInfo(...). See MEISqNodeInfo.
***firmwareVersion**	The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo(...). See MEISqNodeDriveInfo.
paramCount	the number of drive parameter information records that can be written to the driveParamInfo list.
***driveParamInfo**	pointer to the list of drive parameter information records that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiDriveMapParamCount

meiDriveMapConfigCount

Declaration

```
long meiDriveMapConfigCount (MEIDriveMap  driveMap,
                             char         *nodeName,
                             char         *firmwareVersion,
                             long         *configCount);
```

Required Header: stdmei.h

Description

meiDriveMapConfigCount scans the drive map file for a drive entry that matches a particular drive. If an entry is found, this function returns the number of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiDriveMapConfigList function. This function is called first in order to get the size of the drive configuration list. Then the user can use this size to allocate enough memory to hold the complete configuration list before calling meiDriveMapConfigList to fill in the list.

driveMap	a handle to a DriveMap object.
nodeName	the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retreived by calling meiSqNodeInfo. See MEISqNodeInfo.
firmwareVersion	The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See MEISqNodeDriveInfo.
***configCount**	pointer to the variable that will be set by this function.

Return Values	
MPIMessageOK	

See Also

meiDriveMapConfigList

meiDriveMapConfigList

Declaration

```
long meiDriveMapConfigList (MEIDriveMap   driveMap,
                            char          *nodeName,
                            char          *firmwareVersion,
                            long          configCount,
                            long          *configList);
```

Required Header: stdmei.h

Description

meiDriveMapConfigList scans the drive map file for a drive entry that matches a particular drive. If an entry is found, this function returns the list of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiDriveMapConfigCount(...) function. The meiDriveMapConfigCount function is called first in order to get the size of the drive configuration list. Then the user can use this size as a guide to allocate enough memory to hold the complete configuration list before calling this function to fill in the list.

driveMap	a handle to a DriveMap object.
***nodeName**	the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retreived by calling meiSqNodeInfo. See MEISqNodeInfo.
***firmwareVersion**	The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See MEISqNodeDriveInfo.
configCount	the number of drive parameter information records that can be written to the configList list.
***configList**	pointer to the list of drive parameters that make up the drive configuration that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiDriveMapConfigCount

MEIDriveMapMessage

Definition

```
typedef enum {
   MEIDriveMapMessageMAP_FILE_OPEN_ERROR,
   MEIDriveMapMessageMAP_FILE_FORMAT_INVALID,
   MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP,
   MEIDriveMapMessageVERSION_NOT_FOUND_IN_MAP,
   MEIDriveMapMessageDRIVE_PARAM_READ_ONLY,
} MEIDriveMapMessage;
```

Description

MEIDriveMapMessageMAP_FILE_OPEN_ERROR

There was an error when opening the drive map file. The file may not exist, or access to the directory may not be allowed.

MEIDriveMapMessageMAP_FILE_FORMAT_INVALID

The format of the drive map file is invalid.

MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP

The node type specified by the nodeName parameter was not found in the driveMap file.

MEIDriveMapMessageVERSION_NOT_FOUND_IN_MAP

The drive firmware version specified by the firmwareVersion parameter was not found in the driveMap file.

MEIDriveMapMessageDRIVE_PARAM_READ_ONLY

A read-only parameter is included in the configuration list for the specified drive in the driveMap file.

See Also

MEIDriveMapParamAccess

Definition

```
typedef enum MEISqNodeDriveParamAccess {
    MEIDriveMapParamAccessREAD_WRITE,
    MEIDriveMapParamAccessREAD_ONLY,
} MEIDriveMapParamAccess;
```

Description

MEIDriveMapParamAccess indicates what type of access is possible with the drive parameter.

This field of the MEIDriveMapParamInfo structure indicates what type of access is possible with this drive parameter.

See Also

MEIDriveMapParamInfo

MEIDriveMapParamInfo

Definition

```
typedef struct MEIDriveMapParamInfo {
    long                    parameter;
    char                    name[MEIDriveMapParamInfoMAX_STRING_LENGTH];
    MEIDriveMapParamAccess  access;
    MEIDriveMapParamType    type;
    char                    validValues[MEIDriveMapParamInfoMAX_STRING_LENGTH];
    long                    defaultValue;
    char                    help[MEIDriveParamInfoMAX_STRING_LENGTH];
} MEIDriveMapParamInfo;
```

Change History: Modified in the 03.03.00

Description

MEIDriveMapParamInfo holds a set of information describing a drive parameter. This structure is read from the drives map file "drives.dm."

parameter	the number used to address this drive parameter.
name	a null terminated string giving a name for this drive parameter.
access	defines if this drive parameter is read-only or read-write.
type	the data type for this drive parameter. (ex: signed32, unsigned32, float, etc.)
validValues	a string describing the possible valid values for this drive parameter.
defaultValue	The factory default value this drive parameter.
help	A string describing this drive parameter.

See Also

MEIDriveMapParamType

MEIDriveMapParamType

Definition

```
typedef enum MEIDriveMapParamType {
    MEIDriveMapParamTypeSIGNED32,
    MEIDriveMapParamTypeSIGNED16,
    MEIDriveMapParamTypeSIGNED8,
    MEIDriveMapParamTypeUNSIGNED32,
    MEIDriveMapParamTypeUNSIGNED16,
    MEIDriveMapParamTypeUNSIGNED8,
    MEIDriveMapParamTypeHEX,
    MEIDriveMapParamTypeENUMERATED,
    MEIDriveMapParamTypeMASK,
    MEIDriveMapParamTypeCHARACTER,
    MEIDriveMapParamTypeSTRING,
    MEIDriveMapParamTypeSINGLE,
    MEIDriveMapParamTypeACTION,
} MEIDriveMapParamType;
```

Description

MEIDriveMapParamType is an enumeration that indicates which data format should be used with the drive parameter.

MEIDriveMapParamTypeSIGNED32	A signed 32bit integer.
MEIDriveMapParamTypeSIGNED16	A signed 16bit integer.
MEIDriveMapParamTypeSIGNED8	A signed 8bit integer.
MEIDriveMapParamTypeUNSIGNED32	An unsigned 32bit integer.
MEIDriveMapParamTypeUNSIGNED16	An unsigned 16bit integer.
MEIDriveMapParamTypeUNSIGNED8	An unsigned 8bit integer.
MEIDriveMapParamTypeHEX	An integer representred in hexadecimal notation.
MEIDriveMapParamTypeENUMERATED	An integer where each value can be represented by a different name.
MEIDriveMapParamTypeMASK	An integer where each bit has a unique name.
MEIDriveMapParamTypeCHARACTER	An single ASCII character.

MEIDriveMapParamTypeSTRING	A null terminated string of characters.
MEIDriveMapParamTypeSINGLE	A single precision floating point number.
MEIDriveMapParamTypeACTION	Writing to this parameter will perform an action on the drive. No data is assiated with this type.

See Also

MEIDriveMapParamValue

Definition

```
typedef union {
    long            signed32;
    short           signed16;
    char            signed8;
    unsigned long   unsigned32;
    unsigned short  unsigned16;
    unsigned char   unsigned8;
    unsigned long   enumerated;
    unsigned long   hex;
    unsigned long   mask;
    char            character;
    char            string[MEIDriveMapParamMAX_STRING_LENGTH];
    float           single;
} MEIDriveMapParamValue;
```

Change History: Modified in the 03.03.00

Description

The **MEIDriveMapParamValue** union holds the value of a drive parameter. The different fields allow this data type to hold all the different types of drive parameters. The MEISqNodeDriveParamType enumeration is used to identify which of the fields within the MEISqNodeDriveParamValue union to use.

See Also

MEIDriveMapParamType

MEIDriveMapParamInfoMAX_STRING_LENGTH

Declaration

```
#define  MEIDriveMapParamInfoMAX_STRING_LENGTH  (256)
```

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

Description

MEIDriveMapParamInfoMAX_STRING_LENGTH macro defines the maximum length of a string that can be used to describe a drive parameter.

See Also

MEIDriveMapParamMAX_STRING_LENGTH

Declaration

```
#define  MEIDriveMapParamMAX_STRING_LENGTH  (256)
```

Required Header: stdmei.h

Description

MEIDriveMapParamMAX_STRING_LENGTH macro defines the maximum length of a string that can be read from, or written to a drive parameter. The value is defined by a drive map constant.

See Also

Event Objects

Introduction

An **Event** object contains information about an asynchronous event. Typically, events are generated by the controller, but in some special cases it is possible to generate events from the host computer.

The Event object is retrieved through the EventMgr, via the Notify object. The Event object contains data about the type of event, its source, and other information. The user Event fields can be configured to collect data at the time when the event occurs in the controller.

| Error Messages |

Methods

Configuration and Information Methods

mpiEvent**StatusGet**	Get Event status
mpiEvent**StatusSet**	Set Event status
mpiEvent**TypeName**	Get Event type name

Data Types

MPIEvent**Message**
MEIEvent**NotifyData**
MPIEvent**Status**
MEIEvent**StatusInfo**
MPIEvent**Type** / MEIEvent**Type**

Constants

MPIEvent**StatusINFO_COUNT_MAX**	defines the size of the MPIEventStatus.info[] array.

mpiEventStatusGet

Declaration

```
long mpiEventStatusGet(MPIEvent       event,
                       MPIEventStatus *status)
```

Required Header: stdmpi.h

Description

mpiEventStatusGet gets the status of an Event object (***event***) and writes it into the structure pointed to by ***status***. Event status includes the event type, type-specific codes and the event source.

Return Values	
MPIMessageOK	

Sample Code

```
/* Prototype for logging function */
void logToFile(const char*);

MPIEventStatus   eventStatus;
long             returnValue;

/* Wait for motion event */
returnValue =
    mpiNotifyEventWait(notify,
                       &eventStatus,
                       MPIWaitFOREVER);
msgCHECK(returnValue);

/* Log event */
logToFile(mpiEventTypeName(eventStatus->type));
```

See Also

mpiEventStatusSet | meiEventStatusInfo | MPIEventType

EventLog.c

mpiEventStatusSet

Declaration

```
long mpiEventStatusSet(MPIEvent       event,
                       MPIEventStatus *status)
```

Required Header: stdmpi.h

Description

mpiEventStatusSet sets (writes) the status of ***event*** using data from the structure pointed to by ***status***. Event status includes the event type, type-specific codes and the event source.

Return Values	
MPIMessageOK	

Sample Code

```
/* Prototype for logging function */
void logToFile(const char*);

MPIEventStatus   eventStatus;
long             returnValue;

/* Wait for motion event */
returnValue =
    mpiNotifyEventWait(notify,
                       &eventStatus,
                       MPIWaitFOREVER);
msgCHECK(returnValue);

/* Log event */
logToFile(mpiEventTypeName(eventStatus->type));
```

See Also

mpiEventStatusGet | meiEventStatusInfo | MPIEventType

EventLog.c

mpiEventTypeName

Declaration

```
const char* mpiEventTypeName(MPIEventType eventType);
```

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

Description

mpiEventTypeName returns a text description for MPI events. mpiEventTypeName should be called when a text description of the event type is needed.

Return Values	
"Unknown Event"	if *EventTypeName* cannot identify **eventType**.
pointer to Event Type Name	if *EventTypeName* can identify **eventType**.

Sample Code

```
/* Prototype for logging function */
void logToFile(const char*);

MPIEventStatus   eventStatus;
long             returnValue;

/* Wait for motion event */
returnValue =
    mpiNotifyEventWait(notify,
                       &eventStatus,
                       MPIWaitFOREVER);
msgCHECK(returnValue);

/* Log event */
logToFile(mpiEventTypeName(eventStatus->type));
```

See Also

MPIEventType

EventLog.c

MPIEventMessage

Definition

```
typedef enum {
    MPIEventMessageEVENT_INVALID,
} MPIEventMessage;
```

Description

MPIEventMessage is an enumeration of Event error messages that can be returned by the MPI library.

MPIEventMessageEVENT_INVALID
The event type is not valid. This message code is returned by mpiEventStatusSet(...) if the event type is not a member of the MPIEventType or MEIEventType enumerations.

See Also

MEIEventNotifyData

Definition

```
typedef struct MEIEventNotifyData {
    void    *address[MEIXmpSignalUserData];
} MEIEventNotifyData;
```

Description

The ***address*** of an **MEIEventNotifyData** structure is passed as the third (void *external) argument to mpi***Object***EventNotifyGet/Set(...)†.

The address array contains host-based XMP addresses, the contents of which are returned in MEIEventStatusInfo{}.data.

† ***Object*** represents an MPI object like Axis or Motion. Therefore, mpi***Object***EventNotifyGet/Set(...) represents functions like mpiAxisEventNotifyGet(...) and mpiAxisEventNotifySet(...).

See Also

MEIEventStatusInfo

MPIEventStatus

Definition

```
typedef struct MPIEventStatus {
    MPIEventType   type;
    void           *source;
    long           info[MPIEventStatusINFO_COUNT_MAX];
} MPIEventStatus;
```

Description

MPIEventStatus holds information about a particular event that was generated by the XMP.

type	identifies the type of event that was generated.
***source**	identifies what the source of the event was. source will either be a handle to an MPI object or a host pointer. Use mpiObjectModuleId() to identify what source points to.
info	Contains information on what generated the event and the conditions under which it was generated. MEIEventStatusInfo simplifies decoding this array. Sample code is shown on the MEIEventStatusInfo page.

See Also

mpiObjectModuleId | MPIEventType | MPIEventMgr | MPINotify | MEIEventStatusInfo | MPIEventStatusINFO_COUNT_MAX

MEIEventStatusInfo

Definition

```
typedef struct MEIEventStatusInfo {
    union {
        MPIHandle  handle;  /* generic */
        MPIAxis    axis;    /* MEIEventTypeAXIS_FIRST ...
                               MEIEventTypeAXIS_LAST - 1 */
        long       node;    /* MEIEventTypeCAN_FIRST...
                               MEIEventTypeCAN_LAST - 1  */
        long       number;  /* MPIEventTypeMOTION
                               MPIEventTypeMOTOR_FIRST...
                               MPIEventTypeMOTOR_LAST - 1

                                     MEIEventTypeMOTOR_FIRST ...
                               MEIEventTypeMOTOR_LAST - 1 */
        long       value;   /* MPIEventTypeEXTERNAL */
            } type;

            MEIXmpSignalID signalID;

            /* Contents of addresses specified by MEIEventNotifyData{} */
            union {
         long sampleCounter;
         struct {
              long sampleCounter;
         } motion;
         struct {
              long     sampleCounter;
              long     positionError;
              MEIInt64 actualPosition;
              MEIInt64 commandPosition;
         } axis;
         struct {
            /* Data associated with the CAN event. */
            long data[4];
        } can;
         struct {
              long      sampleCounter;
              long      dedicatedIn;
              MEIInt64  encoderPosition;
         } motor;
         long word[MEIXmpSignalUserData];
     } data;
} MEIEventStatusInfo;
```

Change History: Modified in the 03.04.00.

Description

MEIEventStatusInfo is an information structure that tells the XMP what the data in MPIEventStatus.info holds. The information that is returned by this structure is only valid if the default configurations for the recorder are used.

type	A union that specifies the object handle, motion number, or external ID value that generated the event
type.handle	A generic object handle. Used by MPIRecorder and MPIMotor events
type.axis	An axis object handle. Used by MPIAxis events
type.node	The CAN Node number of the MEICan object that generated the event.
type.number	The motion number of the MPIMotion object that generated the event
type.value	An ID value used to identify what external source or MPISequence event was generated
signalID	Specifies what type of object actually generated the event
data	A union that contains extra data about the event that was generated
data.sampleCounter	The value of the sampleCounter when the event was generated
data.motion	A union that contains extra data about the motion event that was generated
data.motion.sampleCounter	The value of the sampleCounter when the motion event was generated
data.axis	A union that contains extra data about the axis event that was generated
data.axis.sampleCounter	The value of the sampleCounter when the axis.event was generated
data.axis.positionError	The value of the axis' position error when the axis event was generated. Data is represented as a float.
data.axis.actualPosition	The value of the axis' actual position when the event was generated
data.axis.commandPosition	The value of the axis' command position when the axis event was generated

data.can.data	A union that contains extra data about the CAN event that was generated.
data.motor	A union that contains extra data about the motor event that was generated
data.motor.sampleCounter	The value of the sampleCounter when the motor event was generated
data.motor.dedicatedIn	The value of the motor's dedicatedIn word when the motor event was generated
data.motor.encoderPosition	The value of the motor's ecoder position when the event was generated
data.word[]	The extra data about the event that was generated formatted as an array of long values

Sample Code

```
    MPINotify      notify
        MPIEventStatus eventStatus;

        . . .

        /* Wait for event */
        returnValue =
            mpiNotifyEventWait(notify,
                                        &eventStatus,
                                         MPIWaitFOREVER);
        msgCHECK(returnValue);

        if (eventStatus.type == MPIEventTypeMOTION_DONE) {
            MEIEventStatusInfo *info;

            info = (MEIEventStatusInfo *)eventStatus.info;

            . . .
        }
```

See Also

MPIEventStatus | MPIAxis

MPIEventType / MEIEventType

Definition: MPIEventType

```
typedef enum {
    MPIEventTypeINVALID,

    MPIEventTypeNONE,                /*  0 */

    /* Motor events */
    MPIEventTypeAMP_FAULT,           /*  1 */
    MPIEventTypeHOME,                /*  2 */
    MPIEventTypeLIMIT_ERROR,         /*  3 */
    MPIEventTypeLIMIT_HW_NEG,        /*  4 */
    MPIEventTypeLIMIT_HW_POS,        /*  5 */
    MPIEventTypeLIMIT_SW_NEG,        /*  6 */
    MPIEventTypeLIMIT_SW_POS,        /*  7 */
    MPIEventTypeENCODER_FAULT,       /*  8 */
    MPIEventTypeAMP_WARNING,         /*  9 */

    /* Motion events */
    MPIEventTypeMOTION_DONE,         /* 10 */
    MPIEventTypeMOTION_AT_VELOCITY,  /* 11 */

    /* Recorder events */
    MPIEventTypeRECORDER_HIGH,       /* 12 */
    MPIEventTypeRECORDER_FULL,       /* 13 */
    MPIEventTypeRECORDER_DONE,       /* 14 */

    /* External events */
    MPIEventTypeEXTERNAL,            /* 15 */
} MPIEventType;
```

Description

MPIEventType is used by the MPIEventMask macros to help generate event masks.

MPIEventTypeNONE	This event type indicates no event was generated.
MPIEventTypeAMP_FAULT	This event type indicates an Amp Fault event was generated from a Motor object.
MPIEventTypeHOME	This event type indicates a Home event was generated from a Motor object.
MPIEventTypeLIMIT_ERROR	This event type indicates a position Error Limit was generated from a Motor object.
MPIEventTypeLIMIT_HW_NEG	This event type indicates a Negative Hardware Limit event was generated from a Motor object.
MPIEventTypeLIMIT_HW_POS	This event type indicates a Positive Hardware Limit event was generated from a Motor object.
MPIEventTypeLIMIT_SW_NEG	This event type indicates a Negative Software Limit event was generated from a Motor object.
MPIEventTypeLIMIT_SW_POS	This event type indicates a Positive Software Limit event was generated from a Motor object.
MPIEventTypeENCODER_FAULT	This event type indicates an Encoder Fault event was generated from a Motor object. See Use of MPIEventTypeENCODER_FAULT.
MPIEventTypeAMP_WARNING	This event type indicates an Amp Warning event was generated from a Motor object.
MPIEventTypeMOTION_DONE	This event type indicates a Motion Done event was generated from a Motion Supervisor object.
MPIEventTypeMOTION_AT_VELOCITY	This event type indicates an At Velocity event was generated from a Motion Supervisor object.
MPIEventTypeRECORDER_HIGH	This event type indicates that the controller's recorded data exceeded the buffer's high limit.
MPIEventTypeRECORDER_FULL	This event type indicates that the controller's recorded data has filled the buffer.
MPIEventTypeRECORDER_DONE	This event type indicates that the controller has recorded the number of requested data records.
MPIEventTypeEXTERNAL	This event type indicates an External event was generated from an external source.

Definition: MEIEventType

```
typedef enum {
    /* Controller events */
    MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED,
    MEIEventTypeCONTROL_FAN,

    /* Motor events */
    MEIEventTypeLIMIT_USER0,
    MEIEventTypeLIMIT_USER1,
    MEIEventTypeLIMIT_USER2,
    MEIEventTypeLIMIT_USER3,
    MEIEventTypeLIMIT_USER4,
    MEIEventTypeLIMIT_USER5,
    MEIEventTypeLIMIT_USER6,
    MEIEventTypeLIMIT_USER7,
    MEIEventTypeLIMIT_USER8,
    MEIEventTypeLIMIT_USER9,
    MEIEventTypeLIMIT_USER10,
    MEIEventTypeLIMIT_USER11,
    MEIEventTypeLIMIT_USER12,
    MEIEventTypeLIMIT_USER13,
    MEIEventTypeLIMIT_USER14,
    MEIEventTypeLIMIT_USER15,

    /* Motion events */
    MEIEventTypeMOTION_OUT_OF_FRAMES,

    /* Axis events */
    MEIEventTypeIN_POSITION_COARSE,
    MEIEventTypeIN_POSITION_FINE,
    MEIEventTypeSETTLED
    MEIEventTypeAT_TARGET,
    MEIEventTypeFRAME,

    /* SynqNet events */
    MEIEventTypeSYNQNET_DEAD,
    MEIEventTypeSYNQNET_RX_FAILURE,
    MEIEventTypeSYNQNET_TX_FAILURE,
    MEIEventTypeSYNQNET_NODE_FAILURE,
    MEIEventTypeSYNQNET_RECOVERY,

    /* SqNode events */
    MEIEventTypeSQNODE_IO_ABORT,
    MEIEventTypeSQNODE_NODE_DISABLE,
    MEIEventTypeSQNODE_NODE_ALARM,
    MEIEventTypeSQNODE_ANALOG_POWER_FAULT,
    MEIEventTypeSQNODE_USER_FAULT,
    MEIEventTypeSQNODE_NODE_FAILURE,
    MEIEventTypeSQNODE_IO_FAULT,

    /* CAN events */
    MEIEventTypeCAN_BUS_STATE,
```

```
    MEIEventTypeCAN_RECEIVE_OVERRUN,
    MEIEventTypeCAN_EMERGENGY,
    MEIEventTypeCAN_NODE_BOOT,
    MEIEventTypeCAN_HEALTH,
    MEIEventTypeCAN_DIGITAL_INPUT,
    MEIEventTypeCAN_ANALOG_INPUT,

} MEIEventType;
```

Change History: Modified in the 03.03.00.

Description

MEIEventType is used by the MPIEventMask macros to help generate event masks.

MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED	This is an event that occurs if the xmp. SystemData.SyncInterrupt.ProcessFlag is set when SynqNet data is transmitted at the end of the firmware's foreground cycle. If the user is using the **SynqInterrupt** feature and sets the **ProcessFlag** at the beginning of the foreground cycle, the firmware checks to see if the user cleared the **ProcessFlag** by the time SynqNet data is transmitted. If the ProcessFlag has not been cleared, the event occurs.
MEIEventTypeCONTROL_FAN	This is an event that can occur when the on-board fan controller detects an error (overheating, fan failure, etc…). **NOTE**: This is for the ZMP only and will not occur on an XMP.
MEIEventTypeLIMIT_USER0	This event type indicates a User Limit event was generated from a Motor object. User Limit number 0.
MEIEventTypeLIMIT_USER1	This event type indicates a User Limit event was generated from a Motor object. User Limit number 1.
MEIEventTypeLIMIT_USER2	This event type indicates a User Limit event was generated from a Motor object. User Limit number 2.
MEIEventTypeLIMIT_USER3	This event type indicates a User Limit event was generated from a Motor object. User Limit number 3.

MEIEventTypeLIMIT_USER4	This event type indicates a User Limit event was generated from a Motor object. User Limit number 4.
MEIEventTypeLIMIT_USER5	This event type indicates a User Limit event was generated from a Motor object. User Limit number 5.
MEIEventTypeLIMIT_USER6	This event type indicates a User Limit event was generated from a Motor object. User Limit number 6.
MEIEventTypeLIMIT_USER7	This event type indicates a User Limit event was generated from a Motor object. User Limit number 7.
MEIEventTypeLIMIT_USER8	This event type indicates a User Limit event was generated from a Motor object. User Limit number 8.
MEIEventTypeLIMIT_USER9	This event type indicates a User Limit event was generated from a Motor object. User Limit number 9.
MEIEventTypeLIMIT_USER10	This event type indicates a User Limit event was generated from a Motor object. User Limit number 10.
MEIEventTypeLIMIT_USER11	This event type indicates a User Limit event was generated from a Motor object. User Limit number 11.
MEIEventTypeLIMIT_USER12	This event type indicates a User Limit event was generated from a Motor object. User Limit number 12.
MEIEventTypeLIMIT_USER13	This event type indicates a User Limit event was generated from a Motor object. User Limit number 13.
MEIEventTypeLIMIT_USER14	This event type indicates a User Limit event was generated from a Motor object. User Limit number 14.
MEIEventTypeLIMIT_USER15	This event type indicates a User Limit event was generated from a Motor object. User Limit number 15.
MEIEventTypeMOTION_OUT_OF_FRAMES	This event type indicates that the number of frames left to be executed in the controller has reached the buffer empty limit / emptyCount (see MPIMotionPoint). Therefore the controller will perform the specified eStop action.

MEIEventTypeIN_POSITION_COARSE	This event type indicates an In Coarse Position event was generated from an Axis object. See Axis Tolerances and Related Events and MPIAxisInPosition.
MEIEventTypeIN_POSITION_FINE	This event type indicates that an In Fine Position event was generated from an Axis object. See Axis Tolerances and Related Events and MPIAxisInPosition.
MEIEventTypeSETTLED	Equivalent to MEIEventTypeIN_POSITION_FINE.
MEIEventTypeAT_TARGET	Reserved Frame Event.
MEIEventTypeFRAME	This event type is currently not supported and is reserved for future use.
MEIEventTypeSYNQNET_DEAD	The SynqNet network was shutdown due to a communication failure. This status/event occurs when the controller fails to read/write data to the SynqNet network interface from an RX_FAILURE or a TX_FAILURE. To recover from a DEAD event, the network must be shutdown and reinitialized. SYNQNET_DEAD is latched by the controller, use meiSynqNetEventReset(...) to clear the status/ event bit.
MEIEventTypeSYNQNET_RX_FAILURE	SynqNet network data receive failure. Generated when the controller fails to receive the packet data buffer (Rincon DMA to internal memory) in two successive controller samples. A SYNQNET_RX_FAILURE is most likely caused by an incorrect RX_COPY_TIMER value (internal) or a timing problem. To recover from an RX_FAILURE event, the network must be shutdown and reinitialized. SYNQNET_RX_FAILURE is latched by the controller, use meiSynqNetEventReset(...) to clear the status/ event bit.
MEIEventTypeSYNQNET_TX_FAILURE	SynqNet network data transmission failure. Generated when the controller fails to transmit the packet data buffer in two successive controller samples. This occurs when the maximum foreground time exceeds the Tx time percentage of the controller's sample period. The default Tx time value is 75% of the controller's sample period. To correct Tx failures, either increase the Tx time or decrease the controller's sample rate. To recover from a TX_FAILURE event, the network must be shutdown and reinitialized. SYNQNET_TX_FAILURE is latched by the controller, use meiSynqNetEventReset(...) to

	clear the status/event bit.
MEIEventTypeSYNQNET_NODE_FAILURE	SynqNet node failure. Generated when any node's upstream or downstream packet error rate counters exceed the failure limit. The failure limits are configured with meiSqNodeConfigSet(...). Use meiSynqNetStatus(...) to read the nodeFailedMask to identify the failed nodes. Also, a SQNODE_NODE_FAILURE will be generated for each node that fails. SYNQNET_NODE_FAILURE is latched by the controller, use meiSynqNetEventReset(...) to clear the status/event bit. To recover from a node failure, the network must be shutdown and reinitialized. **See Also**: SynqNet Node Failure
MEIEventTypeSYNQNET_RECOVERY	SynqNet fault recovery. Generated when any node's upstream or downstream packet error rate counters exceed the fault limit and the data traffic is redirected around the fault. The fault limits are configurable via meiSqNodeConfigSet(...). SYNQNET_RECOVERY is latched by the controller. Use meiSynqNetEventReset(...) to clear the status/event bit.
MEIEventTypeSQNODE_IO_ABORT	SynqNet node I/O abort. Generated when the node I/O Abort is activated. When the I/O Abort is triggered, the node's outputs are disabled (set to the power-on condition). The node I/O Abort can be configured to trigger when either a Synq Lost occurs, Node Disable is active, a Power Fault occurs, or a User Fault is triggered. See MEISqNodeConfigIoAbort{.} for more details.
MEIEventTypeSQNODE_NODE_DISABLE	SynqNet node's Node Disable input is activated. Generated when the Node Disable input signal transitions from inactive to active. This signal is latched in hardware. Use meiSqNodeEventReset(...) to clear the status/event and the hardware latch.
MEIEventTypeSQNODE_NODE_ALARM	SynqNet node analog power failure. Generated when the node's power failure input bit transitions from inactive to active. The power fault circuit is node specific, but is typically connected to an analog power monitor. This signal is latched in hardware. Use meiSqNodeEventReset(...) to clear the status/event and the hardware latch.

MEIEventTypeSQNODE_ANALOG_POWER_FAULT	AnalogPowerFault may be used by a SynqNet node to indicate a problem with analog power rails. For example, the RMB-10V2 reports an AnalogPowerFault if its internal +15V or -15V (used to power the DACs and ADC) are below minimum values. **NOTE**: The exact meaning of AnalogPowerFault is specific to each particular node.
MEIEventTypeSQNODE_USER_FAULT	SynqNet node user fault. Generated when the node's user configurable fault is triggered. The user fault can be configured to monitor any controller memory address and compare the masked value to a specified pattern. This signal is latched by the controller, use meiSqNodeEventReset(...) to clear the status/ event bit.
MEIEventTypeSQNODE_NODE_FAILURE	SynqNet node failure. Generated when a node's upstream or downstream packet error rate counters exceed the failure limit. The failure limits are configured with meiSqNodeConfigSet(...). SQNODE_NODE_FAILURE is latched by the controller, use meiSqNodeEventReset(...) to clear the status/event bit. To recover from a node failure, the network must be shutdown and reinitialized.
MEIEventTypeSQNODE_IO_FAULT	The event indicates that a fault was detected when communicating with one of the slices or SynqNet I/O modules attached to either Slice or SQID nodes.This event is only generated by slice and SynqNet (SQID) I/O nodes.
MEIEventTypeCAN_BUS_STATE	The BusState has changed. Data[0] contains the new bus state.
MEIEventTypeCAN_RECEIVE_OVERRUN	The CAN hardware detected a receive overrun.
MEIEventTypeCAN_EMERGENGY	An emergency message was received from a node. Data[0] contains the node number. Data [1 to 4] contains the contents of the emergency message.
MEIEventTypeCAN_NODE_BOOT	A node boot message was received from a node. Data[0] contains the node number.
MEIEventTypeCAN_HEALTH	The health of a node has changed. Data[0] contains the node number. Data[1] contains the new node health.

MEIEventTypeCAN_DIGITAL_INPUT	A digital input event was received from a node. Data[0] contains the node number. Data [1 to 4] contains the new input state.
MEIEventTypeCAN_ANALOG_INPUT	An analog input event was received from a node. Data[0] contains the node number. Data [1 to 4] contains the new input state.

See Also

MPIEventMask | MPIEventMgr | MPINotify | MPIEventStatus | meiSynqNetEventReset | Error Limit and Limit Switch Errors

Use of MPIEventTypeENCODER_FAULT

MPIEventStatusINFO_COUNT_MAX

Definition

```
#define  MPIEventStatusINFO_COUNT_MAX (16)
```

Description

MEIEventStatusINFO_COUNT_MAX defines the size of the MPIEventStatus.info[] array.

See Also

MPIEventStatus | MPIEventMgr | MPINotify

EventMask Objects

Introduction

The **EventMask** determines which event types will be sent from the controller to the Notify object. By setting bits in the EventMask, specific events can be sent to the Notify object when they occur or by clearing specific bits, the events can be ignored. By default, no Events will occur until the EventMask is set.

The Event, EventMgr, and Notify objects are used with the EventMask to handle controller events.

Data Types

MPIEventMask	An array of unsigned longs

Macros

mpiEventMask**ALL** / meiEventMask**ALL**	Set all events within the event mask.
mpiEventMask**AND_ASSIGN**	Assign dst all events masked by both *src* and *dst*.
mpiEventMask**ASSIGN**	Assign the value of event mask *src* to the event mask *dst*.
mpiEventMask**AXIS** / meiEventMask**AXIS**	Set all MPIAxis events within the event mask.
mpiEventMask**BIT**	Set mask to only handle events of type *type*.
mpiEventMask**BitGET**	Reports if a mask is set to handle events of type type.
mpiEventMask**BitSET**	Sets mask to handle events of type *type*.
mpiEventMask**BIT_POSITION**	Returns the bit number that is associated with MPI/MEI event type.
mpiEventMask**BIT_POSITION_MASK**	Returns an element's bit-mask for the specified event type.
meiEventMask**CAN**	Set all MPICan events within the event mask.
mpiEventMask**CLEAR**	Set mask to handle no events.
mpiEventMask**COMPLEMENT**	Change the value of every bit within the event mask.
mpiEventMask**CONTROL**	
mpiEventMask**EXTERNAL**	Set external events within the event mask.
mpiEventMask**GET**	Reports if a mask is set to handle events of type *type*.
mpiEventMask**IS_CLEAR**	
mpiEventMask**IS_EQUAL**	Tests the equality of two event masks.

mpiEventMask**MOTION** / meiEventMask**MOTION**	Set all MPIMotion events within the event mask.
mpiEventMask**MOTOR** / meiEventMask**MOTOR**	Set all MPIMotor events within the event mask.
mpiEventMask**OR_ASSIGN**	Add all events masked by src to the event mask dst.
mpiEventMask**RECORDER**	Sets all MPIRecorder events within the event mask.
mpiEventMask**SET**	Set mask to handle events of type *type*.
mpiEventMask**SET_ALL**	Set mask to handle all events whose type is enumerated less than type.
meiEventMask**SYNQNET**	Set all MEISynqNet events within the event mask.
meiEventMask**SQNODE**	Set all MEISqNode events within the event mask.
mpiEventMask**WORD**	

Constants

MEIEventMask**BITS_IN_ELEMENT**	Define the number of bits in each data element of MPIEventMask.
MPIEventMask**ELEMENTS**	Define the number of data elements in a MPIEventMask.
MPIEventMask**ELEMENT_TYPE**	Define what the data type MPIEventMask is comprised of.

MPIEventMask

Definition

```
#define MPIEventMaskELEMENTS    (2)
typedef unsigned long MPIEventMaskELEMENT_TYPE;
typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS];
```

Description

MPIEventMask is an array of unsigned longs, with a length defined by MPIEventMaskELEMENTS. Each bit in the array represents a mask for a particular event. Be sure to always use the mpiEventMask (...) macros to set or clear the event masks.

See Also

MPIEventType

MPI EventMask Objects

mpiEventMaskALL / meiEventMaskALL

Declaration: mpiEventMaskALL

```
#define mpiEventMaskALL(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskALL is a macro that sets all the bits associated with MPI events in the event mask. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiXxxxxEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

Declaration: meiEventMaskALL

```
#define meiEventMaskALL(mask)
```

Required Header: stdmei.h
Change History: Modified in the 03.02.00

Description

meiEventMaskALL is a macro that sets all the bits associated with MEI events in the event mask. The MEI event types are defined in the MEIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventType | MPIEventMask

mpiEventMaskAND_ASSIGN

Declaration

```
#define mpiEventMaskAND_ASSIGN(dst,src)
```

Required Header: stdmpi.h

Description

mpiEventMaskAND_ASSIGN is a macro that bitwise ANDs all the bits associated with MPI/MEI events in the event mask ***src*** with ***dst*** and assigns the result to ***dst***. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

dst	An array of unsigned longs. Use MPIEventMask to declare the dst. Each bit in the array represents a mask for a particular event.
src	An array of unsigned longs. Use MPIEventMask to declare the src. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MPIEventMaskASSIGN | MPIEventMaskOR_ASSIGN

mpiEventMaskASSIGN

Declaration

```
#define mpiEventMaskASSIGN(dst,src)
```

Required Header: stdmpi.h

Description

mpiEventMaskASSIGN is a macro that assigns all the bits associated with MPI/MEI events in the event mask ***src*** to the event mask ***dst***. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

dst	An array of unsigned longs. Use MPIEventMask to declare the dst. Each bit in the array represents a mask for a particular event.
src	An array of unsigned longs. Use MPIEventMask to declare the src. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | mpiEventMaskOR_ASSIGN | mpiEventMaskAND_ASSIGN

mpiEventMaskAXIS / meiEventMaskAXIS

Declaration: mpiEventMaskAXIS

```
#define mpiEventMaskAXIS(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskAXIS is a macro that assigns all the bits associated with MPI Axis object events to the event ***mask***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpi*Object*EventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the ***mask***. Each bit in the array represents a mask for a particular event.

Declaration: meiEventMaskAXIS

```
#define meiEventMaskAXIS(mask)
```

Required Header: stdmei.h

Description

meiEventMaskAXIS is a macro that assigns all the bits associated with MEI Axis object events to the event ***mask***. The MEI event types are defined in the MEIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpi*Object*EventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the ***mask***. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MEIEventType

mpiEventMaskBIT

Declaration

```
#define mpiEventMaskBIT(mask,type)
```

Required Header: stdmpi.h

Description

mpiEventMaskBIT is a macro that assigns a bit associated with MPI/MEI event type in the event mask. The event types are defined in the MPIEventType and MEIEventType enumerations. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

See Also

MPIEventMask | MPIEventType | MEIEventType | mpiEventMaskBitGET | mpiEventMaskBitSET

mpiEventMaskBitGET

Declaration

```
#define mpiEventMaskBitGET(mask,bit)
```

Required Header: stdmpi.h

Description

mpiEventMaskBitGET is a macro that returns TRUE if the ***bit*** associated with a MPI/MEI event in the event ***mask*** is TRUE. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
bit	A bit number associated with an event type in an event mask.

Returns
TRUE if the specified bit is TRUE in the event mask. FALSE if the specified bit is FALSE in the event mask.

See Also

MPIEventMask | mpiEventMaskBIT | mpiEventMaskBitSET

mpiEventMaskBitSET

Declaration

```
#define mpiEventMaskBitSET(mask,bit,value)
```

Required Header: stdmpi.h

Description

mpiEventMaskBitSET is a macro that sets (value = TRUE) or clears (value = FALSE) the ***bit*** associated with a MPI/MEI event in the event ***mask***. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
bit	A bit number associated with an event type in an event mask.
value	TRUE to set a bit, FALSE to clear a bit.

See Also

MPIEventMask | mpiEventMaskBitGET | mpiEventMaskBIT

mpiEventMaskBIT_POSITION

Declaration

```
#define mpiEventMaskBIT_POSITION (type)
```

Required Header: stdmpi.h

Description

mpiEventMaskBIT_POSITION is a macro that returns the bit number that is associated with MPI/MEI event ***type***. The event types are defined in the MPIEventType and MEIEventType enumerations.

type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

Returns
A bit number associated with the event type.

See Also

MPIEventMask | MPIEventType | MEIEventType | mpiEventMaskBIT | mpiEventMaskBitGET | mpiEventMaskBitSET

mpiEventMaskBIT_POSITION_MASK

Declaration

```
#define mpiEventMaskBIT_POSITION_MASK (type)
```

Required Header: stdmpi.h

Description

mpiEventMaskBIT_POSITION_MASK is a macro that returns an event mask with a bit set that is associated with MPI/MEI event ***type***. The event types are defined in the MPIEventType and MEIEventType enumerations. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

Returns
An event mask. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MPIEventType | MEIEventType | mpiEventMaskBIT | mpiEventMaskBitGET | mpiEventMaskBitSET

meiEventMaskCAN

Declaration

```
#define meiEventMaskCAN(mask)
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiEventMaskCAN is a macro that assigns all the bits associated with MEI Can Object events through the event ***mask***.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

Returns
TRUE if the specified bit is TRUE in the event mask. FALSE if the specified bit is FALSE in the event mask.

See Also

Handling Can Events | MPIEventMask

mpiEventMaskCLEAR

Declaration

```
#define mpiEventMaskCLEAR(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskCLEAR is a macro that clears all the bits in an event ***mask***.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | mpiEventMaskIS_CLEAR | mpiEventMaskSET_ALL

mpiEventMaskCOMPLEMENT

Declaration

```
#define mpiEventMaskCOMPLEMENT(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskCOMPLEMENT is a macro that inverts all the bits in an event ***mask***. If a bit associated with an event type is TRUE, EventMaskCOMPLEMENT will set the bit to FALSE. And likewise, if a bit associated with an event type is FALSE, EventMaskCOMPLEMENT will set the bit to TRUE.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | mpiEventMaskCLEAR | mpiEventMaskSET_ALL

meiEventMaskCONTROL

Declaration

```
#define meiEventMaskCONTROL(mask)
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiEventMaskCONTROL is a macro that assigns all the bits associated with MPI Control object events to the event ***mask***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiControlEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

mpiControlEventNotifyGet | mpiControlEventNotifySet | mpiControlEventReset | MPIEventMask

mpiEventMaskEXTERNAL

Declaration

```
#define mpiEventMaskEXTERNAL(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskEXTERNAL is a macro that assigns all the bits associated with MPI External events to the event ***mask***. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask

mpiEventMaskGET

Declaration

```
#define mpiEventMaskGET(mask,type)
```

Required Header: stdmpi.h

Description

mpiEventMaskGET is a macro that returns TRUE if the bit associtated with the event ***type*** in the event ***mask*** is TRUE. The event types are defined in the MPIEventType and MEIEventType enumerations. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

Returns
TRUE if the bit associated with the event type is TRUE in the event mask. FALSE if the bit associated with the event type is FALSE in the event mask.

See Also

MPIEventMask | MPIEventType | MEIEventType | mpiEventMaskSET | mpiEventMaskSET_ALL

mpiEventMaskIS_CLEAR

Declaration

```
#define mpiEventMaskIS_CLEAR(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskIS_CLEAR is a macro that returns TRUE if all the bits in an event ***mask*** are FALSE.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

Returns
TRUE is all the bits in an event mask are FALSE. FALSE if any bit in an event mask is TRUE.

See Also

MPIEventMask | mpiEventMaskCLEAR | mpiEventMaskSET_ALL

mpiEventMaskIS_EQUAL

Declaration

```
#define mpiEventMaskIS_EQUAL(mask1,mask2)
```

Required Header: stdmpi.h

Description

mpiEventMaskIS_EQUAL is a macro that returns TRUE if event ***mask1*** is the same as event ***mask2***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask1	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
mask2	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

Returns
TRUE if event mask1 is the same as event mask2. FALSE if event mask1 is different from event mask2.

See Also

MPIEventMask | mpiEventMaskGET | mpiEventMaskSET

mpiEventMaskMOTION / meiEventMaskMOTION

Declaration: mpiEventMaskMOTION

```
#define mpiEventMaskMOTION(mask)
```

Required Header: stdmpi.h

Description

mpiEventMaskMOTION is a macro that assigns all the bits associated with MPI Motion object events to the event ***mask***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

Declaration: meiEventMaskMOTION

```
#define meiEventMaskMOTION(mask)
```

Required Header: stdmei.h

Description

meiEventMaskMOTION is a macro that assigns all the bits associated with MPI Motion object events to the event ***mask***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MEIEventType

mpiEventMaskMOTOR / meiEventMaskMOTOR

Declaration: mpiEventMaskMOTOR

```
#define mpiEventMaskMOTOR(mask)
                          mpiEventMaskBitSET((mask), MPIEventTypeAMP_FAULT, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeHOME, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeLIMIT_ERROR, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeLIMIT_HW_NEG, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeLIMIT_HW_POS, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeLIMIT_SW_NEG, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeLIMIT_SW_POS, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeENCODER_FAULT, 1), \
                          mpiEventMaskBitSET((mask), MPIEventTypeAMP_WARNING, 1)
```

Required Header: stdmpi.h

Declaration

mpiEventMaskMOTOR is a macro that assigns all the bits associated with MPI Motor object events to the event ***mask***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

Definition: meiEventMaskMOTOR

```
#define meiEventMaskMOTOR(mask)
```

Required Header: stdmei.h

Description

meiEventMaskMOTOR is a macro that assigns all the bits associated with MEI Motor object events to the event ***mask***. The MEI event types are defined in the MEIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MEIEventType

mpiEventMaskOR_ASSIGN

Declaration

```
#define mpiEventMaskOR_ASSIGN(dst,src)
```

Required Header: stdmpi.h

Description

mpiEventMaskOR_ASSIGN is a macro that bitwise ORs all the bits associated with MPI/MEI events in the event mask ***src*** with ***dst*** and assigns the result to ***dst***. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

dst	An array of unsigned longs. Use MPIEventMask to declare the dst. Each bit in the array represents a mask for a particular event.
src	An array of unsigned longs. Use MPIEventMask to declare the src. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MPIEventMaskASSIGN | MPIEventMaskAND_ASSIGN

mpiEventMaskRECORDER

Declaration

```
#define mpiEventMaskRECORDER(mask)  mpiEventMaskBitSET((mask),
                                        MPIEventTypeRECORDER_HIGH, 1), \
                                    mpiEventMaskBitSET((mask),
                                        MPIEventTypeRECORDER_FULL, 1), \
                                    mpiEventMaskBitSET((mask),
                                        MPIEventTypeRECORDER_DONE, 1)
```

Required Header: stdmpi.h

Description

mpiEventMaskRECORDER is a macro that assigns all the bits associated with MPI Recorder object events to the event ***mask***. The MPI event types are defined in the MPIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MPIEventType

mpiEventMaskSET

Declaration

```
#define mpiEventMaskSET(mask,type)
```

Required Header: stdmpi.h

Description

mpiEventMaskSET is a macro that sets the ***bit*** associated with MPI event type in the event ***mask***. The event types are defined in the MPIEventType and MEIEventType enumerations. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

See Also

MPIEventMask | mpiEventMaskGET | MPIEventType | MEIEventType | mpiEventMaskSET_ALL

mpiEventMaskSET_ALL

Declaration

```
#define mpiEventMaskSET_ALL(mask,type)
```

Required Header: stdmpi.h

Description

mpiEventMaskSET_ALL is a macro that sets all the ***bits*** associated with MPI/MEI event types that are less than the specified event ***type***. The event types are defined in the MPIEventType and MEIEventType enumerations. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.
type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

See Also

MPIEventMask | MPIEventType | MEIEventType | mpiEventMaskGET | mpiEventMaskSET

meiEventMaskSYNQNET

Declaration

```
#define meiEventMaskSYNQNET(mask)
```

Required Header: stdmei.h

Description

meiEventMaskSYNQNET is a macro that assigns all the bits associated with MEI SynqNet object events to the event mask. The MEI event types are defined in the MEIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MEIEventType

meiEventMaskSQNODE

Declaration

```
#define meiEventMaskSQNODE(mask)
```

Required Header: stdmei.h

Description

meiEventMaskSQNODE is a macro that assigns all the bits associated with MEI SqNode object events to the event ***mask***. The MEI event types are defined in the MEIEventType enumeration. After the event mask bits are initialized, the mask can be passed to a mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

mask	An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event.

See Also

MPIEventMask | MEIEventType

mpiEventMaskWORD

Declaration

```
#define mpiEventMaskWORD(type)
```

Required Header: stdmpi.h

Description

mpiEventMaskWORD is a macro that returns the word number that is associated with MPI/MEI event ***type***. The event types are defined in the MPIEventType and MEIEventType enumerations.

type	An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.

Returns
The word number associated with the event type.

See Also

MPIEventMask | | MPIEventType | MEIEventType | mpiEventMaskBIT_POSITION

MEIEventMaskBITS_IN_ELEMENT

Definition

```
#define MEIEventMaskBITS_IN_ELEMENT ((unsigned long)
                                      (sizeof(MPIEventMaskELEMENT_TYPE) * 8))
```

Description

MEIEventMaskBITS_IN_ELEMENT defines the number of bits in each data element of MPIEventMask.

NOTE: MEIEventMaskBITS_IN_ELEMENT replaced mpiEventMaskBITS_IN_ELEMENT.

See Also

MPIEventMask | mpiEventMaskBIT | mpiEventMaskBitGET | mpiEventMaskBitSET | mpiEventMaskBIT_POSITION_MASK

MPIEventMaskELEMENTS

Definition

```
#define MPIEventMaskELEMENTS (2)
```

Description

MPIEventMaskELEMENTS defines the number of data elements in a MPIEventMask.

See Also

MPIEventMask

MPIEventMaskELEMENT_TYPE

Definition

```
typedef unsigned long MPIEventMaskELEMENT_TYPE
```

Description

MPIEventMaskELEMENT_TYPE defines what the data type MPIEventMask is comprised of.

See Also

MPIEventMask

EventMgr Objects

Introduction

An **EventMgr** object manages the collection and distribution of event messages from the controller to the host. Events include normal motion completion, motor limits, fault conditions, data recorder buffer full, network node faults, etc. The EventNotify methods enable the application to request host notification for specified events, while ignoring other events. The EventMgr receives the event and then distributes the event via the Notify object to the tasks that are waiting for the event. Events that are faults are latched; the fault condition and the event must be cleared before the event can be generated again.

To collect events, the EventMgr must be serviced via the mpiEventMgrService(...) routine. The EventMgr can be polled by periodically by calling the mpiEventMgrService(...) routine or can be placed in an interrupt service routine. For your convenience, there is an apputil module that provides a serviceCreate(...) function that creates an EventMgr service routine for Windows OSs. The Service thread can be configured for polling or interrupts. Sources for the apputil module are included with the software distribution, so you can port it to other OSs.

The controller has a circular buffer which holds up to 128 event messages. If the EventMgr is not serviced within 128 events, event messages will be overwritten as new events occur. For interrupt driven EventMgr service threads, this is not an issue. For a polling EventMgr service thread, this could be an issue if the service thread does not have enough CPU cycles. In the worst-case scenario, events could be lost.

The best way to avoid lost events is to use an interrupt driven EventMgr service routine. If you're using polling, the next best thing is to make sure the EventMgr services the events at a high enough rate to avoid controller message buffer rollover. Determining the worst case EventMgr service latency and the maximum event message rate can be tricky. A simple method is to estimate the event message frequency and make sure each EventMgr can poll at least once for every 32 events (safety factor of 4). If you know that some events occur more frequently than others, then you may want to increase the polling frequency for EventMgrs that process the most frequent events and decrease the polling frequency for EventMgrs that process less frequently.

Generally, use only one EventMgr in your application. Do not use multiple interrupt driven EventMgr service threads to collect the same event. Suppose you configure an EventMgr to service Motion Done events from MS 0 and you configure another EventMgr with MEIEventMgrServiceConfig.allProcesses = TRUE. If one EventMgr collects the event message and acknowledges the interrupt before the other EventMgr can respond, it will miss the event.

If you want to monitor ALL events with MEIEventMgrServiceConfig.allProcesses = TRUE, then use polling (not interrupts). See the sample app EventLog.c for an example.

WARNING

Resetting the motion controller will cause any exising EventMgr objects to become unsynchronized with the motion controller, causing it to miss events. It is recommended that after a controller reset, an application should delete and re-create the EventMgr and Service objects. Then the application should reconnect all Notify objects to the EventMgr and re-enable event reporting by calling ***Object*EventNotifySet**.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiEventMgr**Create**	Create EventMgr object
mpiEventMgr**Delete**	Delete EventMgr object
mpiEventMgr**Validate**	Validate EventMgr object

Configuration and Information Methods

mpiEventMgr**ConfigGet**	Get EventMgr config
mpiEventMgr**ConfigSet**	Set EventMgr config
mpiEventMgr**Event**	Request event notification for all Notify objects on EventMgr's list
meiEventMgr**ServiceConfigGet**	Get processes that EventMgr will service
meiEventMgr**ServiceConfigSet**	Set processes that EventMgr will service

Action Methods

mpiEventMgr**Flush**	Flush pending EventMgr events
mpiEventMgr**Service**	Get list of all pending asynchronous events

Relational Methods

List Methods- for Control Objects

mpiEventMgr**Control**	Return handle of indexth Control object in list
mpiEventMgr**ControlAppend**	Append Control's handle to list
mpiEventMgr**ControlCount**	Count the number of Control objects associated with EventMgr (in list)
mpiEventMgr**ControlFirst**	Return handle to first Control object in list
mpiEventMgr**ControlIndex**	Return the index of a Control object in list
mpiEventMgr**ControlInsert**	Insert Control handle into list
mpiEventMgr**ControlLast**	Get handle to last Control object in list
mpiEventMgr**ControlListGet**	Get list of Control objects associated with EventMgr
mpiEventMgr**ControlListSet**	Create a list of Control objects associated with EventMgr
mpiEventMgr**ControlNext**	Get handle to next Control object in list
mpiEventMgr**ControlPrevious**	Get handle to previous Control object in list
mpiEventMgr**ControlRemove**	Remove a Control object's handle from list

List Methods- for Notify Objects

mpiEventMgr**Notify**	Return handle to a Notify object associated with EventMgr
mpiEventMgr**NotifyAppend**	Append Notify object to list
mpiEventMgr**NotifyCount**	Return number of Notify objects in list
mpiEventMgr**NotifyFirst**	Get first Notify object in list
mpiEventMgr**NotifyIndex**	Get index value for a Notify object in list

mpiEventMgr**NotifyInsert**	Place a Notify object after another Notify object in list
mpiEventMgr**NotifyLast**	Get handle to the Notify object that is last on the list
mpiEventMgr**NotifyListGet**	Get a list of Notify objects
mpiEventMgr**NotifyListSet**	Create a list of Notify objects
mpiEventMgr**NotifyNext**	Get the Notify object just after notify in list
mpiEventMgr**NotifyPrevious**	Get the Notify object just before notify in list
mpiEventMgr**NotifyRemove**	Remove a Notify object from list

Data Types

MPIEventMgr**Config**

MPIEventMgr**Message**

MEIEventMgr**ServiceConfig**

mpiEventMgrService

Declaration

```
long mpiEventMgrService(MPIEventMgr eventMgr,
                        MPIControl  control)
```

Required Header: stdmpi.h

Description

mpiEventMgrService obtains all pending asynchronous events from the motion controller (***control***).

Events generated by enabled sources are returned in First-In/First-Out (FIFO) order each time a thread calls mpiEventMgrService(...).

Typically, after a motion controller generates a hardware interrupt, the Interrupt Service Routine (ISR) is invoked, and the ISR in turn invokes mpiEventMgrService(...) directly or indirectly.

If "control" is	Then
MPIHandleVOID	events will be obtained from all motion controllers on the EventMgr's (***eventMgr***) control list
valid	***control*** must also be present on the EventMgr's (***eventMgr***) control list

Return Values	
MPIMessageOK	

See Also

meiEventMgrServiceConfigGet | meiEventMgrServiceConfigSet | mpiEventMgrFlush

mpiEventMgrCreate

Declaration

```
MPIEventMgr mpiEventMgrCreate(MPIControl control)
```

Required Header: stdmpi.h

Description

mpiEventMgrCreate creates an EventMgr object, with ***control*** as the initial element in the list of Control objects from which the EventMgr obtains asynchronous events (***control*** may be MPIHandleVOID).

EventMgrCreate is the equivalent of a C++ constructor.

Return Values	
handle	to an EventMgr object
MPIHandleVOID	if the object could not be created

See Also

mpiEventMgrDelete | mpiEventMgrValidate

mpiEventMgrDelete

Declaration

```
long mpiEventMgrDelete(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrDelete deletes an EventMgr object and invalidates its handle (***eventMgr***). EventMgrDelete is the equivalent of a C++ destructor.

Deleting an EventMgr object does not delete any of the Control objects that supply the EventMgr with asynchronous events. However, deleting an EventMgr object will delete any unreceived events for that EventMgr.

Return Values	
MPIMessageOK	

See Also

mpiEventMgrCreate | mpiEventMgrValidate

mpiEventMgrValidate

Declaration

```
long mpiEventMgrValidate(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrValidate validates an EventMgr object and its handle (***eventMgr***).

Return Values	
MPIMessageOK	

See Also

mpiEventMgrCreate | mpiEventMgrDelete

mpiEventMgrConfigGet

Declaration

```
long mpiEventMgrConfigGet(MPIEventMgr       eventMgr,
                          MPIEventMgrConfig *config,
                          void              *external)
```

Required Header: stdmpi.h

Description

mpiEventMgrConfigGet gets the configuration of an ***EventMgr*** object (eventMgr) and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

mpiEventMgrConfigSet

mpiEventMgrConfigSet

Declaration

```
long mpiEventMgrConfigSet(MPIEventMgr       eventMgr,
                          MPIEventMgrConfig *config,
                          void              *external)
```

Required Header: stdmpi.h

Description

mpiEventMgrConfigSet sets (writes) the ***eventMgr*** configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

mpiEventMgrConfigGet

mpiEventMgrEvent

Declaration

```
long mpiEventMgrEvent(MPIEventMgr    eventMgr,
                      MPIEventStatus *status)
```

Required Header: stdmpi.h

Description

mpiEventMgrEvent requests that the EventMgr (***eventMgr***) call mpiNotifyEvent(notify, status) for all Notify objects on that Event Manager's list. Each Notify object represents a thread that has requested event notification. The Notify object will determine whether it accepts the event notification or not.

EventMgrEvent enables your application to use the Event Manager to distribute *user-created events* in the same way that events generated by the motion controller are distributed.

Return Values	
MPIMessageOK	

See Also

meiEventMgrServiceConfigGet

Declaration

```
long meiEventMgrServiceConfigGet(MPIEventMgr                eventMgr,
                                 MEIEventMgrServiceConfig *config)
```

Required Header: stdmpi.h

Description

meiEventMgrServiceConfigGet gets the configuration of an ***EventMgr*** object (eventMgr) and writes it into the structure pointed to by ***config***.

Return Values	
MPIMessageOK	

See Also

meiEventMgrServiceConfigSet | mpiEventMgrService

meiEventMgrServiceConfigSet

Declaration

```
long meiEventMgrServiceConfigGet(MPIEventMgr                eventMgr,
                                 MEIEventMgrServiceConfig *config)
```

Required Header: stdmpi.h

Description

meiEventMgrServiceConfigSet sets (writes) the flash configuration for an EventMgr object (***eventMgr***) using data from the structure pointed to by ***config***.

Return Values	
MPIMessageOK	

See Also

meiEventMgrServiceConfigGet | mpiEventMgrService

mpiEventMgrFlush

Declaration

```
long mpiEventMgrFlush(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrFlush flushes any pending events from an EventMgr (***eventMgr***).

Return Values	
MPIMessageOK	

See Also

mpiEventMgrControl

Declaration

```
MPIControl mpiEventMgrControl(MPIEventMgr eventMgr,
                              long        index)
```

Required Header: stdmpi.h

Description

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

eventMgr	a handle to the EventMgr object.
index	a position in the list.

Return Values	
MPIMessageOK	
handle	to the ***index***th motion controller (Control) associated with an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***index*** is less than 0 if ***index*** is greater than or equal to **mpiEventMgrCount(eventMgr)**
MPIMessageARG_INVALID	
MEIListMessageELEMENT_NOT_FOUND	
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrControlAppend

Declaration

```
long mpiEventMgrControlAppend(MPIEventMgr eventMgr,
                              MPIControl  control)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlAppend appends ***control*** to the list of motion controllers associated with an EventMgr (***eventMgr***). Add "control" to the end of the list.

eventMgr	a handle to the EventMgr object.
control	a handle to a Control object.

Return Values	
MPIMessageOK	
MPIMessageHANDLE_INVALID	
MPIMessageOBJECT_ON_LIST	
MPIMessageNO_MEMORY	

See Also

mpiEventMgrControlCount

Declaration

```
long mpiEventMgrControlCount(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlCount returns the number of elements on the list.

eventMgr	a handle to the EventMgr object.

Return Values	
number	of motion controllers associated with an EventMgr (***eventMgr***)
-1	if ***eventMgr*** is an invalid handle
0	if ***eventMgr*** has no associated motion controllers

See Also

mpiEventMgrControlFirst

Declaration

```
MPIControl mpiEventMgrControlFirst(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlFirst returns the first element in the list. This function can be used in conjuntion with mpiEventMgrControlNext(...) in order to iterate through the list. MPIHandleVOID is returned if the list is empty.

eventMgr	a handle to the EventMgr object.

Return Values	
handle	to the first motion controller (Control) associated with an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***eventMgr*** has no associated motion controllers
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrControlLast

mpiEventMgrControlIndex

Declaration

```
long mpiEventMgrControlIndex(MPIEventMgr eventMgr,
                             MPIControl  control)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlIndex returns the position of "control" on the list.

eventMgr	a handle to the EventMgr object.
control	a handle to a Control object.

Return Values	
index	of ***control*** in the list of motion controllers associated with an EventMgr (***eventMgr***)
-1	if ***eventMgr*** is invalid if ***control*** was not found in the list of motion controllers

See Also

mpiEventMgrControlInsert

Declaration

```
long mpiEventMgrControlInsert(MPIEventMgr eventMgr,
                              MPIControl  control,
                              MPIControl  insert)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlInsert inserts a Control object (***insert***) after the Control object (***control***) in the list of motion controllers associated with ***eventMgr***.

eventMgr	a handle to the EventMgr object.

Return Values	
MPIMessageOK	if *EventMgrControlInsert* successfully inserts the Control object (***insert***) after another Control object (***control***) in the list of motion controllers

See Also

mpiEventMgrControlLast

Declaration

```
MPIControl mpiEventMgrControlLast(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

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

eventMgr	a handle to the EventMgr object.

Return Values	
handle	to the last motion controller (Control) associated with an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***eventMgr*** has no associated motion controllers
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrControlFirst

mpiEventMgrControlListGet

Declaration

```
long mpiEventMgrControlListGet(MPIEventMgr eventMgr,
                               long        *controlCount,
                               MPIControl  *controlList)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlListGet returns the list of Control objects associated with an EventMgr object (***eventMgr***).

EventMgrControlListGet also writes the number of Control handles (in the list) to the location pointed to by ***controlCount***, and writes an array (of ***controlCount*** Control handles) to the location pointed to by ***controlList***.

eventMgr	a handle to the EventMgr object.

Return Values	
MPIMessageOK	

See Also

mpiEventMgrControlListSet

mpiEventMgrControlListSet

Declaration

```
long mpiEventMgrControlListSet(MPIEventMgr eventMgr,
                               long        controlCount,
                               MPIControl  *controlList)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlListSet creates a list of ***controlCount*** Control objects using the Control handles specified by ***controlList***. Any existing motion controller list is completely replaced.

The ***controlList*** parameter is the address of an array of ***controlCount*** Control handles, or is NULL (if ***controlCount*** is equal to zero).

A motion controller list can also be created incrementally (i.e., one Control at a time) by using the append and/or insert methods described in this section. The initial Control of a control list may be specified using the ***control*** parameter of mpiEventMgrCreate(...). The list methods in this section can be used to examine and manipulate a motion controller list regardless of how the list was created.

Return Values	
MPIMessageOK	

See Also

mpiEventMgrControlListGet

mpiEventMgrControlNext

Declaration

```
MPIControl mpiEventMgrControlNext(MPIEventMgr  eventMgr,
                                  MPIControl   control)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlNext returns the next element following "control" on the list. MPIHandleVOID is returned if "control" is the last element on the list, or if "control" is not in the list at all. This function can be used in conjuntion with mpiEventMgrControlFirst() in order to iterate through the list.

eventMgr	a handle to the EventMgr object.
control	a handle to a Control object.

Return Values	
handle	to the motion controller following ***control*** in the list of motion controllers associated with an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***control*** is the last motion controller
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrControlPrevious

mpiEventMgrControlPrevious

Declaration

```
MPIControl  mpiEventMgrControlPrevious(MPIEventMgr eventMgr,
                                       MPIControl  control)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlPrevious returns the previous element prior to "control" on the list. MPIHandleVOID is returned if "control" is the first element on the list, or if "control" is not in the list at all. This function can be used in conjuntion with mpiEventMgrControlLast(...) in order to iterate through the list backwards.

eventMgr	a handle to the EventMgr object.
control	a handle to a Control object.

Return Values	
handle	to the motion controller preceding ***control*** in the list of motion controllers associated with an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***control*** is the first motion controller
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrControlNext

mpiEventMgrControlRemove

Declaration

```
long mpiEventMgrControlRemove(MPIEventMgr eventMgr,
                              MPIControl  control)
```

Required Header: stdmpi.h

Description

mpiEventMgrControlRemove removes a Control object (***control***) from the list of Control objects associated with ***eventMgr***.

eventMgr	a handle to the EventMgr object.
control	a handle to a Control object.

Return Values	
MPIMessageOK	if *EventMgrControlRemove* successfully removes a Control object from the list of Control objects associated with an EventMgr (***eventMgr***)

See Also

mpiEventMgrControl

mpiEventMgrNotify

Declaration

```
MPINotify mpiEventMgrNotify(MPIEventMgr eventMgr,
                            long        index)
```

Required Header: stdmpi.h

Description

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

eventMgr	a handle to the EventMgr object.
index	a position in the list.

Return Values	
handle	to the ***index***th Notify object associated with an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***index*** is less than 0 if ***index*** is greater than or equal to **mpiEventMgrCount(eventMgr)**
MPIMessageARG_INVALID	
MEIListMessageELEMENT_NOT_FOUND	
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrNotifyAppend

Declaration

```
long mpiEventMgrNotifyAppend(MPIEventMgr eventMgr,
                             MPINotify   notify)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyAppend appends a Notify object (***notify***) to the list of Notify objects maintained by an EventMgr object (***eventMgr***).

eventMgr	a handle to the EventMgr object.
notify	a handle to a Notify object.

Return Values	
MPIMessageOK	
MPIMessageHANDLE_INVALID	
MPIMessageOBJECT_ON_LIST	
MPIMessageNO_MEMORY	

See Also

mpiEventMgrNotifyCount

Declaration

```
long mpiEventMgrNotifyCount(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyCount returns the number of elements on the list.

eventMgr	a handle to the EventMgr object.

Return Values	
number	of Notify objects in the list (of Notify objects) maintained by an EventMgr (***eventMgr***)
-1	if ***eventMgr*** is invalid
0	if the list (of Notify objects) is empty

See Also

mpiEventMgrNotifyFirst

Declaration

```
MPINotify mpiEventMgrNotifyFirst(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyFirst returns the first element in the list. This function can be used in conjuntion with mpiEventMgrNotifyNext(...) in order to iterate through the list.

eventMgr	a handle to the EventMgr object.

Return Values	
handle	to the first Notify object in the list (of Notify objects) maintained by an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if the list (of Notify objects) is empty
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrNotifyLast | mpiEventMgrNotifyNext

mpiEventMgrNotifyIndex

Declaration

```
long mpiEventMgrNotifyIndex(MPIEventMgr eventMgr,
                            MPINotify   notify)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyIndex returns the position of ***notify*** on the list.

eventMgr	a handle to the EventMgr object.
notify	a handle to the EventMgr object.

Return Values	
index	of ***notify*** in the list (of Notify objects) maintained by an EventMgr (***eventMgr***)
-1	if eventMgr is invalid if ***notify*** was not found in the list

See Also

mpiEventMgrNotifyInsert

Declaration

```
long mpiEventMgrNotifyInsert(MPIEventMgr eventMgr,
                             MPINotify   notify,
                             MPINotify   insert)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyInsert places a Notify object (***insert***) after another Notify object (***notify***) in the list (of Notify objects) maintained by an EventMgr object (***eventMgr***).

Return Values	
MPIMessageOK	

See Also

mpiEventMgrNotifyLast

Declaration

```
MPINotify mpiEventMgrNotifyLast(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

Description

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

eventMgr	a handle to the EventMgr object.

Return Values	
handle	to the first Notify object in the list (of Notify objects) maintained by an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if the list (of Notify objects) is empty
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrNotifyFirst | mpiEventMgrNotifyPrevious

mpiEventMgrNotifyListGet

Declaration

```
long mpiEventMgrNotifyListGet(MPIEventMgr eventMgr,
                              long        *notifyCount,
                              MPINotify   *notifyList)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyListGet returns the list of Notify objects for an EventMgr object (***eventMgr***). *EventMgrNotifyListGet* also sets (writes) the number (of Notify objects in the list) to the location pointed to by ***notifyCount***, and sets (writes) an array (of ***notifyCount*** Notify handles) to the contents of the location pointed to by ***notifyList***.

Return Values	
MPIMessageOK	

See Also

mpiEventMgrNotifyListSet

mpiEventMgrNotifyListSet

Declaration

```
long mpiEventMgrNotifyListSet(MPIEventMgr  eventMgr,
                              long         notifyCount,
                              MPINotify    *notifyList)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyListSet creates a list of Notify objects, where the number of Notify objects is specified by ***notifyCount***, using the Notify handles specified by ***notifyList***. The ***notifyList*** argument is the address of an array of ***notifyCount*** Notify handles, or is NULL if ***notifyCount*** is zero. Any existing notify object list is completely replaced.

You can also create a Notify object list incrementally (i.e., one Notify object is created at a time), by using the EventMgrAppend and/or EventMgrInsert methods. After creating a list of Notify objects, use the EventMgrList methods to examine and manipulate the list, regardless of how the list was created.

Return Values	
MPIMessageOK	

See Also

mpiEventMgrNotifyListGet

mpiEventMgrNotifyNext

Declaration

```
MPINotify mpiEventMgrNotifyNext(MPIEventMgr eventMgr,
                                MPINotify   notify)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyNext returns the next element following ***notify*** on the list. This function can be used in conjuntion with mpiEventMgrNotifyFirst(...) in order to iterate through the list.

eventMgr	a handle to the EventMgr object.
notify	a handle to a Notify object.

Return Values	
handle	to the first Notify object in the list (of Notify objects) maintained by an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if the list (of Notify objects) is empty
MPIMessageHANDLE_INVALID	

See Also

mpiEventMgrNotifyPrevious | mpiEventMgrNotifyFirst

mpiEventMgrNotifyPrevious

Declaration

```
MPINotify mpiEventMgrNotifyPrevious(MPIEventMgr eventMgr,
                                    MPINotify   notify)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyPrevious returns the previous element prior to ***notify*** on the list. This function can be used in conjuntion with mpiEventMgrNotifyLast(...) in order to iterate through the list backwards.

eventMgr	a handle to the EventMgr object.
notify	a handle to a Notify object.

Return Values	
handle	to the Notify object just before another Notify object (***notify***) in the list (of Notify objects) maintained by an EventMgr (***eventMgr***)
MPIHandleVOID	if ***eventMgr*** is invalid if ***notify*** is the first Notify object in the list
MPIMessageHANDLE_INVALID	either ***eventMgr*** or ***notify*** is an invalid handle

See Also

mpiEventMgrNotifyNext | mpiEventMgrNotifyLast

mpiEventMgrNotifyRemove

Declaration

```
long mpiEventMgrNotifyRemove(MPIEventMgr eventMgr,
                             MPINotify   notify)
```

Required Header: stdmpi.h

Description

mpiEventMgrNotifyRemove removes a Notify object (***notify***) from the list of Notify objects maintained by an EventMgr object (***eventMgr***).

eventMgr	a handle to the EventMgr object.
notify	a handle to a Notify object.

Return Values	
MPIMessageOK	

See Also

MPIEventMgrConfig

Definition

```
typedef   MPIEmpty  MPIEventMgrConfig;
```

Change History: Modified in the 03.03.00

Description

MPIEventMgrConfig is an empty structure. It is a placeholder for possible future EventMgr configurations.

See Also

MPIEventMgrMessage

Definition

```
typedef enum {

    MPIEventMgrMessageEVENTMGR_INVALID,
} MPIEventMgrMessage;
```

Description

MPIEventMessage is an enumeration of the EventMgr error messages that can be returned by the MPI library.

MPIEventMgrMessageEVENTMGR_INVALID

Not supported.

Sample Code

```
MPIControl      control;
MPIEventMgr     eventMgr;
long            returnValue;
.

eventMgr =
        mpiEventMgrCreate(control);
returnValue =
        mpiEventMgrValidate(eventMgr);
```

See Also

MPIEventMgr | mpiEventMgrCreate | mpiEventMgrValidate

MEIEventMgrServiceConfig

Definition

```
typedef struct MEIEventMgrServiceConfig {
   MPI_BOOL  allProcesses; /* TRUE => collect
                              events from all processes,
                              else EventMgr process only */
} MEIEventMgrServiceConfig;
```

Change History: Modified in the 03.03.00

Description

allProcesses	is a boolean value. If allProcesses=TRUE, then the event manager will handle events originating from all processes. If allProcesses=FALSE, then the event manager will only handle events originating from the same process in which the event manager was created.

See Also

Filter Objects

Introduction

A **Filter** object manages a single filter on a controller. It represents the control algorithm used to control a motor in a closed-loop system. The Filter contains an algorithm, a set of coefficients, inputs, and an output. Its primary responsibility is to take the difference between the command and actual positions and then calculate the output based on the control algorithm and coefficients.

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

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiFilter**Create**	Create Filter object
mpiFilter**Delete**	Delete Filter object
mpiFilter**Validate**	Validate Filter object

Configuration and Information Methods

mpiFilter**ConfigGet**	Get Filter configuration
mpiFilter**ConfigSet**	Set Filter configuration
mpiFilter**FlashConfigGet**	Get flash configuration for Filter
mpiFilter**FlashConfigSet**	Set flash configuration for Filter
mpiFilter**GainGet**	Get gain coefficients
mpiFilter**GainSet**	Set current gain index
mpiFilter**GainIndexGet**	Get current gain index
mpiFilter**GainIndexSet**	Set current gain index

Memory Methods

mpiFilter**Memory**	Get address to Filter memory
mpiFilter**MemoryGet**	Copy data from Filter memory to application memory
mpiFilter**MemorySet**	Copy data from application memory to Filter memory

Relational Methods

mpiFilter**AxisMapGet**	Get object map of axes associated with Filter
mpiFilter**AxisMapSet**	Set axes associated with Filter
mpiFilter**Control**	Return handle of Control that is assoiciated with Filter
mpiFilter**MotorMapGet**	Get object map of Motors associated with Filter
mpiFilter**MotorMapSet**	Set Motors to be associated with Filter

mpiFilter**Number**	Get index of Filter (for Control list)

Action Methods

mpiFilter**IntergratorReset**	Reset the integrators of filter.

Postfilter Methods

meiFilter**PostfilterGet**	Reads postfilter information.
meiFilter**PostfilterSet**	Writes postfilter information.
meiFilter**PostfilterSectionGet**	Reads postfilter section information.
meiFilter**PostfilterSectionSet**	Writes postfilter section information.

Data Types

MPIFilter**Coeff**
MPIFilter**Config** / MEIFilter**Config**
MEIFilter**Form**
MPIFilter**Gain**
MEIFilter**GainIndex**
MEIFilter**GainPID**
MEIFilter**GainPIDCoeff**
MEIFilter**GainPIV**
MEIFilter**GainPIVCoeff**
MEIFilter**GainTypePID**
MEIFilter**GainTypePIV**
MPIFilter**Message**
MEIFilter**Type**
MEI**PostfilterSection**

Constants

MPIFilter**CoeffCOUNT_MAX**
MPIFilter**GainCOUNT_MAX**
MEI**MaxBiQuadSections**

mpiFilterCreate

Declaration

```
MPIFilter mpiFilterCreate(MPIControl  control,
                          long        number)
```

Required Header: stdmpi.h

Description

mpiFilterCreate creates a Filter object associated with a filter (***number***), that is located on a motion controller (***control***). FilterCreate is the equivalent of a C++ constructor.

Return Values	
handle	to an Filter object
MPIHandleVOID	if the Filter object could not be created

See Also

mpiFilterDelete | mpiFilterValidate

mpiFilterDelete

Declaration

```
long mpiFilterDelete(MPIFilter filter)
```

Required Header: stdmpi.h

Description

mpiFilterDelete deletes a Filter object and invalidates its handle (***filter***). FilterDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiFilterCreate | mpiFilterValidate

mpiFilterValidate

Declaration

```
long mpiFilterValidate(MPIFilter filter)
```

Required Header: stdmpi.h

Description

mpiFilterValidate validates the Filter object and its handle (***filter***).

Return Values	
MPIMessageOK	

See Also

mpiFilterCreate | mpiFilterDelete

mpiFilterConfigGet

Declaration

```
long mpiFilterConfigGet(MPIFilter       filter,
                        MPIFilterConfig *config,
                        void            *external)
```

Required Header: stdmpi.h

Description

mpiFilterConfigGet gets a Filter's (***filter***) configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Filter's configuration information in ***external*** is in addition to the Filter's configuration information in ***config***, i.e, the Filter's 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 **MEIFilterConfig{}** or is NULL.

Return Values	
MPIMessageOK	

See Also

mpiFilterConfigSet | MEIFilterConfig

mpiFilterConfigSet

Declaration

```
long mpiFilterConfigSet(MPIFilter       filter,
                        MPIFilterConfig *config,
                        void            *external)
```

Required Header: stdmpi.h

Description

mpiFilterConfigGet sets a Filter's (***filter***) configuration using data from the structure pointed to by ***config***, and from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Filter's configuration information in ***external*** is in addition to the Filter's configuration information in ***config***, i.e, the Filter's 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 MEIFilterConfig{} or is NULL.

Return Values	
MPIMessageOK	

See Also

mpiFilterConfigGet | MEIFilterConfig

mpiFilterFlashConfigGet

Declaration

```
long mpiFilterFlashConfigGet(MPIFilter       filter,
                             void            *flash,
                             MPIFilterConfig *config,
                             void            *external)
```

Required Header: stdmpi.h

Description

mpiFilterFlashConfigGet gets a Filter's (***filter***) flash configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Filter's flash configuration information in ***external*** is in addition to the Filter'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).

Remarks

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

filter	a handle to a Filter 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 filter object of type MPIFilterConfig.
***external**	a pointer to a configuration structure for the filter object of type MEIFilterConfig.

Return Values	
MPIMessageOK	

See Also

MEIFlash | mpiFilterFlashConfigSet | MEIFilterConfig

mpiFilterFlashConfigSet

Declaration

```
long mpiFilterFlashConfigSet(MPIFilter       filter,
                             void            *flash,
                             MPIFilterConfig *config,
                             void            *external)
```

Required Header: stdmpi.h

Description

mpiFilterFlashConfigSet sets a Filter's (***filter***) flash configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Filter's flash configuration information in ***external*** is in addition to the Filter'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).

Remarks

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

filter	a handle to a Filter 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 filter object of type MPIFilterConfig.
***external**	a pointer to a configuration structure for the filter object of type MEIFilterConfig.

Return Values	
MPIMessageOK	

See Also

MEIFlash | mpiFilterFlashConfigGet | MEIFilterConfig

mpiFilterGainGet

Declaration

```
long mpiFilterGainGet(MPIFilter     filter,
                      long          gainIndex,
                      MPIFilterGain *gain)
```

Required Header: stdmpi.h

Description

mpiFilterGainGet gets the gain coefficients of a Filter (***filter***, for the gain index specified by ***gainIndex***) and writes them into the structure pointed to by ***gain***. Post filters are set using the meiFilterPostfilterGet/Set and meiFilterPostfilterSectionGet/Set methods.

filter	A handle to a Filter object.
gainIndex	The index number of the filter gain table. Unless you are using gain tables, set this value to 0. If you are setting gain tables in Motion Console, this is the drop down box in Filter Summary->Coeffs that says "Gain Table 0", "Gain Table 1", etc.
***gain**	Pointer to the gain structure. The gain structure holds the closed loop filter gains. The PID and PIV gains are both set here. The only difference in setting PID vs. PIV gains are the names for the gain indices (see PID example below).

Return Values	
MPIMessageOK	

Sample Code

```
/* Sets reasonable tuning parameters for a Trust TA9000 test stand */
void setPIDs(MPIFilter filter)
{
   MPIFilterGain gain;
   long returnValue;

   returnValue = mpiFilterGainGet(filter, 0, &gain);
   msgCHECK(returnValue);

   gain.coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_POSITION].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY].f = (float)45;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION].f = (float)101000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION].f = (float)450;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING].f = (float)15000;
```

```
    gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_REST].f = (float)15000;
    gain.coeff[MEIFilterGainPIDCoeffDRATE].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMIT].f = (float)32767;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH].f = (float)32767;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITLOW].f = (float)-32767;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_OFFSET].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffNOISE_POSITIONFFT].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffNOISE_FILTERFFT].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffNOISE_VELOCITYFFT].f = (float)0;

    returnValue = mpiFilterGainSet(filter, 0, &gain);
    msgCHECK(returnValue);
}
```

Another way to change filter coefficients is to use mpiFilterConfigGet /Set.

```
    returnValue = mpiFilterConfigGet(filter, &config, NULL);
    msgCHECK(returnValue);

    /*
      Look in MEIFilterGainPIDCoeff to get the indexes.
      Not all of the above coefficients are shown in this short example.
    */

  config.gain[0].coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
  config.gain[0].coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
  config.gain[0].coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;

  returnValue = mpiFilterConfigSet(filter, &config, NULL);
  msgCHECK(returnValue);
```

See Also

mpiFilterGainSet | mpiFilterConfigGet | mpiFilterConfigSet | meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet

mpiFilterGainSet

Declaration

```
long mpiFilterGainSet(MPIFilter     filter,
                      long          gainIndex,
                      MPIFilterGain *gain)
```

Required Header: stdmpi.h

Description

mpiFilterGainSet sets the gain coefficients of a Filter (***filter***, for the gain index specified by ***gainIndex***) using data from the structure pointed to by ***gain***. Post filters are set using the meiFilterPostfilterGet/Set and meiFilterPostfilterSectionGet/Set methods.

filter	A handle to a Filter object.
gainIndex	The index number of the filter gain table. Unless you are using gain tables, set this value to 0. If you are setting gain tables in Motion Console, this is the drop down box in Filter Summary->Coeffs that says "Gain Table 0", "Gain Table 1", etc.
***gain**	Pointer to the gain structure. The gain structure holds the closed loop filter gains. The PID and PIV gains are both set here. The only difference in setting PID vs. PIV gains are the names for the gain indices (see PID example below).

Return Values	
MPIMessageOK	

Sample Code

```
/* Sets reasonable tuning parameters for a Trust TA9000 test stand */
void setPIDs(MPIFilter filter)
{
   MPIFilterGain gain;
   long returnValue;

   returnValue = mpiFilterGainGet(filter, 0, &gain);
   msgCHECK(returnValue);

   gain.coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_POSITION].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY].f = (float)45;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION].f = (float)101000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION].f = (float)450;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_REST].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffDRATE].f = (float)0;
```

```
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMIT].f = (float)32767;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH].f = (float)32767;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITLOW].f = (float)-32767;
    gain.coeff[MEIFilterGainPIDCoeffOUTPUT_OFFSET].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffNOISE_POSITIONFFT].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffNOISE_FILTERFFT].f = (float)0;
    gain.coeff[MEIFilterGainPIDCoeffNOISE_VELOCITYFFT].f = (float)0;

    returnValue = mpiFilterGainSet(filter, 0, &gain);
    msgCHECK(returnValue);
}
```

Another way to change filter coefficients is to use mpiFilterConfigGet /Set.

```
    returnValue = mpiFilterConfigGet(filter, &config, NULL);
    msgCHECK(returnValue);

    /*
      Look in MEIFilterGainPIDCoeff to get the indexes.
      Not all of the above coefficients are shown in this short example.
    */

  config.gain[0].coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
  config.gain[0].coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
  config.gain[0].coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;

  returnValue = mpiFilterConfigSet(filter, &config, NULL);
  msgCHECK(returnValue);
```

See Also

mpiFilterGainGet | mpiFilterConfigGet | mpiFilterConfigSet | meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet

mpiFilterGainIndexGet

Declaration

```
long mpiFilterGainIndexGet(MPIFilter filter,
                           long      *gainIndex)
```

Required Header: stdmpi.h

Description

mpiFilterGainIndexGet gets the current gain index of a Filter (***filter***) and writes it to the location pointed to by ***gainIndex***. Reading the gain index tells you what gain table is being used currently.

If the filter is in state MEIXmpSwitchType MEIXmpSwitchTypeMOTION_ONLY, the gain index is automatically changed by the firmware as described at MEIXmpSwitchType. When the filter is in state MEIXmpSwitchType MEIXmpSwitchTypeNONE, the gain index is controlled by the user.

Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV).

Return Values	
MPIMessageOK	

See Also

MPIFilterConfig | mpiFilterConfigGet | mpiFilterConfigSet | MEIFilterGainIndex | MEIXmpSwitchType | mpiFilterGainIndexSet | mpiFilterGainGet | mpiFilterGainSet

mpiFilterGainIndexSet

Declaration

```
long mpiFilterGainIndexSet(MPIFilter   filter,
                           long        gainIndex)
```

Required Header: stdmpi.h

Description

mpiFilterGainIndexSet sets the current gain index of a Filter (***filter***) to ***gainIndex***. Writing the gain index controls what gain table is currently being used.

If the filter is in state MEIXmpSwitchType **MEIXmpSwitchTypeMOTION_ONLY**, the gain index is changed automatically by the firmware as described at MEIXmpSwitchType. Be aware that the filter can change the gain index in real-time, thereby overwriting your changes in this mode.

When the filter is in state MEIXmpSwitchType **MEIXmpSwitchTypeNONE**, the gain index is controlled by the user. This is the normal state when using FilterGainIndexSet(...). Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV).

Return Values	
MPIMessageOK	

See Also

MPIFilterConfig | mpiFilterConfigGet | mpiFilterConfigSet | MEIFilterGainIndex | MEIXmpSwitchType | mpiFilterGainIndexGet | mpiFilterGainGet | mpiFilterGainSet

mpiFilterMemory

Declaration

```
long mpiFilterMemory(MPIFilter  filter,
                     void       **memory)
```

Required Header: stdmpi.h

Description

mpiFilterMemory writes an address, which is used to access a Filter's (***filter***) memory to the contents of ***memory***. This address, or an address calculated from it, can be passed as the src parameter to **MPIFilterMemoryGet(...)** and as the ***dst*** parameter to **MPIFilterMemorySet(...)**.

Return Values	
MPIMessageOK	

See Also

mpiFilterMemoryGet | mpiFilterMemorySet

mpiFilterMemoryGet

Declaration

```
long mpiFilterMemoryGet(MPIFilter     filter,
                        void          *dst,
                        const  void   *src,
                        long          count)
```

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

Description

mpiFilterMemoryGet copies ***count*** bytes of a Filter's (***filter***) memory (starting at address src) and writes them into application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiFilterMemorySet | mpiFilterMemory

mpiFilterMemorySet

Declaration

```
long mpiFilterMemorySet(MPIFilter     filter,
                        void          *dst,
                        const  void   *src,
                        long          count)
```

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

Description

mpiFilterMemorySet copies ***count*** bytes of application memory (starting at address ***src***) and writes them into a Filter's (***filter***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiFilterMemorySet | mpiFilterMemory

mpiFilterAxisMapGet

Declaration

```
long mpiFilterAxisMapGet(MPIFilter    filter,
                         MPIObjectMap *axisMap)
```

Required Header: stdmpi.h

Description

mpiFilterAxisMapGet gets the object map of the Axes that are associated with a Filter (***filter***), and writes it into the structure pointed to by ***axisMap***.

Return Values	
MPIMessageOK	

See Also

mpiFilterAxisMapSet

mpiFilterAxisMapSet

Declaration

```
long mpiFilterAxisMapSet(MPIFilter    filter,
                         MPIObjectMap axisMap)
```

Required Header: stdmpi.h

Description

mpiFilterAxisMapSet sets the Axes associated with a Filter (***filter***), using data from the object map specified by ***axisMap***.

Return Values	
MPIMessageOK	

See Also

mpiFilterAxisMapGet

mpiFilterControl

Declaration

```
MPIControl mpiFilterControl(MPIFilter filter)
```

Required Header: stdmpi.h

Description

mpiFilterControl returns a handle to the motion controller (Control object) associated with the specified Filter object (***filter***).

Return Values	
handle	to a Control object that a Filter object is associated with
MPIHandleVOID	if the Filter object is invalid

See Also

mpiFilterConfigGet | MEIFilterConfig

mpiFilterMotorMapGet

Declaration

```
long mpiFilterMotorMapGet(MPIFilter    filter,
                          MPIObjectMap *motorMap)
```

Required Header: stdmpi.h

Description

mpiFilterMotorMapGet gets the object map of the Motors associated with the Filter (***filter***), and writes it into the structure pointed to by ***motorMap***.

Return Values	
MPIMessageOK	

See Also

mpiFilterMotorMapSet

mpiFilterMotorMapSet

Declaration

```
long mpiFilterMotorMapSet(MPIFilter    filter,
                          MPIObjectMap motorMap)
```

Required Header: stdmpi.h

Description

mpiFilterMotorMapSet sets the Motors associated with the Filter (***filter***) using data from the object map specified by ***motorMap***.

Return Values	
MPIMessageOK	

See Also

mpiFilterMotorMapGet

mpiFilterNumber

Declaration

```
long mpiFilterNumber(MPIFilter  filter,
                     long       *number)
```

Required Header: stdmpi.h

Description

For a motion controller that ***filter*** is associated with, **mpiFilterNumber** writes the index of ***filter*** to the contents of ***number***.

Return Values	
MPIMessageOK	

See Also

mpiFilterIntergratorReset

Declaration

```
long mpiFilterIntegratorReset(MPIFilter filter)
```

Required Header: stdmpi.h

Description

mpiFilterIntergratorReset resets the integrators of filter.

Return Values	
MPIMessageOK	
MPIFilterMessageINVALID_ALGORITHM	

Sample Code

```
/* Enable the amplifier for every motor attached to a motion supervisor */
void motionAmpEnable(MPIMotion  motion)
{
        MPIControl              control;
        MPIAxis                 axis;
        MPIMotor                motor;
        MPIFilter               filter;
        MPIObjectMap    map;
        MPIObjectMap    motionMotorMap;
        long                    motorIndex;
        long                    filterIndex;
        long                    returnValue;
        double                  position;
        long                    enableState;


        /* Get the controller handle */
        control = mpiMotionControl(motion);

        for (axis =  mpiMotionAxisFirst(motion);
                 axis != MPIHandleVOID;
                 axis =  mpiMotionAxisNext(motion, axis)) {

                /* Get the object map for the motors */
                returnValue = mpiAxisMotorMapGet(axis, &map);
                msgCHECK(returnValue);

                /* Add map to motionMotorMap */
                motionMotorMap |= map;
        }

        /* For every motor ... */
        for (motorIndex = 0; motorIndex < MEIXmpMAX_Motors; motorIndex++) {

                if (mpiObjectMapBitGET(motionMotorMap, motorIndex)) {
```

```
                        /* Create motor handle */
                        motor = mpiMotorCreate(control, motorIndex);
                        msgCHECK(mpiMotorValidate(motor));

                        /* Get the state of the amplifier */
                        returnValue = mpiMotorAmpEnableGet(motor, &enableState);
                        msgCHECK(returnValue);

                        /* If the amplifier is disabled ... */
                        if (enableState == FALSE) {

                                /* For every axis */
                                for (axis =  mpiMotionAxisFirst(motion);
                                         axis != MPIHandleVOID;
                                         axis =  mpiMotionAxisNext(motion, axis)) {

                                        /* Get the object map for the motors */
                                        returnValue = mpiAxisMotorMapGet(axis, &map);
                                        msgCHECK(returnValue);

                                        /* If axis is attached to motor ... */
                                        if (mpiObjectMapBitGET(map, motorIndex)) {

                                                /* Get the actual position of the axis */
                                                returnValue = mpiAxisActualPositionGet
(axis, &position);
                                                msgCHECK(returnValue);


                                                /* Set command position equal to actual
position */
                                                returnValue = mpiAxisCommandPositionSet
(axis, position);
                                                msgCHECK(returnValue);
                                        }
                                }

                                /* Get the object map for the filters */
                                returnValue = mpiMotorFilterMapGet(motor, &map);
                                msgCHECK(returnValue);

                                /* For every filter ... */
                                for (filterIndex = 0;
                                         filterIndex < MEIXmpMAX_Filters;
                                         filterIndex++) {

                                        if (mpiObjectMapBitGET(map, filterIndex)) {

                                                /* Create filter handle */
                                                filter = mpiFilterCreate(control,
filterIndex);
                                                msgCHECK(mpiFilterValidate(filter));


                                                /* Reset integrator */
                                                returnValue = mpiFilterIntegratorReset
(filter);
                                                msgCHECK(returnValue);

                                                /* Delete filter handle */
                                                returnValue = mpiFilterDelete(filter);
                                                msgCHECK(returnValue);
                                        }
```

```
                                }

                                /* Enable the amplifier */
                                returnValue = mpiMotorAmpEnableSet(motor, TRUE);
                                msgCHECK(returnValue);
                        }

                        /* Delete motor handle */
                        returnValue = mpiMotorDelete(motor);
                        msgCHECK(returnValue);
                }
        }
}
```

Troubleshooting

If an axis is not in an error state and the filter associated with that axis' motor has a non-zero integration term, then it is very likely that the integrator has built up a substantial integral term. Enabling the motor's amplifier when this has happened could cause the motor to jump with enormous force. Use **mpiFilterIntegratorReset** to reset the integrator before enabling the motor's amplifier to prevent this kind of jump.

Another condition that can cause the motor to jump upon enabling its amplifier is that the command position of the axis is not equal to the actual position of the axis. To prevent this situation, one should use **mpiAxisActualPositionGet** and **mpiAxisCommandPositionSet**. Please refer to this functions for a more in depth discussion.

See Also

MPIFilter | MEIFilterConfig | MEIFilterGainPID | MEIFilterGainPIV
mpiAxisActualPositionGet | mpiAxisCommandPositionSet

meiFilterPosfilterGet

Declaration

```
long meiFilterPostfilterGet(MPIFilter            filter,
                            long                 *sectionCount,
                            MEIPostfilterSection *sections);
```

Required Header: stdmei.h

Description

meiFilterPostfilterGet reads an MPIFilter object's postfilter configuration. It writes to ***sectionCount*** the number of sections within a postfilter if ***sectionCount*** is not NULL. It also writes to ***sections*** the current array of ***filter***'s postfilter sections if sections is not NULL.

The MPI calculates the post filter coefficients and takes into consideration the sample rate of the controller at that time. If you change the sample rate of the controller, you will need to recalculate the post filters. This can be done for all filters specified in Hertz by setting the filters again with the MPI. The MPI will calculate the filters using the current servo sample rate.

Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

filter	the handle of the MPIFilter object whose postfilter configuration is to be read.
***sectionCount**	the data location where the postfilter's current section count will be written.
***sections**	the data location where the postfilter's current section configuration data will be written.

Return Values	
MPIMessageOK	
MPIFilterMessageCONVERSION_DIV_BY_0	
MPIFilterMessageINVALID_FILTER_FORM	

Sample Code

```
/*   Count the number of resonator sections in a MPIFilter object's
postfilter.
     Sample usage:

     returnValue =
         filterResonatorCount(filter, &resonatorCount);
*/

long filterResonatorCount(MPIFilter filter, long* count)
{
     MPIFilterConfig config;
     MEIPostfilterSection sections[MEIMaxBiQuadSections];
     long sectionCount, index;
```

```
     long returnValue = (count==NULL) ? MPIMessageARG_INVALID :
MPIMessageOK;

     if (returnValue == MPIMessageOK)
     {
         returnValue =
             meiFilterPostfilterGet(filter, &sectionCount, sections);

         if (returnValue == MPIMessageOK)
         {
             for (*count=0, index=0; index sectionCount; ++index)
             {
                  if (section[index].type == MEIFilterTypeRESONATOR) ++(*count);
             }
         }
      }
      return returnValue;
}
```

See Also

MEIPostfilterSection | meiFilterPostfilterGet | meiFilterPostfilterSet | meFilterPostfilterSectionGet | MEIMaxBiQuadSections | Post Filter Theory

meiFilterPosfilterSet

Declaration

```
long meiFilterPostfilterSet(MPIFilter             filter,
                            long                  *sectionsCount,
                            MEIPostfilterSection  *sections);
```

Required Header: stdmei.h

Description

meiFilterPostfilterSet sets the number of postfilter sections within an MPIFilter object and configures each postfilter section as well. If ***numberOfSections*** equals zero, then ***sections*** can be NULL and the postfilter will be disabled.

The MPI calculates the post filter coefficients and takes into consideration the sample rate of the controller at that time. If you change the sample rate of the controller, you will need to recalculate the post filters. This can be done for all filters specified in Hertz by setting the filters again with the MPI. The MPI will calculate the filters using the current servo sample rate.

Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

filter	the handle of the MPIFilter object whose postfilter sections will be configured.
***sectionsCount**	the number of postfilter sections to set in the ***filter*** object.
***sections**	a pointer to an array of MEIPostfilterSection data structures to be set in ***filter***.

Return Values	
MPIMessageOK	

Sample Code

```
/*   Set a 4th order low-pass post-filter by using
     two 2nd order low-pass sections.
     Sample usage:

     returnValue =
         fourthOrderLowPass(filter, 300 /* Hz */);
*/
long filterFouthOrderLowpass(MPIFilter filter, long breakPointFrequency)
{
    MPIFilterConfig config;
    MEIPostfilterSection section[MEIMaxBiQuadSections];
    long returnValue;
```

```
    section[0].type = MEIFilterTypeLOW_PASS;
    section[0].form = MEIFilterFormINT_BIQUAD;
    section[0].data.lowPass.breakpoint = breakPointFrequency;
    section[1] = section[0]; /* copy first section */

    returnValue =
        meiFilterPostfilterSet(filter, 2, section);

    return returnValue;
}
```

See Also

MEIPostfilterSection | meiFilterPostfilterGet | meFilterPostfilterSectionSet | MEIMaxBiQuadSections | Post Filter Theory

meiFilterPosfilterSectionGet

Declaration

```
long meiFilterPostfilterSectionGet(MPIFilter              filter,
                                   long                   sectionNumber,
                                   MEIPostfilterSection   *section);
```

Required Header: stdmei.h

Description

meiFilterPostfilterSectionGet reads the configuration of a single section of an MPIFilter object's postfilter. It writes to ***section** the configuration of ***filter***'s postfilter ***sectionNumber***th section.

The MPI calculates the post filter coefficients and takes into consideration the sample rate of the controller at that time. If you change the sample rate of the controller, you will need to recalculate the post filters. This can be done for all filters specified in Hertz by setting the filters again with the MPI. The MPI will calculate the filters using the current servo sample rate.

Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

filter	the handle of the MPIFilter object whose postfilter section configuration is to be read.
sectionNumber	the index of the postfilter section whose configuration is to be read.
section	the data location where the postfilter's current section configuration data will be written.

Return Values	
MPIMessageOK	
MPIFilterMessageCONVERSION_DIV_BY_0	
MPIFilterMessageSECTION_NOT_ENABLED	
MPIFilterMessageINVALID_FILTER_FORM	

Sample Code

```
/*   Test a section of a MPIFilter object's postfilter to
     see if it is a notch type.
     Sample usage:

     returnValue =
         isSectionTypeNotch(filter, 0, &isNotch);
*/
long isSectionTypeNotch(MPIFilter filter, long sectionIndex, long* isNotch)
{
    MPIFilterConfig config;
    MEIPostfilterSection section;
    long returnValue = (isNotch==NULL) ? MPIMessageARG_INVALID :
MPIMessageOK;

    if (returnValue == MPIMessageOK)
    {
        returnValue =
            meiFilterPostfilterSectionGet(filter, sectionIndex, §ion);
        if (returnValue == MPIMessageOK)
        {
            *isNotch = (section.type == MEIFilterTypeNOTCH) ? TRUE : FALSE;
        }
     }

     return returnValue;
}
```

See Also

MEIPostfilterSection | meiFilterPostfilterGet | meFilterPostfilterSectionSet | MEIMaxBiQuadSections | Post Filter Theory

meiFilterPosfilterSectionSet

Declaration

```
long meiFilterPostfilterSectionSet(MPIFilter              filter,
                                   long                   sectionNumber,
                                   MEIPostfilterSection   *section);
```

Required Header: stdmei.h

Description

meiFilterPostfilterSectionSet sets the configuration of a single section of an MPIFilter object's postfilter. It sets ***filter***'s postfilter ***sectionNumber***th section to the configuration specified in ****section***. If the postfilter type is IIR, then this method is essentially equivalent to meiFilterPostfilterSet().

The MPI calculates the post filter coefficients taking into consideration the sample rate of the controller at that time. If you change the change the sample rate of the controller, you will need to recalculate your post filters. This can be done for all filters specified in Hertz by setting the filters again using the MPI. The MPI will calculate the filters using the current servo sample rate.

Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

filter	the handle of the MPIFilter object whose postfilter section configuration is to be set.
sectionNumber	the index of the postfilter section whose configuration is to be set.
***section**	the data location of the section configuration to copy to the controller.

Return Values	
MPIMessageOK	

Sample Code

```
/*   Set a section of a MPIFilter object's postfilter
     to a unity gain filter type.
     Sample usage:

     returnValue =
         setSectionTypeUnityGain(filter, 3);
*/
long setSectionTypeUnityGain(MPIFilter filter, long sectionIndex)
{
    MPIFilterConfig config;
    MEIPostfilterSection section;
    long returnValue;

    section.type = MEIFilterTypeUNITY_GAIN;
    section.form = MEIFilterFormBIQUAD;

    returnValue =
        meiFilterPostfilterSectionSet(filter, sectionIndex,
§ion);

    return returnValue;
}
```

See Also

MEIPostfilterSection | meiFilterPostfilterSet | meFilterPostfilterSectionGet | MEIMaxBiQuadSections | Post Filter Theory

MPIFilterCoeff

Definition

```
typedef union {
    float   f;
    long    l;
} MPIFilterCoeff;
```

Description

MPIEventStatus holds information about a particular event that was generated by the XMP.

f	float coefficient
l	long coefficient

See Also

MPIFilterCoeffCOUNT_MAX | MEIFilterGainPIDCoeff | MEIFilterGainPIVCoeff

MPIFilterConfig / MEIFilterConfig

Definition: MPIFilterConfig

```
typedef struct MPIFilterConfig {
    long           gainIndex;
    MPIFilterGain  gain[MPIFilterGainCOUNT_MAX];

    MPIObjectMap   axisMap;
    MPIObjectMap   motorMap;
} MPIFilterConfig;
```

Description

gainIndex	Gain table index. Gain tables number 0 to MPIFilterGainCOUNT_MAX -1 (MPIFilterGainCOUNT_MAX = 5).
gain	See MPIObjectMap
axisMap	See MPIObjectMap
motorMap	See MPIObjectMap

Definition: MEIFilterConfig

```
typedef struct MEIFilterConfig {
    char                  userLabel[MEIObjectLabelCharMAX+1];
                               /* +1 for NULL terminator */
    MEIXmpAlgorithm       Algorithm;

    MEIXmpAxisInput       Axis[MEIXmpFilterAxisInputs];

    long                  *VelPtr;

    MEIXmpSwitchType      GainSwitchType;
    float                 GainDelay;
    long                  GainWindow;
    MEIXmpSwitchType      PPISwitchType;
    MEIXmpPPIMode         PPIMode;
    float                 PPIDelay;
    long                  PPIWindow;
    MEIXmpIntResetConfig  ResetIntegratorConfig;
    float                 ResetIntegratorDelay;

    MEIXmpFilterForm      PostFilterForm;
    MEIXmpPostFilter      PostFilter;
} MEIFilterConfig;
```

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

Description

MEIFilterConfig contains configuration information specific to a controller. With the exception of the Algorithm element, MEIFilterConfig contains configuration information that are more intuitively accessed by other means (Postfilter parameter) or information for advanced setups and custom controller firmware.

userLabel	This value consists of 16 characters and is used to label the filter object for user identification purposes. The userLabel field is NOT used by the controller.
Algorithm	This value defines the algorithm that the filter is executing every servo cycle. The most common values are: MEIXmpAlgorithmPID PID algorithm MEIXmpAlgorithmPIV PIV algorithm MEIXmpAlgorithmNONE No control algorithm
Axis [MEIXmpFilterAxisInputs]	This array defines the axis (pointer to the axis) and coefficient for the position input into the filter. The input to the filter is the position error of the axis, which is multiplied by the coefficient defined by the Axis array.
***VelPtr**	Pointer to an velocity value for algorithms that require a velocity input (such as the PIV algorithm).
AuxInput [MEIXmpFilterAuxInputs]	This array is a place holder for additional filter inputs from analog sources. This is currently not supported and is reserved for future use.
GainSwitchType	Value to define the gain table switch type. Not implemented in standard firmware.
GainDelay	Custom Delay Not implemented in standard firmware.
GainWindow	Custom Delay Not implemented in standard firmware.
PPISwitchType	Value to define the gain switch type for PPI mode. Not implemented in standard firmware.
PPIMode	Value to define the PPI switch mode. Not implemented in standard firmware.
PPIDelay	Custom Delay Not implemented in standard firmware.
PPIWindow	Custom Window Not implemented in standard firmware.
ResetIntegratorConfig	Value to define the integrator's reset configuration. Not supported in standard firmware.
ResetIntegratorDelay	Value to define the integrator's reset delay. Not supported in standard firmware.

PostFilterForm	This value defines the form for postfilters when they are configured using mpiFilterConfigGet/Set(). Supported values are: • **MEIXmpFilterFormIIR**, IIR Filter • **MEIXmpFilterFormBIQ**, Bi-Quad Filter • **MEIXmpFilterFormSS_BIQ**, State Space form of Bi-Quad Filter • **MEIXmpFilterFormINT_BIQ**, Integer (64-bit) Bi-Quad Filter • **MEIXmpFilterFormINT_SS_BIQ**, Integer State Space form of Bi-Quad Filter Though the postfilter may be configured through this parameter, it is strongly recommended that users use the meiFilterPostfilter.() methods instead for a more intuitive and user-friendly interface.
PostFilter	This array defines the configuration for the filter's postfilter (the type, the length and values for the post filter coefficients). Though the postfilter may be configured though this parameter, it is strongly recommended that users use the meiFilterPostfilter.() methods instead for a more intuitive interface. Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

Sample Code

```
/*   Test whether an MPIFilter object's control loop algorithm is PID.
     Sample usage:

     returnValue =
         isAlgorithmPid(filter, &isPid);
*/

long isAlgorithmPid(MPIFilter filter, long* isPid)
{
     MEIFilterConfig xmpConfig;
     long returnValue = (isPid==NULL) ? MPIMessageARG_INVALID : MPIMessageOK;

     if (returnValue == MPIMessageOK)
     {
         returnValue =
             mpiFilterConfigGet(filter, NULL, &xmpConfig);
         if (returnValue == MPIMessageOK)
         {
             *isPid = (xmpConfig.Algorithm == MEIXmpAlgorithmPID) ? TRUE :
```

```
FALSE;
            }
        }

        return returnValue;
}
```

See Also

mpiFilterConfigGet | mpiFilterConfigSet | meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet

MEIFilterForm

Definition

```
typedef enum{
    MEIFilterFormIIR,
    MEIFilterFormBIQUAD,
    MEIFilterFormSS_BIQUAD,
    MEIFilterFormINT_BIQUAD,
    MEIFilterFormINT_SS_BIQUAD,
} MEIFilterForm;
```

Description

MEIFilterForm describes the form that a digital filter takes on the controller. Please note that the equations listed below use the coefficients loaded onto the controller, not necessarily the coefficients used by the MPI. A user may specify a low pass filter with only a single parameter (the breakpoint) and request that the form of the filter be a space-state biquad form on the controller.

Digital filtering on the XMP is accomplished through 32-bit words. This equates to the use of single precision floating point numbers - a 24-bit mantissa or about 7 decimal places of accuracy. This lack of precision can cause errors in the filtering process normally appearing as DC gain shifts or limit cycling, this especially true when the filter requires more than one section, a 6th order low pass filter would be one example. Filter forms using integer math can provide more internal precision for coefficients and internal registers but at the cost of less dynamic range. Filter forms using integer math take more processing time for the controller and can potentially limit the maximum sample rate of the controller.

The state-space (SS) filter forms allow the scaling of the input and the output, whereas the non-state-space forms only allow output scaling. This helps to prevent the loss of precision of the internal registers while still maintaining a very large dynamic range. Filter forms using state-space forms take more processing time for the controller and can potentially limit the maximum sample rate of the controller. However, a non-integer state-space filter form takes less processing power than an integer non-state-space filter form.

MEIFilterFormIIR	***Deprecated***. Cascaded biquad sections offer better precision and better calculation performance.
MEIFilterFormBIQUAD	Second Order digital filter form, for implementing low/high pass, notch, lead/lag and custom filters. The filter is a single precision floating point canonical form. The biquad filter is defined by the following discrete transfer function: The XMP's representation of this filter is: w0: Intermediate result u(k): filter input a1, a2, b0, b1, and b2: discrete biquad coefficients y(k):filter output x1k and x2k: filter states
MEIFilterFormSS_BIQUAD	Second order digital filter form, for implementing low/high pass, notch, lead/lag and custom filters. The filter is a single precision, floating point state space implementation. This filter applies input and output scaling to the canonical form. The XMP's state space representation of this filter is: u(k): filter input d1, c1, c2, a2, a1,b1: discrete biquad coefficients y(k):filter output p1k and p2k: filter states
MEIFilterFormINT_BIQUAD	Second Order digital filter form, for implementing low/high pass, notch, lead/lag and custom filters. The filter is a fixed point canonical form state space implementation. This form is a fixed point implementation of the floating point form MEIFilterFormBIQUAD. See the definition of MEIFilterFormBIQUAD above for the defining equations for this filter. The input coefficients for this filter (b0, b1, b2, a1 and a2) should all be greater than -2, and less than 2. The coefficients are represented as 32 bit 2's complement, with 1=2^30. The coefficient's numerical format is 1.29 (1 bit whole, 29 bits fractional), and the controller uses an 80 bit accumulator. Only the 32 bit result of the multiplication is output from each section.

MEIFilterFormINT_SS_BIQUAD	Second Order digital filter form, for implementing low/high pass, notch, lead/lag and custom filters. The filter is a fixed point canonical form state space implementation. This form is a fixed point implementation of the floating point form MEIFilterFormSS_BIQUAD. See the definition of MEIFilterFormSS_BIQUAD above for the defining equations for this filter. The input coefficients for this filter (d1, c1, c2, a2, a1 and b1) should all be greater than -2, and less than 2. The coefficients are represented as 32 bit 2's complement, with 1=2^30. The coefficient's numerical format is 1.29 (1 bit whole, 29 bits fractional), and the controller uses an 80 bit accumulator. Only the 32 bit result of the multiplication is output from each section.

See Also

MEIPostfilterSection

MPIFilterGain

Definition

```
typedef struct MPIFilterGain {
   MPIFilterCoeff  coeff[MPIFilterCoeffCOUNT_MAX];
} MPIFilterGain;
```

Description

coeff	see MPIFilterCoeff

Sample Code

```
/* Sets reasonable tuning parameters for a Trust TA9000 test stand */
void setPIDs(MPIFilter filter)
{
   MPIFilterGain gain;
   long returnValue;

   returnValue = mpiFilterGainGet(filter, 0, &gain);
   msgCHECK(returnValue);

   gain.coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_POSITION].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY].f = (float)45;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION].f = (float)101000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION].f = (float)450;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_REST].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffDRATE].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMIT].f = (float)32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH].f = (float)32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITLOW].f = (float)-32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_OFFSET].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_POSITIONFFT].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_FILTERFFT].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_VELOCITYFFT].f = (float)0;

   returnValue = mpiFilterGainSet(filter, 0, &gain);
   msgCHECK(returnValue);
}
```

See Also

MPIFilterGainCOUNT_MAX | MEIFilterGainPIDCoeff | MEIFilterGainPIVCoeff

MEIFilterGainIndex

Definition

```
typedef enum {

    /* Gain table index for normal firmware. */
    MEIFilterGainIndexNO_MOTION = MEIXmpGainNOT_MOVING,
    MEIFilterGainIndexACCEL     = MEIXmpGainACCEL,
    MEIFilterGainIndexDECEL     = MEIXmpGainDECEL,
    MEIFilterGainIndexVELOCITY  = MEIXmpGainCONSTANT_VEL,

    /* Gain table index for Custom 1 firmware. */
    MEIFilterGainIndexSTOPPING2 = MEIXmpGainSTOPPED2,
    MEIFilterGainIndexSTOPPING1 = MEIXmpGainSTOPPED1,
    MEIFilterGainIndexSETTLING  = MEIXmpGainSETTLING,
    MEIFilterGainIndexMOVING    = MEIXmpGainMOVING,
    MEIFilterGainIndexSTOPPING3 = MEIXmpGainSTOPPED3,

    /* Gain table index for Custom 5 firmware. */
    MEIFilterGainIndexMIN       = MEIXmpGainMIN,
    MEIFilterGainIndexMAX       = MEIXmpGainMAX,
    MEIFilterGainIndexNONE      = MEIXmpGainNONE,
    MEIFilterGainIndexSLOPE     = MEIXmpGainSLOPE,

    MEIFilterGainIndexLAST      = MEIXmpGainLAST,
    MEIFilterGainIndexALL       = MEIFilterGainIndexLAST,
                                        /* used for gain get/set() */
    MEIFilterGainIndexFIRST     = MEIFilterGainIndexINVALID + 1,

    MEIFilterGainIndexDEFAULT   = MEIFilterGainIndexNO_MOTION,
} MEIFilterGainIndex;
```

Description

MEIFilterGainIndex is an enumeration for the gain index used in gain scheduling.

In standard firmware, only
MEIFilterGainIndexNO_MOTION,
MEIFilterGainIndexACCEL,
MEIFilterGainIndexDECEL, and
MEIFilterGainIndexVELOCITY
are used. The gain index that is currently used can be found with mpiFilterGainIndexGet(...).

Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV). To change the gain scheduling type from NONE (uses only the gains in gain table index 0), use MEIFilterConfig. GainSwitchType is set with mpiFilterConfigSet(...).

When setting filter gain parameters using mpiFilterGainGet(...) and mpiFilterGainSet(...), use the gain index value to write to a gain index of your choosing.

MEIFilterGainIndexNO_MOTION	No commanded motion. Trajectory parameters Velocity, Acceleration, and Jerk equal zero.
MEIFilterGainIndexACCEL	Acceleration portion of the commanded move.
MEIFilterGainIndexDECEL	Deceleration portion of the commanded move.
MEIFilterGainIndexVELOCITY	Constant velocity portion of the commanded move. Gain switching is configured by setting the GainSwtichType, GainDelay, and GainWindow in the MEIFilterConfig{...} structure and calling mpiFilterConfigGet/Set(...). The GainSwitchType has the following options:

See Also

MEIFilterConfig | mpiFilterConfigGet | mpiFilterConfigSet | MEIXmpSwitchType | mpiFilterGainIndexSet | mpiFilterGainIndexGet | mpiFilterGainGet | mpiFilterGainSet

MEIFilterGainPID

Definition

```
typedef struct MEIFilterGainPID {
    struct {
    float   proportional;     /* Kp */
    float   integral;         /* Ki */
    float   derivative;       /* Kd */
    } gain;
    struct {
        float   position;     /* Kpff */
        float   velocity;     /* Kvff */
        float   acceleration; /* Kaff */
        float   friction;     /* Kfff */
    } feedForward;
    struct {
        float   moving;       /* MovingIMax */
        float   rest;         /* RestIMax */
    } integrationMax;
    long    dRate;            /* DRate */
    struct {
        float   limit;        /* OutputLimit */
        float   limitHigh;    /* OutputLimitHigh */
        float   limitLow;     /* OutputLimitLow */
        float   offset;       /* OutputOffset */
    } output;
    struct {
        float   positionFFT;  /* Ka0 */
        float   filterFFT;    /* Ka1 */
        float   velocityFFT;  /* Ka2 */
    } noise;
} MEIFilterGainPID;
```

Description

MEIFilterGainPID is a structure that defines the filter coefficients for the PID filter algorithm.

See Also

High/Low Output Limits section for special instructions regarding MEIFilterGainPID.
MEIFilterGainPIDCoeff

MEIFilterGainPIDCoeff

Definition

```
typedef         enum {
    MEIFilterGainPIDCoeffGAIN_PROPORTIONAL, /* Kp */
    MEIFilterGainPIDCoeffGAIN_INTEGRAL,     /* Ki */
    MEIFilterGainPIDCoeffGAIN_DERIVATIVE,   /* Kd */

    MEIFilterGainPIDCoeffFEEDFORWARD_POSITION,     /* Kpff */
    MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY,     /* Kvff */
    MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION, /* Kaff */
    MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION,     /* Kfff */

    MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING, /* MovingIMax */
    MEIFilterGainPIDCoeffINTEGRATIONMAX_REST,   /* RestIMax */

    MEIFilterGainPIDCoeffDRATE,             /* DRate */

    MEIFilterGainPIDCoeffOUTPUT_LIMIT,      /* OutputLimit */
    MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH,  /* OutputLimitHigh */
    MEIFilterGainPIDCoeffOUTPUT_LIMITLOW,   /* OutputLimitLow */
    MEIFilterGainPIDCoeffOUTPUT_OFFSET,     /* OutputOffset */

    MEIFilterGainPIDCoeffNOISE_POSITIONFFT, /* Ka0 */
    MEIFilterGainPIDCoeffNOISE_FILTERFFT,   /* Ka1 */
    MEIFilterGainPIDCoeffNOISE_VELOCITYFFT, /* Ka2 */
} MEIFilterGainPIDCoeff;
```

Description

MEIFilterGainPIDCoeff is a structure of enums that defines the filter coefficients for the PID filter algorithm.

Sample Code

```
/* Sets reasonable tuning parameters for a Trust TA9000 test stand */
void setPIDs(MPIFilter filter)
{
   MPIFilterGain gain;
   long returnValue;

   returnValue = mpiFilterGainGet(filter, 0, &gain);
   msgCHECK(returnValue);

   gain.coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_POSITION].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY].f = (float)45;
```

```
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION].f = (float)101000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION].f = (float)450;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_REST].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffDRATE].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMIT].f = (float)32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH].f = (float)32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITLOW].f = (float)-32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_OFFSET].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_POSITIONFFT].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_FILTERFFT].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_VELOCITYFFT].f = (float)0;

   returnValue = mpiFilterGainSet(filter, 0, &gain);
   msgCHECK(returnValue);
}
```

See Also

MEIFilterGainPID

MEIFilterGainPIV

Definition

```
typedef         struct MEIFilterGainPIV {
    struct {
        float   proportional;   /* Kpp */
        float   integral;       /* Kip */
    } gainPosition;
    struct {
        float   proportional;   /* Kpv */
    } gainVelocity1;
    struct {
        float   position;       /* Kpff */
        float   velocity;       /* Kvff */
        float   acceleration;   /* Kaff */
        float   friction;       /* Kfff */
    } feedForward;
    struct {
        float   moving;     /* MovingIMax */
        float   rest;       /* RestIMax */
    } integrationMax;
    struct {
        float   feedback;   /* Kdv */
    } gainVelocity2;
    struct {
        float   limit;      /* OutputLimit */
        float   limitHigh;  /* OutputLimitHigh */
        float   limitLow;   /* OutputLimitLow */
        float   offset;     /* OutputOffset */
    } output;
    struct {
        float   integral;        /* Kiv */
        float   integrationMax;  /* VintMax */
    } gainVelocity3;
    struct {
        float   positionFFT;  /* Ka0 */
        float   smoothing;    /* Ka1 */
        float   filterFFT;    /* Ka2 */
    } noise;
} MEIFilterGainPIV;
```

Change History: Modified in the 03.02.00

Description

MEIFilterGainPIV is a structure that defines the filter coefficients for the PIV filter algorithm.

See Also

High/Low Output Limits section for special instructions regarding MEIFilterGainPIV.
MEIFilterGainPIVCoeff

MEIFilterGainPIVCoeff

Definition

```
typedef         enum {
    MEIFilterGainPIVCoeffGAINPOSITION_PROPORTIONAL,     /* Kpp */
    MEIFilterGainPIVCoeffGAINPOSITION_INTEGRAL,         /* Kip */


    MEIFilterGainPIVCoeffGAINVELOCITY_PROPORTIONAL,     /* Kpv */


    MEIFilterGainPIVCoeffFEEDFORWARD_POSITION,          /* Kpff */
    MEIFilterGainPIVCoeffFEEDFORWARD_VELOCITY,          /* Kvff */
    MEIFilterGainPIVCoeffFEEDFORWARD_ACCELERATION,      /* Kaff */
    MEIFilterGainPIVCoeffFEEDFORWARD_FRICTION,          /* Kfff */


    MEIFilterGainPIVCoeffINTEGRATIONMAX_MOVING,         /* MovingIMax */
    MEIFilterGainPIVCoeffINTEGRATIONMAX_REST,           /* RestIMax */


    MEIFilterGainPIVCoeffGAINVELOCITY_FEEDBACK,         /* Kdv */


    MEIFilterGainPIVCoeffOUTPUT_LIMIT,          /* OutputLimit */
    MEIFilterGainPIVCoeffOUTPUT_LIMITHIGH,      /* OutputLimitHigh */
    MEIFilterGainPIVCoeffOUTPUT_LIMITLOW,       /* OutputLimitLow */
    MEIFilterGainPIVCoeffOUTPUT_OFFSET,         /* OutputOffset */


    MEIFilterGainPIVCoeffGAINVELOCITY_INTEGRAL,         /* Kiv */
    MEIFilterGainPIVCoeffGAINVELOCITY_INTEGRATIONMAX,   /* Vintmax */


    MEIFilterGainPIVCoeffNOISE_POSITIONFFT,      /* Ka0 */
    MEIFilterGainPIVCoeffSMOOTHINGFILTER_GAIN,   /* Ka1 */
    MEIFilterGainPIVCoeffNOISE_FILTERFFT,        /* Ka2 */
} MEIFilterGainPIVCoeff;
```

Change History: Modified in the 03.02.00

Description

MEIFilterGainPIVCoeff is a structure of enums that defines the filter coefficients for the PIV filter algorithm.

Sample Code

```
/* Sets reasonable tuning parameters for a Trust TA9000 test stand */
void setPIDs(MPIFilter filter)
{
   MPIFilterGain gain;
   long returnValue;

   returnValue = mpiFilterGainGet(filter, 0, &gain);
   msgCHECK(returnValue);

   gain.coeff[MEIFilterGainPIDCoeffGAIN_PROPORTIONAL].f = (float)100;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_INTEGRAL].f = (float)0.2;
   gain.coeff[MEIFilterGainPIDCoeffGAIN_DERIVATIVE].f = (float)1000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_POSITION].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY].f = (float)45;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION].f = (float)101000;
   gain.coeff[MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION].f = (float)450;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffINTEGRATIONMAX_REST].f = (float)15000;
   gain.coeff[MEIFilterGainPIDCoeffDRATE].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMIT].f = (float)32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH].f = (float)32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_LIMITLOW].f = (float)-32767;
   gain.coeff[MEIFilterGainPIDCoeffOUTPUT_OFFSET].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_POSITIONFFT].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_FILTERFFT].f = (float)0;
   gain.coeff[MEIFilterGainPIDCoeffNOISE_VELOCITYFFT].f = (float)0;

   returnValue = mpiFilterGainSet(filter, 0, &gain);
   msgCHECK(returnValue);
}
```

See Also

High/Low Output Limits section for special instructions regarding MEIFilterGainPIV.
MEIFilterGainPIV

MEIFilterGainTypePID

Definition

```
static MEIDataType MEIFilterGainTypePID[MPIFilterCoeffCOUNT_MAX] =
{
    MEIDataTypeFLOAT, /* Kp               */
    MEIDataTypeFLOAT, /* Ki               */
    MEIDataTypeFLOAT, /* Kd               */

    MEIDataTypeFLOAT, /* Kpff             */
    MEIDataTypeFLOAT, /* Kvff             */
    MEIDataTypeFLOAT, /* Kaff             */
    MEIDataTypeFLOAT, /* Kfff             */

    MEIDataTypeFLOAT, /* MovingIMax       */
    MEIDataTypeFLOAT, /* RestIMax         */

    MEIDataTypeLONG,  /* DRate            */

    MEIDataTypeFLOAT, /* OutputLimit      */
    MEIDataTypeFLOAT, /* OutputLimitHigh  */
    MEIDataTypeFLOAT, /* OutputLimitLow   */
    MEIDataTypeFLOAT, /* OutputOffset     */
    MEIDataTypeFLOAT, /* Ka0              */
    MEIDataTypeFLOAT, /* Ka1              */
    MEIDataTypeFLOAT, /* Ka2              */
};
```

Description

MEIFilterGainTypePID is a static array that describes the data type of the coefficients for the PID algorithm. Specifically, an element of MEIFilterGainTypePID describes which member of the union MPIFilterCoeff to access when using the data structure MPIFilterCoeff.

MEIFilterGainTypePID allows for a more simple design of general case utilities and configuration routines. If it is known that only the PID parameters will be used, then the data structure MEIFilterGainPID can be used directly without having to manipulate MPIFilterCoeff, MPIFilterCoeff, and MEIFilterGainTypePID.

Sample Code

```
/*  Read the current value of a filter's PID coefficient.  Sample usage:

    returnValue =
        getPidFilterCoeff(filter, MEIFilterGainPIDCoeffGAIN_PROPORTIONAL, &kp);
*/
long getPidFilterCoeff(MPIFilter filter, long index, double* value)
{

    MPIFilterConfig config;
    long returnValue = (value==NULL) ? MPIMessageARG_INVALID : MPIMessageOK;

    if (returnValue == MPIMessageOK)
    {
        returnValue = mpiFilterConfigGet(filter, &config, NULL);

        if (returnValue == MPIMessageOK)
        {
            switch(MEIFilterGainTypePID[index])
            {
                case MEIDataTypeLONG:
                    *value = config.gain[config.gainIndex].coeff[index].l;
                    break;
                case MEIDataTypeFLOAT:
                    *value = config.gain[config.gainIndex].coeff[index].f;
                    break;
                default:
                    returnValue = MPIMessageARG_INVALID;
             }
         }
    }
return returnValue;
}
```

See Also

MPIFilterCoeff | MEIFilterGainTypePIV | MEIFilterGainPID | MEIDataType | MPIFilterGain

MEIFilterGainTypePIV

Definition

```
static MEIDataType MEIFilterGainTypePIV[MPIFilterCoeffCOUNT_MAX] =
{
    MEIDataTypeFLOAT, /* Kpp              */
    MEIDataTypeFLOAT, /* Kip              */


    MEIDataTypeFLOAT, /* Kpv              */


    MEIDataTypeFLOAT, /* Kpff             */
    MEIDataTypeFLOAT, /* Kvff             */
    MEIDataTypeFLOAT, /* Kaff             */
    MEIDataTypeFLOAT, /* Kfff             */


    MEIDataTypeFLOAT, /* MovingIMax       */
    MEIDataTypeFLOAT, /* RestIMax         */


    MEIDataTypeFLOAT, /* Kdv              */


    MEIDataTypeFLOAT, /* OutputLimit      */
    MEIDataTypeFLOAT, /* OutputLimitHigh  */
    MEIDataTypeFLOAT, /* OutputLimitLow   */
    MEIDataTypeFLOAT, /* OutputOffset     */


    MEIDataTypeFLOAT, /* Kiv              */
    MEIDataTypeFLOAT, /* Vintmax          */
    MEIDataTypeFLOAT, /* Ka0              */
    MEIDataTypeFLOAT, /* Ka1              */
    MEIDataTypeFLOAT, /* Ka2              */
};
```

Description

MEIFilterGainTypePIV is a static array that describes the data type of the coefficients for the PIV algorithm. Specifically, an element of MEIFilterGainTypePIV describes which member of the union MPIFilterCoeff to access when using the data structure MPIFilterCoeff.

MEIFilterGainTypePIV allows for a more simple design of general case utilities and configuration routines. If it is known that only the PIV parameters will be used, then the data structure MEIFilterGainPIV can be used directly without having to manipulate MPIFilterCoeff, MPIFilterCoeff, and MEIFilterGainTypePIV.

Sample Code

```
/*   Read the current value of a filter's PIV coefficient.  Sample usage:

     returnValue =
         getPivFilterCoeff(filter, MEIFilterGainPIVCoeffGAINVELOCITY_PROPORTIONAL,
&kpv);
*/
long getPivFilterCoeff(MPIFilter filter, long index, double* value)
{
     MPIFilterConfig config;
     long returnValue = (value==NULL) ? MPIMessageARG_INVALID : MPIMessageOK;

     if (returnValue == MPIMessageOK)
     {

         returnValue = mpiFilterConfigGet(filter, &config, NULL);

         if (returnValue == MPIMessageOK)
         {
              switch(MEIFilterGainTypePIV[index])
              {
                   case MEIDataTypeLONG:
                        *value = config.gain[config.gainIndex].coeff[index].l;
                        break;
                   case MEIDataTypeFLOAT:
                        *value = config.gain[config.gainIndex].coeff[index].
f;
                        break;
                   default:
                        returnValue = MPIMessageARG_INVALID;
              }
         }
    }

    return returnValue;
}
```

See Also

MPIFilterCoeff | MEIFilterGainTypePID | MEIFilterGainPIV | MEIDataType | MPIFilterGain

MPIFilterMessage

Definition

```
typedef enum {
    MPIFilterMessageFILTER_INVALID,
    MPIFilterMessageINVALID_ALGORITHM,
    MPIFilterMessageINVALID_DRATE,
    MPIFilterMessageCONVERSION_DIV_BY_0,
    MPIFilterMessageSECTION_NOT_ENABLED,
    MPIFilterMessageINVALID_FILTER_FORM,
} MPIFilterMessage;
```

Description

MPIFilterMessage is an enumeration of Filter error messages that can be returned by the MPI library.

MPIFilterMessageFILTER_INVALID

The filter number is out of range. This message code is returned by mpiFilterCreate(...) if the filter number is less than zero or greater than or equal to MEIXmpMAX_Filters.

MPIFilterMessageINVALID_ALGORITHM

The filter algorithm is not valid. This message code is returned by mpiFilterIntegratorReset(...) if the filter algorithm is not a member of the MEIXmpAlgorithm enumeration (does not support integrators). This problem occurs if the filter type is set to user or an unknown type with mpiFilterConfigSet(...).

MPIFilterMessageINVALID_DRATE

The filter derivative rate is not valid. This message code is returned by mpiFilterConfigSet(...) if the filter derivative rate is less than 0 or greater than 7.

NOTE: The derivative rate for all gain tables must be in the range [0,7], not just the derivative rate for the current gain table.

MPIFilterMessageCONVERSION_DIV_BY_0

Returned when meiFilterPostfilterGet(...) or meiFilterPostfilterSectionGet(...) cannot convert digital coefficients to analog coefficients. When this error occurs, the offending section(s) will report its type as MEIFilterTypeUNKNOWN and will not contain any analog data.

MPIFilterMessageSECTION_NOT_ENABLED

Returned when meiFilterPostfilterGet(...) or meiFilterPostfilterSectionGet(...) attempt to read postfilter data when no postfilter sections are enabled.

MPIFilterMessageINVALID_FILTER_FORM

Returned when meiFilterPostfilterGet(...) or meiFilterPostfilterSectionGet(...) cannot interpret the current postfilter's form (when the form is something other than NONE, IIR, or BIQUAD).

See Also

mpiFilterCreate

MEIFilterType

Definition

```
typedef enum {
    MEIFilterTypeUNITY_GAIN,
        /* B0 = 1    B1=B2=A1=A2 = 0
        (effectively acting as no filter) */
    MEIFilterTypeSINGLE_ORDER,
    MEIFilterTypeLOW_PASS,
    MEIFilterTypeHIGH_PASS,
    MEIFilterTypeNOTCH,
    MEIFilterTypeRESONATOR,
    MEIFilterTypeLEAD_LAG,
    MEIFilterTypeZERO_GAIN,
        /* b0=b1=b2=a1=a2 = 0
        (this does act as a filter.... zeroing the output) */
    MEIFilterTypeBIQUAD,
        /* Only valid for setting.
        Reading will not return these types */
    MEIFilterTypeDIGITAL_BIQUAD,
    MEIFilterTypePOLES_ZEROS,
    MEIFilterTypeDIGITAL_POLES_ZEROS,
    MEIFilterTypeUNKNOWN,
        /* algorithm couldn't figure out what
        this filter was from the coeffs! */
} MEIFilterType;
```

Description

NOTE: The MPI will attempt to return analog & digital biquad and pole/zero information from meiFilterPostfilterGet(...) and meiFilterPostfilterSectionGet(...). However, the filter types MEIFilterTypeDIGITAL_BIQUAD, MEIFilterTypePOLES_ZEROS, and MEIFilterTypeDIGITAL_POLES_ZEROS are never returned by get() calls -- they are used only for setting postfilters. MEIFilterTypeBIQUAD will only be returned by meiFilterPostfilterGet(...) and meiFilterPostfilterSectionGet(...) if the analog coefficients can be calculated (there is no division by 0) and the section cannot be identified as one of the other analog filter types.

MEIFilterTypeUNITY_GAIN	A unity gain filter. This effectively performs no filtering.
MEIFilterTypeSINGLE_ORDER	A single order filter
MEIFilterTypeLOW_PASS	A low pass filter
MEIFilterType_HIGH_PASS	A high pass filter.
MEIFilterTypeNOTCH	A notch filter
MEIFilterTypeRESONATOR	A resonator filter.
MEIFilterTypeLEAD_LAG	A lead or lag filter.
MEIFilterTypeZERO_GAIN	Zeros the output of a filter.
MEIFilterTypeBIQUAD	An analog biquad filter. When reading postfilter data, this type means that the postfilter section could not be identified as a standard filter type.
MEIFilterTypeDIGITAL_BIQUAD	A digital biquad filter. This is only used for setting postfilter sections.
MEIFilterTypePOLES_ZERO	Analog poles and zeros filter (maximum of two poles and zeros) with unity zero-frequency amplitude. This is only used for setting postfilter sections.
MEIFilterTypeDIGITAL_POLES_ZEROS	Digital poles and zeros filter (maximum of two poles and zeros) with unity zero-frequency amplitude. This is only used for setting postfilter sections.
MEIFilterTypeUNKNOWN	Returned by meiFilterPostfilterGet(...) and meiFilterPostfilterSectionGet(...) if analog coefficients cannot be found. only digital data will be available.

See Also

MEIPostfilterSection | meiFilterPosterfilterGet | meiFilterPosterfilterSet | meiFilterPosterfilterSectionGet | meiFilterPosterfilterSectionSet

MEIPostfilterSection

Definition

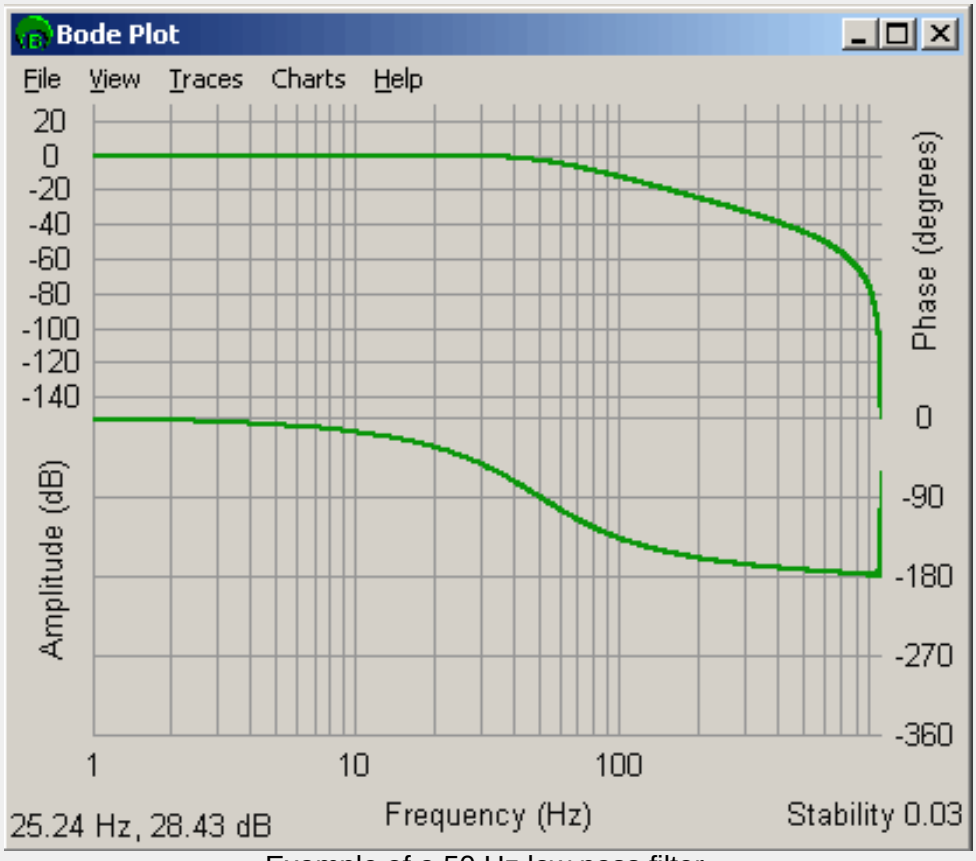
```
typedef struct MEIPostfilterSection {
   MEIFilterType    type;
   MEIFilterForm    form;
   struct {
        struct {
            double breakPoint;      /* Hz */
        } lowPass;

        struct {
            double breakPoint;     /* Hz */
        } highPass;

        struct {
            double centerFrequency; /* Hz */
            double bandwidth;       /* Hz */
        } notch;

        struct {
            double centerFrequency; /* Hz */
            double bandwidth;       /* Hz */
            double gain;            /* dB */
        } resonator;

        struct {
            double lowFrequencyGain;    /* dB */
            double highFrequencyGain;   /* dB */
            double centerFrequency;     /* Hz */
        } leadLag;

        struct {
            double a1;
            double a2;
            double b0;
            double b1;
            double b2;
        } biquad;

        struct {
            double a1;
            double a2;
            double b0;
            double b1;
            double b2;
        } digitalBiquad;
```

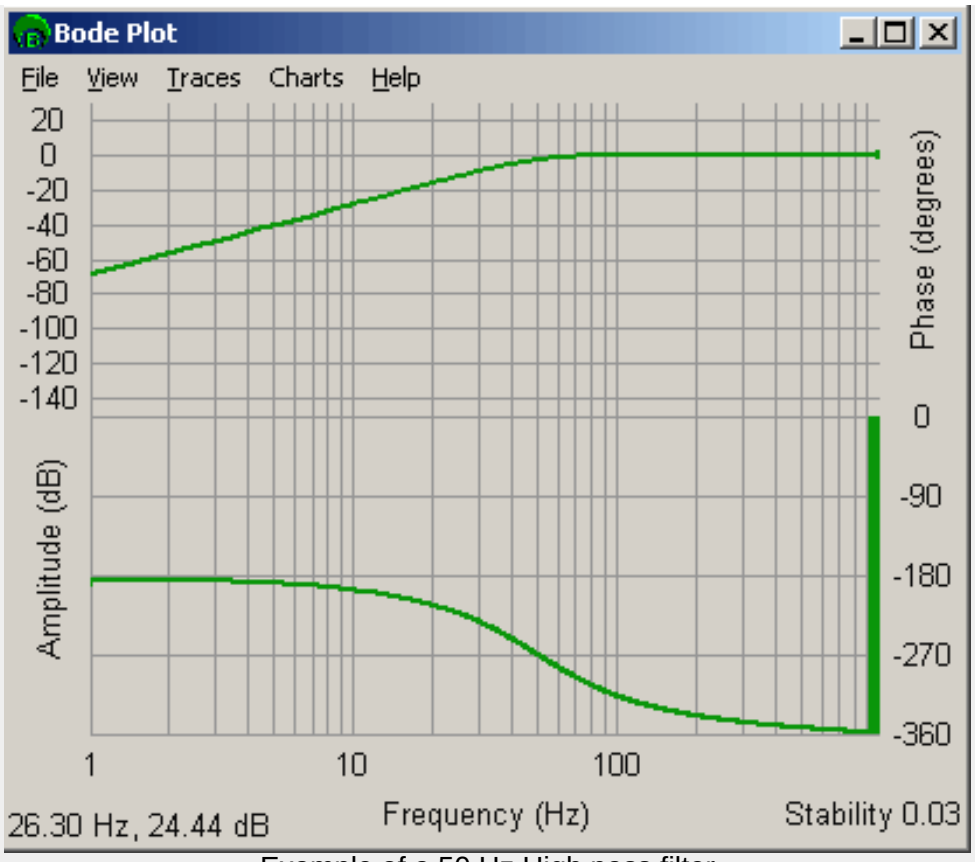
```
        struct {
            long poleCount;
            long zeroCount;
            struct {
                double  real;
                double  imag;
            } pole[2];
            struct {
                double  real;
                double  imag;
            } zero[2];
         } polesZeros;

        struct {
            long poleCount;
            long zeroCount;
            struct {
               double real;
               double imag;
            } pole[2];
            struct {
               double real;
               double imag;
            } zero[2];
         } digitalPolesZeros;

        struct {
            double d1;
            double c1;
            double c2;
            double a2;
            double a1;
            double b1;
        } stateSpaceBiquad;
    } data;
} MEIPostfilterSection;
```

Description

MEIPostfilterSection holds the configuration data for a single section of an MPIFilter object's postfilter. The MPI calculates the post filter coefficients and takes into consideration the sample rate of the controller at that time. If you change the sample rate of the controller, you will need to recalculate the post filters. This can be done for all filters specified in Hertz by setting the filters again with the MPI. The MPI will calculate the filters using the current servo sample rate.

Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

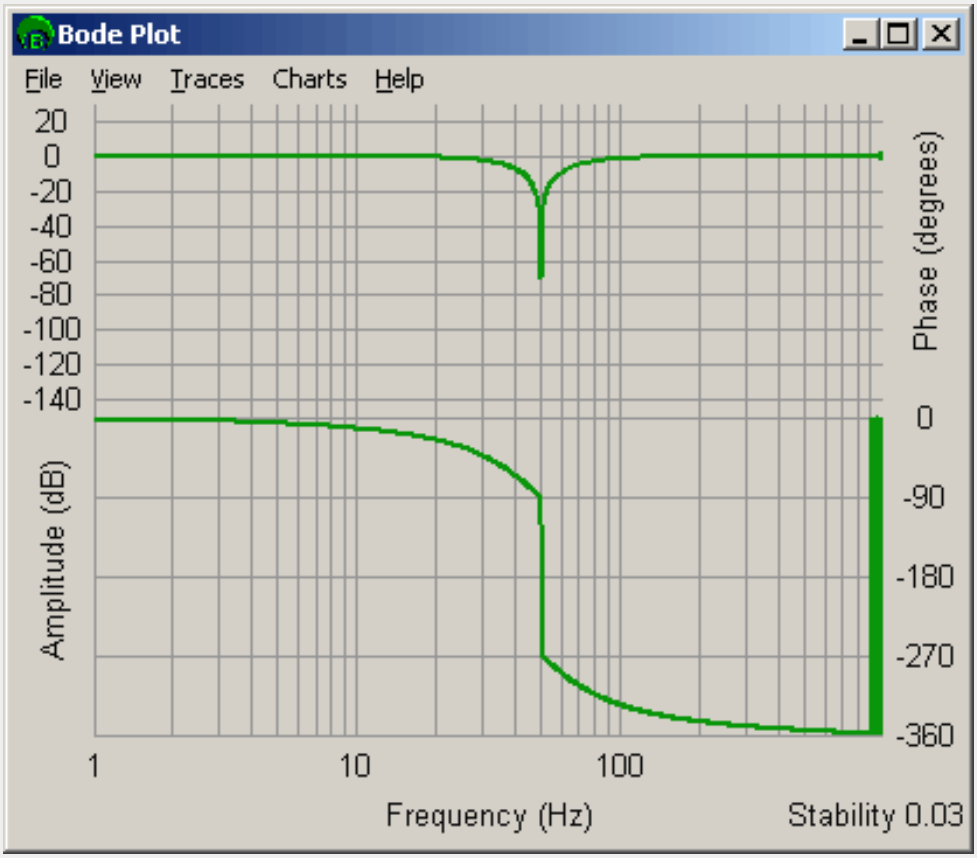
type	The postfilter section type. This field determines which field of the MEIPostfilterSection.data union is used by meiFilterPostfilter.() methods. More information about particular filter types can be found below and in the MEIFilterType documentation.
form	The form of a postfilter section. The form determines how a particular postfilter section is calculated on the controller. All forms have certain limitations and tradeoffs. Please refer to MEIFilterForm for more information.
lowPass.breakpoint	The break point (measured in Hertz) of a low pass postfilter section.  Example of a 50 Hz low pass filter.
highPass. breakpoint	The break point (measured in Hertz) of a high pass postfilter section.



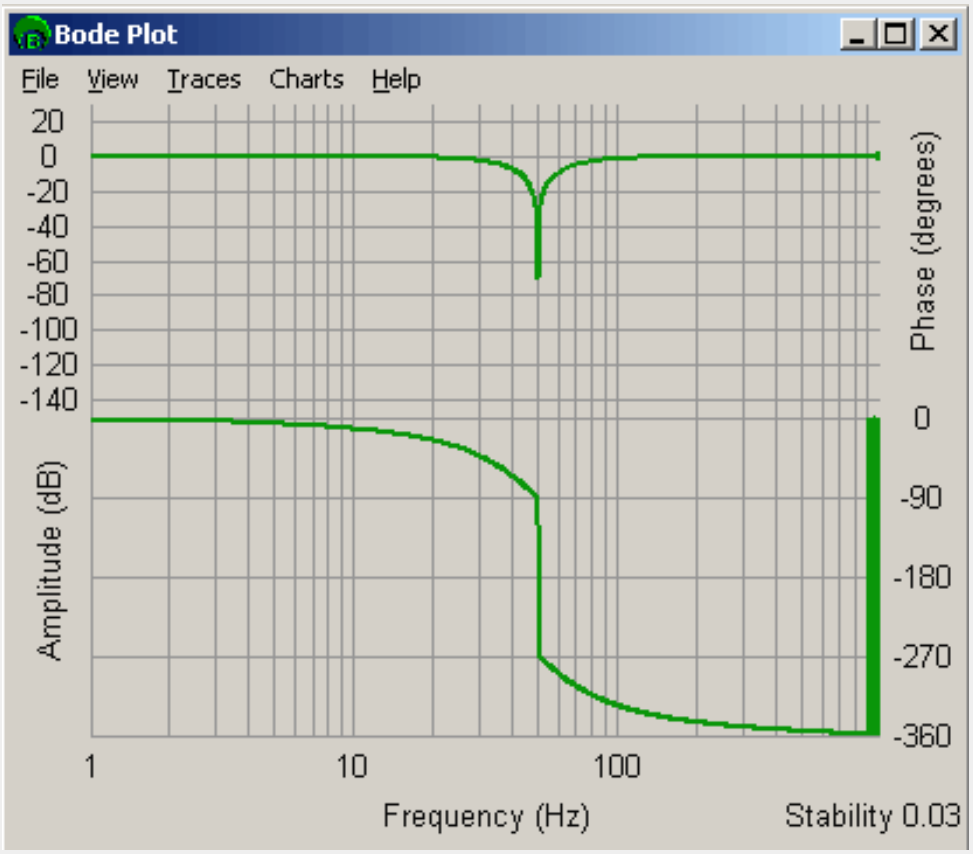
Example of a 50 Hz High pass filter

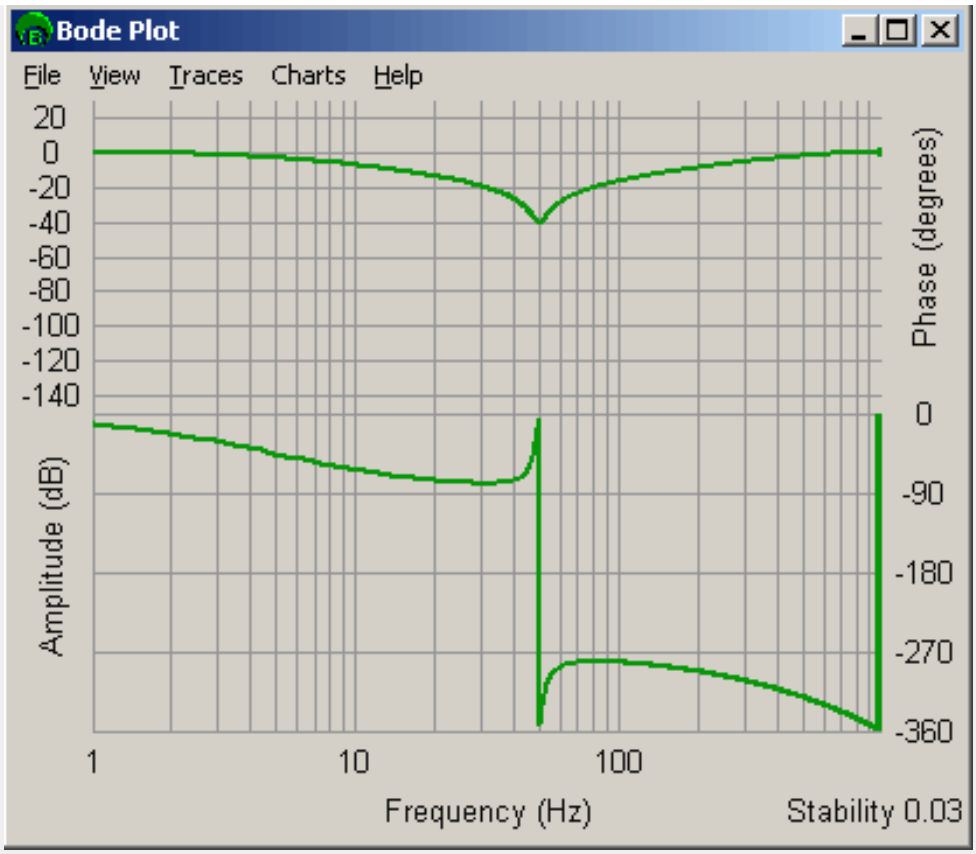
notch. centerFrequency

The center frequency (measured in Hertz) of a notch postfilter section.



Example of a 50 Hz Center / 50 Hz Bandwidth Notch filter. Note that phase wrapping gives the illusion that phase drops 180 degrees after the center frequency. The

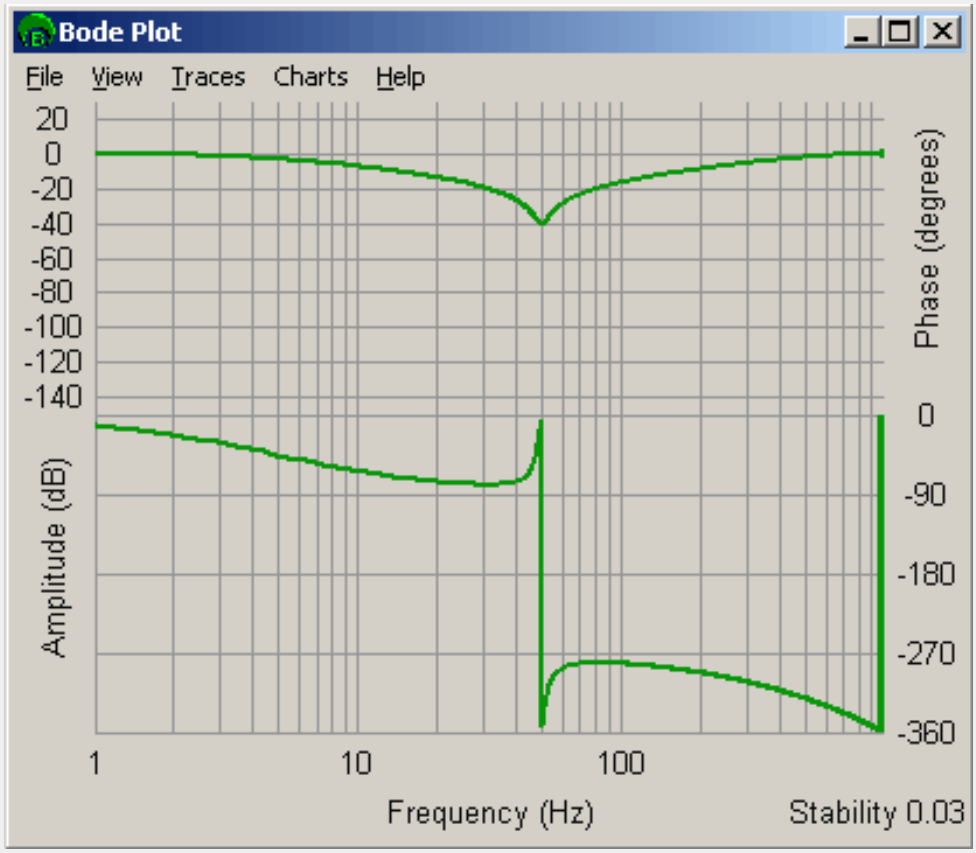
	phase raises by 180 degrees.
notch.bandwidth	The bandwidth (measured in Hertz) of a notch postfilter section.  Example of a 50 Hz Center / 50 Hz Bandwidth Notch filter. Note that phase wrapping gives the illusion that phase drops 180 degrees after the center frequency. The phase raises by 180 degrees.
resonator. centerFrequency	The center frequency (measured in Hertz) of a resonator postfilter section.

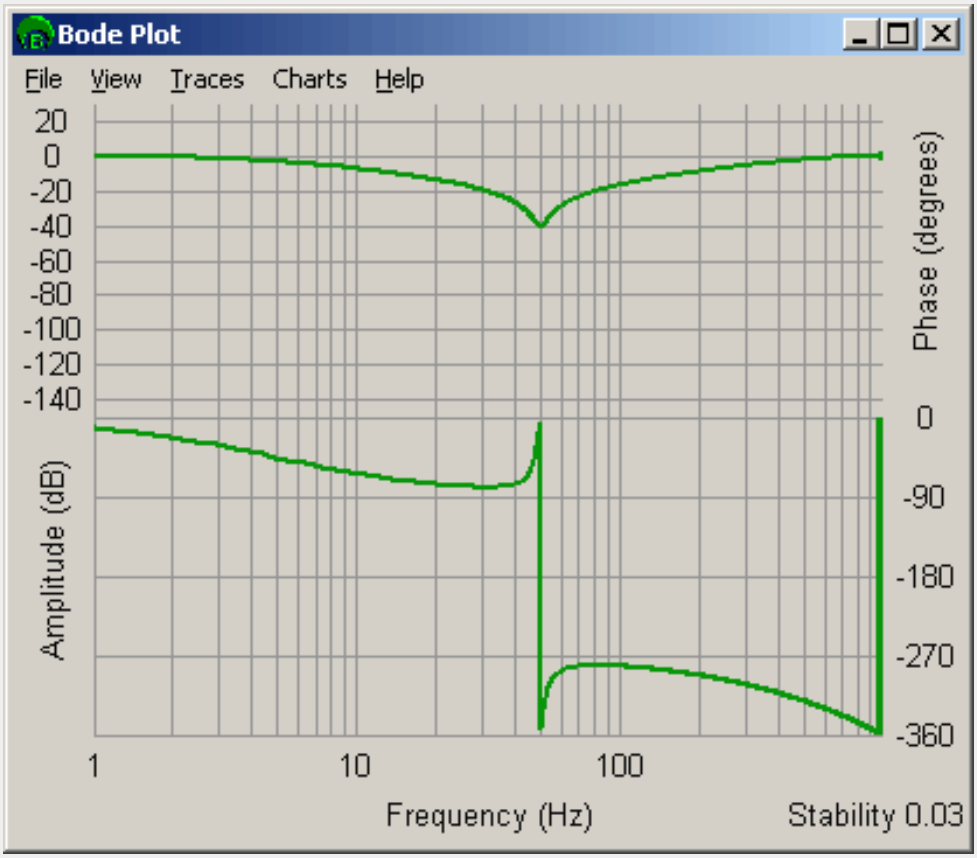


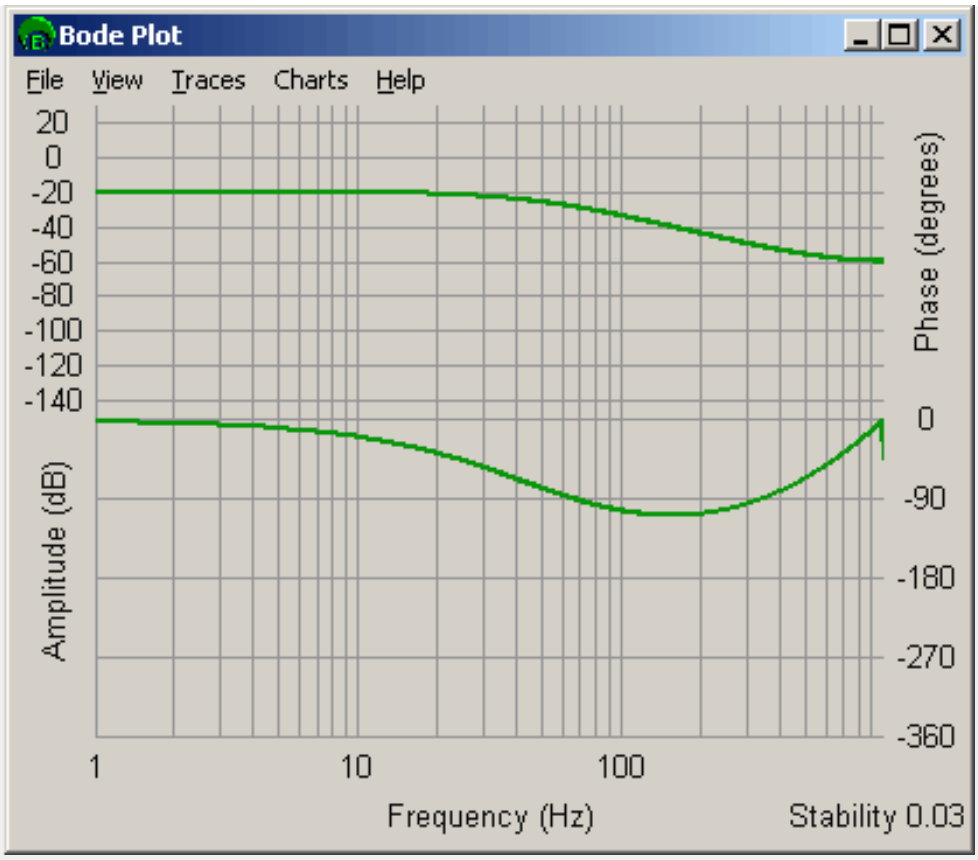
Example of a 50 Hz center / 50 Hz Bandwidth / -40 dB Gain Resonator filter. Note that phase wrapping gives the illusion that the phase drops 360 degrees after the center frequency.

resonator. bandwidth

The bandwidth (measured in Hertz) of a resonator postfilter section.



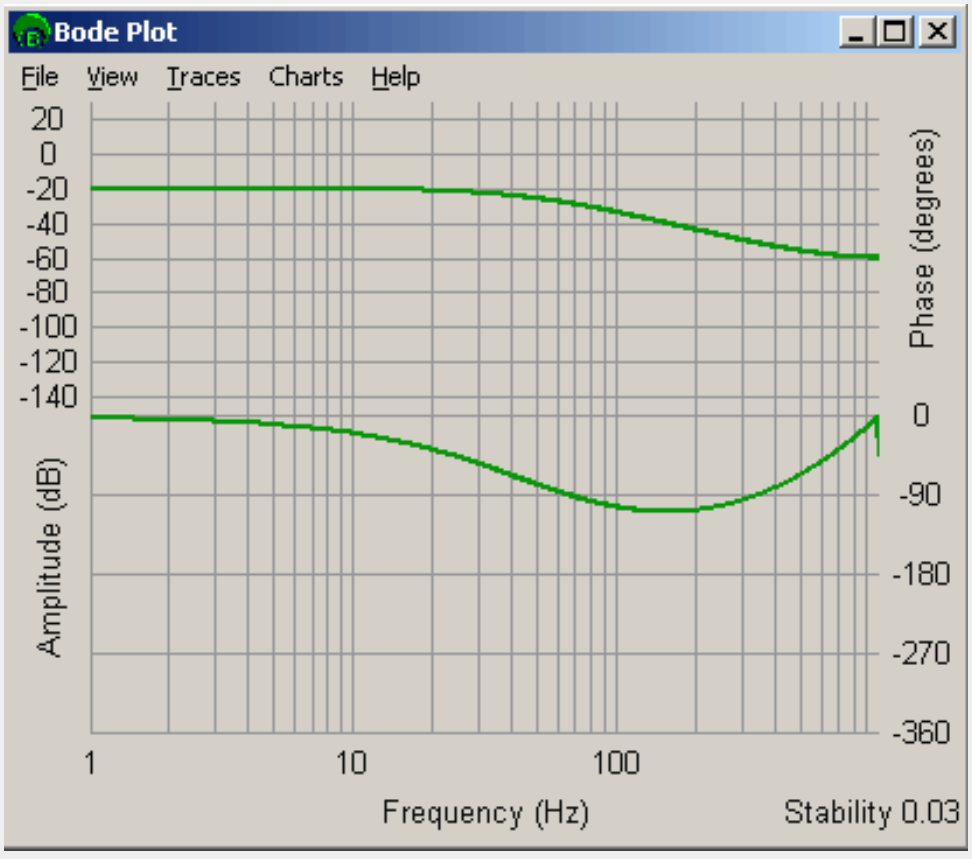
	Example of a 50 Hz center / 50 Hz Bandwidth / -40 dB Gain Resonator filter. Note that phase wrapping gives the illusion that the phase drops 360 degrees after the center frequency.
resonator.gain	The center frequency gain (measured in dB) of a resonator postfilter section.  Example of a 50 Hz center / 50 Hz Bandwidth / -40 dB Gain Resonator filter. Note that phase wrapping gives the illusion that the phase drops 360 degrees after the center frequency.
leadLag. centerFrequency	The center frequency (measured in Hertz) of a lead or lag postfilter section. The amplitude at this frequency is the average amplitude of the low and high frequency amplitudes. The gain (measured in dB) at this point is given by: $$centerFrequencyGain = 20 \cdot \log_{10}\left(\frac{10^{\frac{lowFrequencyGain}{20}} + 10^{\frac{highFrequencyGain}{20}}}{2}\right)$$

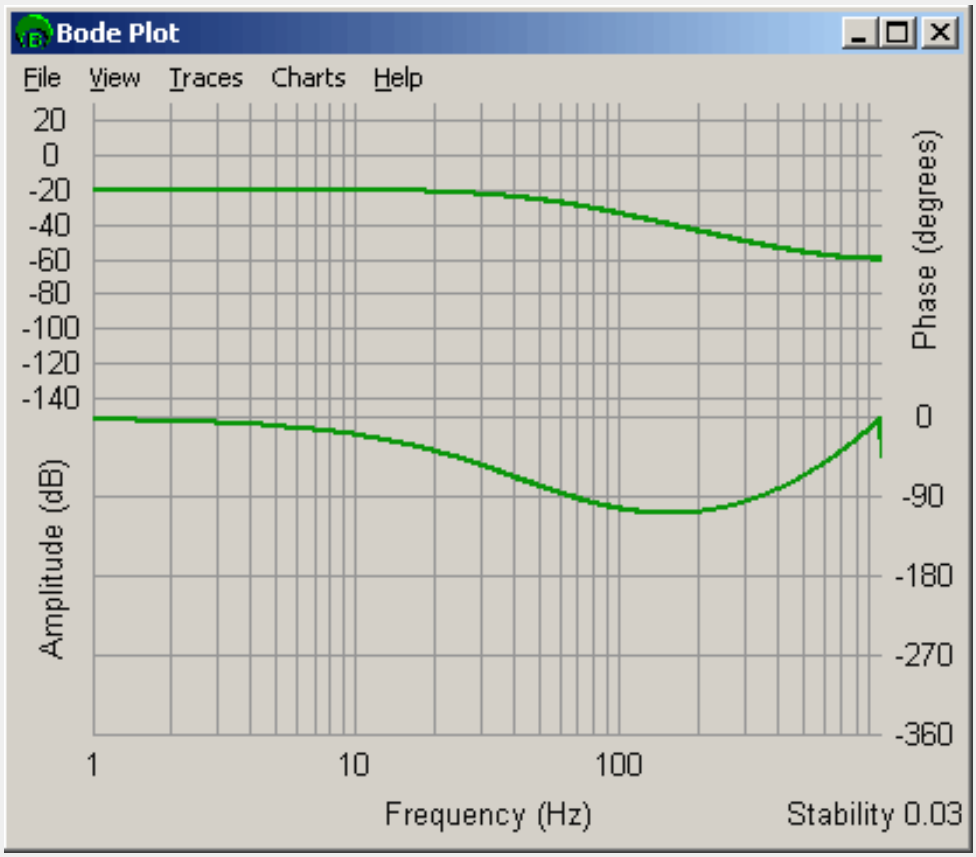


Example of a -20 dB low frequency gain / -60 dB high frequency gain / 50 Hz center lead lag filter.

leadLag.lowFrequencyGain

The low frequency gain (measured in dB) of a lead or lag postfilter section.



	Example of a -20 dB low frequency gain / -60 dB high frequency gain / 50 Hz center lead lag filter.
leadLag. highFrequencyGain	The high frequency gain (measured in dB) of a lead or lag postfilter section.  Example of a -20 dB low frequency gain / -60 dB high frequency gain / 50 Hz center lead lag filter.
biquad.a1 **biquad.a2** **biquad.b0** **biquad.b1** **biquad.b2**	The analog coefficients of a single order or bi-quad postfilter section. Analog values of the postfilter coefficients are produced as parts of a Laplace Transform: $$H(s) = \frac{b_0 + b_1 \cdot s + b_2 \cdot s^2}{1 + a_1 \cdot s + a_2 \cdot s^2} \text{ where } s = j \cdot \omega_{warped}$$ and $$\omega_{warped} = 2 \cdot sampleRate \cdot \tan\left(\frac{\pi \cdot f}{sampleRate}\right)$$
digitalBiquad.a1 **digitalBiquad.a2**	

digitalBiquad.b0	The digital coefficients of a single order or bi-quad postfilter section.
digitalBiquad.b1	
digitalBiquad.b2	
digitalBiquad.d1	The digital coefficients of a state-space bi-quad postfilter section.
digitalBiquad.c1	
digitalBiquad.c2	
digitalBiquad.a2	
digitalBiquad.a1	
digitalBiquad.b1	
polesZeros. poleCount	Analog poles and zeros.
polesZeros. zeroCount	
polesZeros.pole[]. real	
polesZeros.pole[]. imag	
digitalPolesZeros. poleCount	Digital poles and zeros.
digitalPolesZeros. zeroCount	
digitalPolesZeros. pole[].real	
digitalpolesZeros. pole[].imag	
stateSpaceBiquad. d1	State space coefficients.
stateSpaceBiquad. c1	
stateSpaceBiquad. c2	
stateSpaceBiquad. a2	
stateSpaceBiquad. a1	
stateSpaceBiquad. b1	

Sample Code

```
/*   Set a 4th order low-pass post-filter by using two
     2nd order low-pass sections.
     Sample usage:

     returnValue =
         fourthOrderLowPass(filter, 300 /* Hz */);
*/
long filterFouthOrderLowpass(MPIFilter filter, long breakPointFrequency)
{
    MPIFilterConfig config;
    MEIPostfilterSection sections[2];
    long returnValue;

    section[0].type = MEIFilterTypeLOW_PASS;
    section[0].form = MEIFilterFormINT_BIQUAD;
    section[0].lowPass.breakpoint = breakPointFrequency;
    section[1] = section[0]; /* copy first section */

    returnValue =
        meiFilterPostfilterSet(filter, 2, sections);

    return returnValue;
}
```

See Also

MEIFilterType | MEIFilterForm | MEIMaxIIRCoefficients | meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet | Post Filter Theory

MPIFilterCoeffCOUNT_MAX

Definition

```
#define MPIFilterCoeffCOUNT_MAX (20)
```

Description

MPIFilterCoeffCOUNT_MAX is a constant that defines the maximum number of filter coefficients contained in a gain table.

See Also

MPIFilterCoeff

MPIFilterGainCOUNT_MAX

Definition

```
#define MPIFilterGainCOUNT_MAX (5)
```

Description

MPIFilterGainCOUNT_MAX is a constant that defines the maximum number of filter gain tables. The first gain table is used by the standard filter types (all filter types except for the user filter type as defined by the structure MEIXmpAlgorithm). Additional gain tables can be used for manual or automatic gain switching. For firmware that implements automatic gain switching, please contact MEI. Manual gain switching can be accomplished by specifying the gainIndex of the mpiFilterConfig structure using the mpiFilterConfigSet method. Valid gainIndex values range from 0 to MPIFilterGainCOUNT_MAX.

See Also

MPIFilterGain

MEIMaxBiQuadSections

Definition

```
#define MEIMaxBiQuadSections  (6)
```

Description

MEIMaxBiQuadSections is the maximum number of Bi-Quad sections a postfilter can use.

NOTE: The PIV algorithm uses the last Bi-Quad section internally. Thus a user can only use (MEIMaxBiQuadSections - 1) Bi-quad sections with the PIV algorithm.

See Also

MEIPostFilterSection | meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet

Flash Objects

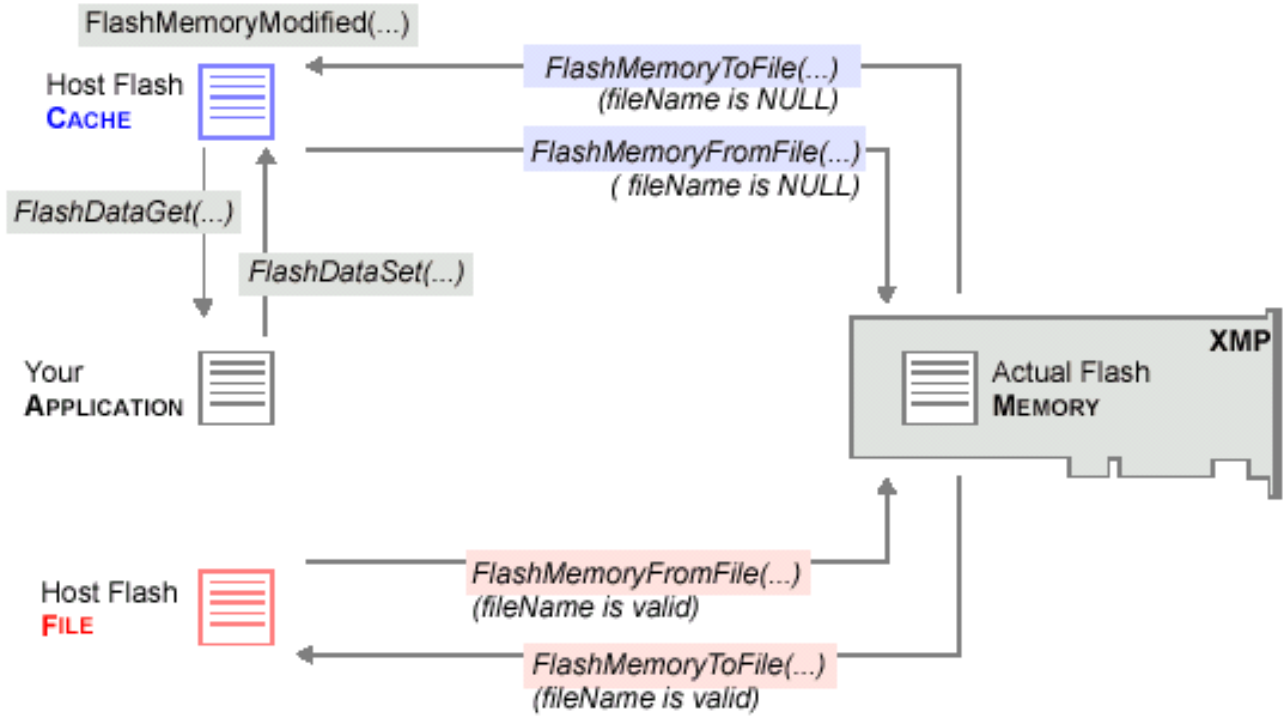
Introduction

A **Flash** object manages nonvolatile flash memory on the XMP/ZMP motion controllers. To optimize flash memory, a host-resident flash memory cache is used to provide faster writing performance.

After your application creates a Flash object (using meiFlashCreate), the Flash object then creates a host-resident flash memory cache. meiFlashCreate will create the flash object and initialize an internal (host-resident) cache with a copy of the flash on the board. All Flash functions (with the exception of meiFlashMemoryFromFile) will modify the host resident cache. When you have finished modifying the host resident cache, use meiFlasheMemoriyFromFile(fileName=NULL) to write the host cache data to the board's flash memory. meiFlashMemoryToFile can be used to copy the board's flash to a user specified file. (If the filename is NULL, the data is copied to the host resident cache.) Host resident cache is deleted when meiFlashDelete is called.

Use the meiFlashDataGet/Set methods to move data between your application and the flash cache. Use the meiFlashMemoryToFile and meiFlashMemoryFromFile methods to move data between the flash cache (or file) and the actual flash memory. Typically, your application would:

1. Create a Flash object [using meiFlashCreate(...)].
2. Pass the **MEIFlash** handle to the FlashConfig[Get/Set](...) methods of the objects to be configured (which in turn call the meiFlashData[Get/Set](...) methods).
3. Write the flash cache to actual flash memory [using meiFlashMemoryFromFile(...)].
4. Delete the Flash object.



| Error Messages |

Methods

Create, Delete, Validate Methods

meiFlash**Create**	Create Flash object
meiFlash**Delete**	Delete Flash object
meiFlash**Validate**	Validate Flash object

Configuration and Information Methods

meiFlash**ConfigGet**	Copy flash config from cache to application memory
meiFlash**ConfigSet**	Copy flash config from application memory to cache
meiFlash**DataGet**	Get count bytes of flash data memory and write them in application memory
meiFlash**DataSet**	Set count bytes of flash data memory using application memory

Memory Methods

meiFlash**MemoryFromFileImage**	
meiFlash**MemoryFromFileType**	Write actual flash memory using the binary image contained in filename
meiFlash**MemoryGet**	Copy count bytes of flash memory to application memory
meiFlash**MemoryModified**	Determine if flash cache has been modified
meiFlash**MemorySet**	Copy count bytes of application memory to flash memory
meiFlash**MemoryFromFile**	Write actual flash memory to cache or to file

meiFlash**MemoryToFile**	Save actual flash memory to cache or to file
meiFlash**MemoryVerify**	

Relational Methods

meiFlash**Control**	Return handle of Control that is associated with Flash

Data Types

- MEIFlash**Config**
- MEIFlash**FileMaxNameChars**
- MEIFlash**FileMaxChars**
- MEIFlash**FileMaxPathChars**
- MEIFlash**Files**
- MEIFlash**FileType**
- MEIFlash**Message**
- MEIFlash**Section**

meiFlashCreate

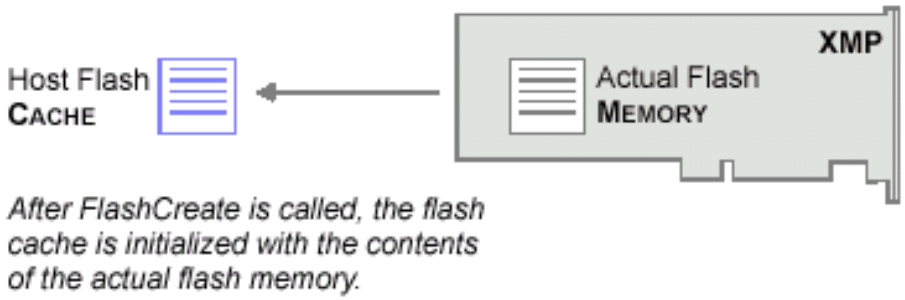
Declaration

```
MEIFlash meiFlashCreate(MPIControl control)
```

Required Header: stdmei.h

Description

meiFlashControl creates a Flash object and a host-resident copy of flash memory on motion controller ***control*** (called the flash cache). *FlashCreate* is the equivalent of a C++ constructor.



After FlashCreate is called, the flash cache is initialized with the contents of the actual flash memory.

Return Values	
handle	to a Flash object
MPIHandleVOID	if the object could not be created

See Also

meiFlashDelete | meiFlashValidate

meiFlashMemoryFromFile

Declaration

```
long meiFlashMemoryFromFile(MEIFlash      flash,
                            MEIFlashFiles *filesIn,
                            MEIFlashFiles *filesOut);
```

Required Header: stdmei.h

Description

meiFlashMemoryFromFile reads the filenames pointed to by ***filesIn*** and copies the binary images into the controller's flash memory. The array of structures pointed to by ***filesOut*** is filled in with the names of the files that were successfully copied into the controller's flash memory. After the next power cycle or mpiControlReset(...) the controller will load the flash images into the local processor and FPGA(s).

flash	a handle to a flash object
***filesIn**	a pointer to an array of flash filename structures to download to the controller. See MEIFlashFiles.
***filesOut**	a pointer to an array of flash filename structures that were successfully downloaded to the controller. See MEIFlashFiles.

Return Values	
MPIMessageOK	

See Also

MEIFlashFiles | mpiControlReset

meiFlashMemoryToFile

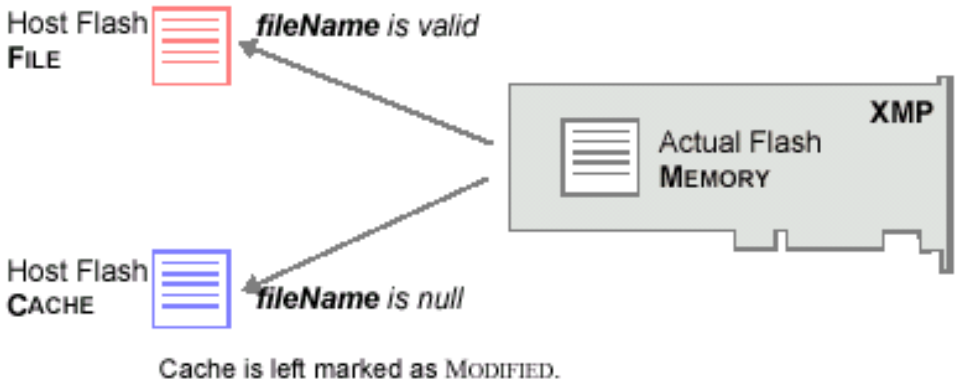
Declaration

```
long meiFlashMemoryToFile(MEIFlash           flash,
                          const char         *fileName,
                          MEIFlashFileType   fileType)
```

Required Header: stdmei.h

Description

meiFlashMemoryToFile saves actual ***flash*** memory to a binary image contained in ***fileName***, or to the flash memory cache.



Return Values	
MPIMessageOK	

See Also

meiFlashDelete

Declaration

```
long meiFlashDelete(MEIFlash flash)
```

Required Header: stdmei.h

Description

meiFlashDelete deletes a Flash object and invalidates its handle (***flash***). *FlashDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

meiFlashCreate | meiFlashValidate

meiFlashDataGet

Declaration

```
long meiFlashDataGet(MEIFlash flash,
                     void     *dst,
                     void     *src,
                     long     count)
```

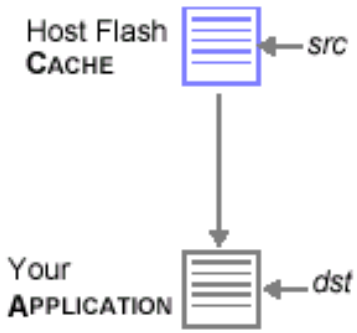
Required Header: stdmei.h

Description

meiFlashDataGet gets ***count*** bytes of ***flash*** data memory starting at address ***src*** and puts (writes) them in application memory starting at address ***dst***. The ***src*** pointer must point into the **MEIXmpData {...}** structure defined in *xmp.h* and be based on the firmware address (MEIXmpData *) returned by mpiControlMemory(...).

Your application cannot access Flash memory directly; instead your application will access the host-resident flash memory cache maintained by ***flash***.

meiFlashDataGet(...) reads from the flash cache and is called only by applications and utilities, while **meiFlashMemoryGet(...)** is a low-level method that reads directly from actual flash memory and is called primarily by other flash methods.



Return Values	
MPIMessageOK	

See Also

mpiControlMemory | meiFlashMemoryGet | meiFlashDataSet

meiFlashDataSet

Declaration

```
long meiFlashDataSet(MEIFlash flash,
                     void     *dst,
                     void     *src,
                     long     count)
```

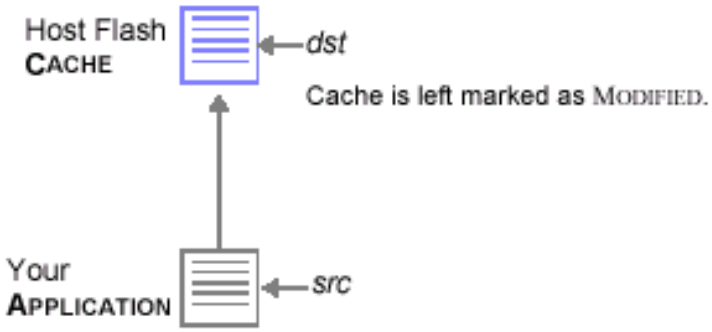
Required Header: stdmei.h

Description

meiFlashDataSet sets (writes) ***count*** bytes of ***flash*** data memory (starting at address ***dst***) using application memory (starting at address ***src***). The ***dst*** pointer must point into the **MEIXmpData{...}** structure defined in *xmp.h* and be based on the firmware address (MEIXmpData *) returned by mpiControlMemory(...).

Your application cannot access Flash memory directly; instead your application will access the host-resident flash memory cache maintained by flash.

mpiControlMemory(...) returns an external pointer that points to the MEIXmpBufferData{...} structure. You cannot use this external pointer with the meiFlashDataSet method to access flash data memory.



Return Values	
MPIMessageOK	

See Also

mpiControlMemory | meiFlashDataGet

meiFlashValidate

Declaration

```
long meiFlashValidate(MEIFlash flash)
```

Required Header: stdmei.h

Description

meiFlashValidate validates the Flash object and its handle (***flash***).

Return Values	
MPIMessageOK	

See Also

meiFlashCreate | meiFlashDelete

meiFlashConfigGet

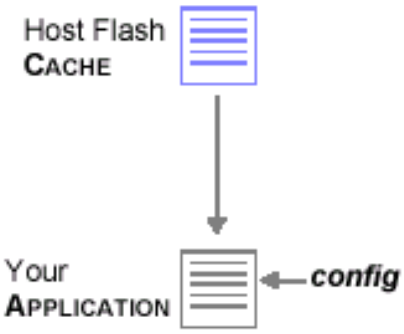
Declaration

```
long meiFlashConfigGet(MEIFlash       flash,
                       MEIFlashConfig *config)
```

Required Header: stdmei.h

Description

meiFlashConfigGet gets the flash configuration and writes it into the structure pointed to by ***config***. The flash configuration includes data about the actual flash memory device (its type and size).



Return Values	
MPIMessageOK	

See Also

meiFlashConfigSet

meiFlashConfigSet

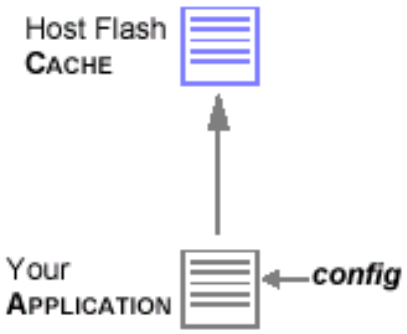
Declaration

```
long meiFlashConfigGet(MEIFlash       flash,
                       MEIFlashConfig *config)
```

Required Header: stdmei.h

Description

meiFlashConfigSet sets (reads) the control configuration from the structure pointed to by ***config***.



Return Values	
MPIMessageOK	

See Also

meiFlashConfigGet

meiFlashMemoryFromFileImage

Declaration

```
long meiFlashMemoryFromFileImage(MEIFlash          flash,
                                 const char        *fileImage,
                                 MEIFlashFileType  *fileType);
```

Required Header: stdmei.h

Description

meiFlashMemoryFromFileImage is used to erase and program controller flash memory. The values to be programmed are stored in the byte array fileImage. The parameter ***fileType*** is used by this method to determine the flash sector and size of the flash memory region to be programmed.

Supported values for ***fileType*** are:

- **MEIFlashFileTypeCode**, Code
- **MEIFlashFileTypeDataInt**, Internal Data
- **MEIFlashFileTypeDataExt**, External Data
- **MEIFlashFileTypeSynqNet**, SynqNet
- **MEIFlashFileTypeCodeAndData**, Code, Internal and External Data.
- **MEIFlashFileTypeFPGA0**, FPGA (Rincon)
- **MEIFlashFileTypeALL**, Code, Internal Data, External Data, and FPGA.

This method exists primarily for compatibility with older software. For new designs, it is recommeded that you use the the meiFlashMemoryFromFileType() method instead.

flash	a handle to a flash object
***fileImage**	a pointer to data in memory to be copied to flash.
***fileType**	where the fileImage will be copied to in flash.

Return Values	
MPIMessageOK	
MEIFlashMessageFLASH_INVALID	
MPIMessageNO_MEMORY	
MPIMessageTIMEOUT	
MEIFlashMessageFLASH_WRITE_ERROR	

See Also

meiFlashMemoryFromFileType

meiFlashMemoryFromFileType

Declaration

```
long meiFlashMemoryFromFileType(MEIFlash          flash,
                                const char        *fileName,
                                MEIFlashFileType  fileType)
```

Required Header: stdmei.h

Description

meiFlashMemoryFromFileType writes actual flash memory using the binary image contained in ***fileName***, or using the flash memory cache.

flash	a handle to a Flash object
***filename**	a string
fileType	an enumeration corresponding to the flash file types

Return Values	
MPIMessageOK	

See Also

meiFlashCreate | meiFlashMemoryFromFile

meiFlashMemoryGet

Declaration

```
long meiFlashMemoryGet(MEIFlash    flash,
                       void        *dst,
                       const void  *src,
                       long        count)
```

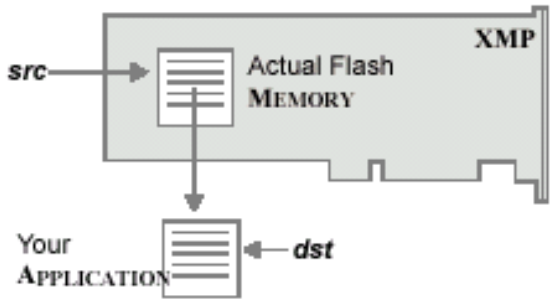
Required Header: stdmei.h
Change History: Modified in the 03.03.00

Description

meiFlashMemoryGet copies ***count*** bytes of actual flash memory (***flash***, starting at address ***src***) to application memory (starting at address ***dst***).

You should calculate the ***src*** pointer by considering flash memory as a stream of bytes, because FlashMemoryGet will adjust the pointer for the actual type of flash memory.

meiFlashMemoryGet(...) is a low-level method that reads directly from actual flash memory and is called primarily by other flash methods, while **meiFlashDataGet(...)** reads from the flash cache and is called only by applications and utilities.



Return Values	
MPIMessageOK	

See Also

meiFlashMemorySet

meiFlashMemoryModified

Declaration

```
long meiFlashMemoryModified(MEIFlash flash,
                            long     *modified)
```

Required Header: stdmpi.h

Description

meiFlashMemoryModified determines if the flash memory cache has been modified. Note that unless meiFlashDataSet(...) has been called previously, the meiFlashMemoryModified(...) method will always return False, regardless of whether the cache has been modified or not modified.

If the "flash cache"	*Then*
has been modified	*FlashMemoryModified* writes True to the location pointed to by modified
has not been modified	*FlashMemoryModified* writes False to the location pointed to by modified

Return Values	
MPIMessageOK	

See Also

meiFlashDataSet

meiFlashMemorySet

Declaration

```
long meiFlashMemorySet(MEIFlash    flash,
                       void        *dst,
                       const void  *src,
                       long        count)
```

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

Description

meiFlashMemorySet copies application memory (starting at address ***src***) to ***count*** bytes of flash memory (***flash***, starting at address ***dst***).

You should calculate the ***dst*** pointer by considering flash memory as a stream of bytes, because FlashMemoryGet will adjust the pointer for the actual type of flash memory.

Remarks

flash memory is of type *unsigned long* *

Return Values	
MPIMessageOK	

See Also

meiFlashMemoryGet

meiFlashMemoryVerify

Declaration

```
long meiFlashMemoryVerify(MEIFlash           flash,
                          const char         *fileName,
                          MEIFlashFileType   fileType);
```

Required Header: stdmei.h

Description

meiFlashMemoryVerify is function that verifies Flash memory against an input file. It takes a flash object, the filename of the input file, and a file type.

The ***fileType*** is used to tell the function what part of flash memory to verify (Code space, data space, both, etc. see MEIFlashFileType enum in flash.h).

NOTE: meiFlashMemoryVerify has been added back to the MPI in order to allow customers who wish to verify that the file loaded in flash is the same as the default file.

flash	a handle to a Flash object
***filename**	a string
fileType	an enumeration corresponding to the flash file types

Return Values	
MPIMessageOK	
MEIFlashMessageFLASH_VERIFY_ERROR	

See Also

meiFlashControl

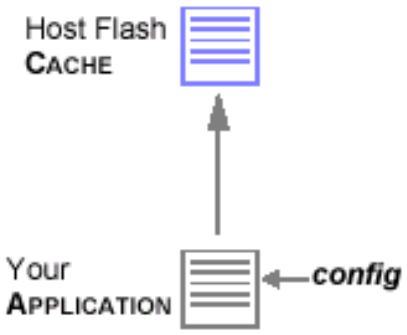
Declaration

```
MPIControl meiFlashControl(MEIFlash flash)
```

Required Header: stdmei.h

Description

meiFlashControl returns a handle to the motion controller (Control object) that a Flash object (***flash***) is associated with.



Return Values	
handle	to a Control object that a Flash object is associated with
MPIHandleVOID	if the Flash object is invalid

See Also

MEIFlashConfig

Definition

```
typedef struct MEIFlashConfig {
    long                wordSize;
    long                sectorSize;
    long                extMemSize;
    MEIFlashSection     all;
    MEIFlashSection     code;
    MEIFlashSection     codeBoot0;
    MEIFlashSection     codeBoot;
    MEIFlashSection     codeMain;
    MEIFlashSection     data;
    MEIFlashSection     dataExt;
    MEIFlashSection     synqNet;
    MEIFlashSection     FPGA0;
} MEIFlashConfig;
```

Description

The **MEIFlashConfig** structure contains the flash configuration parameters. This data is stored in the host's memory and is internally used by the MPI library to manage flash reads/writes. Typically, applications do not need to access the flash configurations.

WARNING:
This data structure is for MEI internal purposes only and should not be configured by a customer.

wordSize	The size of a flash word in bytes.
sectorSize	The size of a flash sector in bytes.
extMemSize	The size of the controller's external memory in bytes.
all	The flash section parameters for the controller's code, data, and local FPGA images.
code	The flash section parameters for the controller's boot and main code.
codeBoot0	The flash section parameters for the controller's boot and main code. (ZMP)
codeBoot	The flash section parameters for the controller's boot code.
codeMain	The flash section parameters for the controller's main code.
data	The flash section parameters for the controller's internal data memory.

dataExt	The flash section parameters for the controller's external data memory.
synqNet	This is for the controller's synqnet data memory.
FPGA0	The flash section parameters for the local FPGA image number 0.

See Also

MEIFlashSection | MEIFlashFileType | meiFlashConfigGet | meiFlashConfigSet

MEIFlashFileMaxNameChars

Definition

```
#define MEIFlashFileMaxNameChars   (12)  /*8.3 format */
```

Description

The **MEIFlashFileMaxNameChars** define is used internally by the MPI when linking a series of strings to create the full path name of the flash file. If the user tries to use a file with more than 12 characters (8.3 format), then the MPI will not be able to create the correct filename and may result in problems when opening the file.

See Also

MEIFlashFileMaxChars | MEIFlashFileMaxPathChars

MEIFlashFileMaxChars

Definition

```
#define MEIFlashFileMaxChars   (120)
```

Description

The MPI has to create filenames with the full path in order to open the flash files.
MEIFlashFileMaxChars is an arbitrary size that is used by the MPI to allocate memory space for the filename. Full path filenames should not exceed 120 characters.

See Also

MEIFlashFileMaxNameChars | MEIFlashFileMaxPathChars

MEIFlashFileMaxPathChars

Definition

```
#define MEIFlashFileMaxPathChars (MEIFlashFileMaxChars -
                                  MEIFlashFileMaxNameChars)
```

Description

MEIFlashFileMaxPathChars is used internally by the MPI when creating full path filenames. This is the number of characters available for the path, not including the actual filename.

See Also

MEIFlashFileMaxNameChars | MEIFlashFileMaxChars

MEIFlashFiles

Definition

```
typedef struct MEIFlashFiles {
    char binFile [MEIFlashFileMaxChars];
    char FPGAFile[MEIXmpFlashMaxFPGAFiles][MEIFlashFileMaxChars];
} MEIFlashFiles;
```

Description

The **MEIFlashFiles** structure specifies the binary files to be downloaded to the controller.

WARNING:
This data structure is for MEI internal purposes only and should not be configured by a customer.

binFile	A controller firmware filename. The firmware file contains configuration data and executable code. The firmware file format is binary.
FPGAFile	An array of FPGA filenames. The FPGA file contains the binary image that is loaded into the FPGA component. Only the appropriate FPGA file can be loaded.

See Also

meiFlashMemoryFromFile

MEIFlashFileType

Definition

```
typedef enum {
    MEIFlashFileTypeNONE = 0,
    MEIFlashFileTypeCode,
    MEIFlashFileTypeDataInt,
    MEIFlashFileTypeDataExt,
    MEIFlashFileTypeSynqNet,
    MEIFlashFileTypeCodeAndData,
    MEIFlashFileTypeFPGA0,
    MEIFlashFileTypeALL    /* Loads Code and all FPGAs
                              (for .bin files that include
                              the FPGA images) */
} MEIFlashFileType;
```

Description

MEIFlashFileType is an enumeration of file types. Each file type contains code or data for the controller's flash memory. This enumeration is used to specify what code/data is to be copied to flash memory with meiFlashMemoryFromFileType(...) or copied from flash memory with meiFlashMemoryToFile(...).

MEIFlashFileTypeCode	Controller processor code only.
MEIFlashFileTypeDataInt	Controller internal data only.
MEIFlashFileTypeDataExt	Controller external data only.
MEIFlashFileTypeSynqNet	Used for flashing the synqNet page of flash.
MEIFlashFileTypeCodeAndData	Controller processor code and data.
MEIFlashFileTypeFPGA0	Local FPGA image number 0.
MEIFlashFileTypeALL	All code, data, and FPGA images.

See Also

meiFlashMemoryFromFileType | meiFlashMemoryToFile | meiFlashMemoryVerify

MEIFlashMessage

Definition

```
typedef enum {

    MEIFlashMessageFLASH_INVALID,
    MEIFlashMessageFLASH_VERIFY_ERROR,
    MEIFlashMessageFLASH_WRITE_ERROR,
    MEIFlashMessagePATH,
    MEIFlashMessageNETWORK_TOPOLOGY_ERROR,
} MEIFlashMessage;
```

Description

MEIFlashMessage lists the error messages returned by the MEIFlash module.

MEIFlashMessageFLASH_INVALID

The flash object is not valid. This message code is returned by meiFlashMemoryFromFile(...), meiFlashMemoryFromFileType(...), meiFlashMemoryToFile(...), or meiFlashMemoryVerify(...) if the flash object's memory cache has not been allocated. The flash memory cache is allocated in mpiFlashCreate(...). To prevent this problem, create flash objects with mpiFlashCreate(...) and check the return value.

MEIFlashMessageFLASH_VERIFY_ERROR

The flash memory verify failed. This message code is returned by meiFlashMemoryVerify(...) if the read from flash memory does not match the read from the file. This indicates that the specified flash file is different from the flash memory. This message code can also be returned from meiFlashMemoryFromFile(...), since it calls meiFlashMemoryVerify(...). To correct this problem, first check that the specified file is the one you intended. If the specified file is correct, use meiFlashMemoryFromFile(...) to download the file.

MEIFlashMessageFLASH_WRITE_ERROR

The flash memory write failed. This message code is returned by meiFlashMemoryFromFile(...) or meiFlashMemoryFromFileType(...) if the flash memory write fails. This indicates a problem in the flash memory component or subsystem.

MEIFlashMessagePATH

The flash file path is too long. This message code is returned by meiFlashMemoryFromFile(...) if the file path is longer than MEIFlashFileMaxPathChars. To correct this problem, use a shorter path.

MEIFlashMessageNETWORK_TOPOLOGY_ERROR

The network topology has not been saved to flash. This message code is returned by object flash config set methods if the network topology has not been previously saved to flash. This message code serves as a warning of a potential safety problem. Saving object configurations to flash will cause those values to be sent to nodes during the next network intialization, even if the flash configurations are not compatible with the network topology. If the network topology is saved to flash, then the controller will automatically verify the topology before sending object configuration values to the nodes.

See Also

MEIFlash

MEIFlashSection

Definition

```
typedef struct MEIFlashSection {
    unsigned char   *address;
    long            size;
    long            sectorIndex;
} MEIFlashSection;
```

Description

The **MEIFlashSection** structure contains the flash configuration parameters for a specified section.

***address**	A pointer to a flash sector address.
size	Length of the flash memory in bytes.
sectorIndex	The flash sector number.

See Also

MEIFlashConfig | meiFlashConfigGet | meiFlashConfigSet

Global Objects

Introduction

Data types that are used by more than one module are defined in the mpidef.h header file.The definitions listed in this section are available to all modules, and are defined in mpidef.h.

Data Types

MPI**Action** / MEI**Action**
MPI**ActionSource**
MEI**DataType**
MPI**IoSource**
MPI**IoType**
MPI**IoTrigger**
MEI**MaxBiQuadSections**
MPI**ModuleId** / MEI**ModuleId**
MEI**NetworkObjectInfo**
MEI**NetworkObjectType**
MEI**NetworkPort**
MEI**NetworkType**
MPI**State**
MPI**Status**
MEI**StatusFlag**
MPI**StatusMask** / MEI**StatusMask**
MPI**Trajectory**
MPI**Wait**

Constants

MPI_INTERFACE_VERSION
MPI_VERSION
MPI_VERSION_MAJOR
MPI_VERSION_MINOR
MPI_VERSION_MAJOR_ID
MPI_VERSION_MINOR_ID
MPI_VERSION_RELEASE

MPI_VERSION_STRINGIZE

Macros

mpi**StatusMaskBIT**

MPIAction / MEIAction

Definition: MPIAction

```
typedef enum {
    MPIActionINVALID,

    MPIActionNONE,
    MPIActionSTOP,
    MPIActionE_STOP,
    MPIActionE_STOP_ABORT,
    MPIActionE_STOP_MODIFY,
    MPIActionE_STOP_CMD_EQ_ACT,
    MPIActionABORT,

    MPIActionDONE,
    MPIActionSTART,
    MPIActionRESUME,
    MPIActionRESET,
    MPIActionCANCEL_REPEAT,
} MPIAction;
```

Change History: Modified in the 03.03.00

Description

MPIAction enumerations are used to perform some sort of action on an MPI object. Currently, only MPIMotion and MPIMotor use the MPIAction enumerations. One can command an MPIMotion object to perform some action with the mpiMotionAction(...) method, while one can get and set the types of actions that will be performed when certain motor events occur with the MPIMotorEventConfig structure with the mpiMotorEventConfigGet(...) and mpiMotorEventConfigSet(...) methods.

MPIActionNONE	Performs no action. Use with MPIMotorEventConfig to prevent a motor event from performing an action.
MPIActionSTOP	Makes a motion supervisor perform a stop. This action can be commanded with mpiMotionAction(...) or by a motor event on the controller. Please see MPIMotionDecelTime for more information about stop actions.

MPIActionE_STOP	Makes a motion supervisor perform an e-stop. This action can be commanded with mpiMotionAction(...) or by a motor event on the controller. Please see MPIMotionDecelTime for more information about e-stop actions.
MPIActionE_STOP_ABORT	Makes a motion supervisor perform an e-stop and then an abort. This action can be commanded with mpiMotionAction(...) or by a motor event on the controller. Please see MPIMotionDecelTime for more information about e-stop actions.
MPIActionE_STOP_MODIFY	Makes a motion supervisor perform an e-stop modify. This action is equivalent to an mpiMotionModify(...) that is pre-loaded into the controller and ends in an error state. This action can be commanded with mpiMotionAction(...) or by a motor event on the controller. Please see MPIAxisEstopModify for more information about e-stop modify actions.
MPIActionE_STOP_CMD_EQ_ACT	The command position is set equal to the previous sample's actual position. After the E_STOP time expires, the AmpEnable is disabled. The settling parameters are supported for this action, when settleOnEstopCmdEqAct in the MPIAxisConfig structure is enabled.
MPIActionABORT	Makes a motion supervisor perform an abort. This action can be commanded with mpiMotionAction(...) or by a motor event on the controller.
MPIActionDONE	is currently not supported and is reserved for future use.
MPIActionSTART	Intended to force a motion supervisor to start when it is waiting for some event (a delay or hold) before starting. This action is currently not supported.
MPIActionRESUME	Makes a motion supervisor to resume motion after a stop action has occurred. A motion supervisor can only resume a motion after a stop event, not an e-stop event. This action can be commanded with mpiMotionAction(...).
MPIActionRESET	Makes a motion supervisor return to an idle state after an error has occurred or after a stop, e-stop, abort, or e-stop/ abort action has occurred. While abort actions and certain errors cause all associated motors to turn off their amp-enable lines, this action does not change the state of any amp-enable lines. One will have to call the method mpiMotorAmpEnableSet(...) to re-enable the amplifiers. This action can be commanded with mpiMotionAction(...).

MPIActionCANCEL_REPEAT	This action makes a repeating cam start or finish at the end of the next cycle. i.e. The cam will continue executing until it passes the start of finish of the cam table. See Repeating Cams .

Remarks

An **MPIAction** can be generated from the host or the firmware. Below is a table where MPIActions originate (start):

MPIAction	Originating from Host	Originating from XMP Firmware
Start	mpiMotionAction(...) (currently unsupported)	NEVER
Resume	mpiMotionAction(...)	NEVER
Reset	mpiMotionAction(...)	NEVER
Stop	mpiMotionAction(...)	Event
E_Stop	mpiMotionAction(...)	Event
ABORT	mpiMotionAction(...)	Event
DONE	NEVER	NEVER

Definition: MEIAction

```
typedef enum {
    MEIActionMAP = MPIActionLAST,

} MEIAction;
```

Description

MEIActionMAP	MotionAction will write the axis mapping relationship of the motion supervisor to the controller. This mapping is written automatically when mpiMotionStart(...) is called.

See Also

mpiMotionAction | MPIMotionDecelTime | MPIMotorEventConfig
mpiMotorEventConfigGet | mpiMotorEventConfigSet | MPIEvent | MPIAxisConfig

How STOP Events Work

MPIActionSource

Definition

```
typedef enum MPIActionSource
{
    MPIActionSourceUSER,
    MPIActionSourceCONTROLLER,
} MPIActionSource;
```

Change History: Added in the 03.04.00

Description

MPIActionSource is an enumeration that helps determine whether a non-idle axis and motion supervisor state was caused by an application on the host computer or by an action created by a controller event.

Idle states are not checked for the action source because the idle state is considered to be the "normal" state. If the state of an axis or motion supervisor is idle, then MPIStatus.actionSource will be set to ***MPIActionSourceCONTROLLER***.

It is sometimes useful to know whether a non-idle state was caused by a controller event or host action. If the action source was the controller, the controller may be queried to find out what event caused the error state. In a multi-threaded or multi-process environment, it can also be useful to know if a another thread or process commanded an action that placed the motion supervisor or axis into a non-idle state.

MPIActionSourceUSER	An application on the host computer commanded an action via mpiMotionAction that put the axis or motion supervisor into the current non-idle state.
MPIActionSourceCONTROLLER	An event on the motion controller put the axis or motion supervisor into the current non-idle state.

Remarks

The MPIMotor object does not store information about action sources. Therefore, mpiMotorStatus will always set MPIStatus.actionSource to ***MPIActionSourceCONTROLLER***.

Sample Code

```
MPIStatus status;
   returnValue = mpiAxisStatus(axis, &status, NULL);
   msgCHECK(returnValue);

   if (status.state != MPIStateIDLE)
   {
       switch(status.actionSource)
       {
           case MPIActionSourceUSER:
                printf("Action source was the host computer\n");
                break;
           case MPIActionSourceCONTROLLER:
                printf("Action source was the controller\n");
                break;
           default:
                printf("ERROR: The action source value is invalid.\n");
                break;
       }
   }
```

See Also

MPIStatus | mpiAxisStatus | mpiMotionStatus | mpiMotionAction

MEIDataType

Definition

```
typedef enum {
    MEIDataTypeINVALID = 0, /* this should stay zero - static
                               arrays init to zero by default */
    MEIDataTypeCHAR,
    MEIDataTypeSHORT,
    MEIDataTypeUSHORT,
    MEIDataTypeLONG,
    MEIDataTypeULONG,
    MEIDataTypeFLOAT,
    MEIDataTypeDOUBLE,
    MEIDataTypeINT64,
    MEIDataTypeUINT64,
} MEIDataType;

static MEIDataType  MEIFilterGainTypePID[MPIFilterCoeffCOUNT_MAX] =
{
    MEIDataTypeFLOAT, /* Kp               */
    MEIDataTypeFLOAT, /* Ki               */
    MEIDataTypeFLOAT, /* Kd               */

    MEIDataTypeFLOAT, /* Kpff             */
    MEIDataTypeFLOAT, /* Kvff             */
    MEIDataTypeFLOAT, /* Kaff             */
    MEIDataTypeFLOAT, /* Kfff             */

    MEIDataTypeFLOAT, /* MovingIMax       */
    MEIDataTypeFLOAT, /* RestIMax         */

    MEIDataTypeLONG,  /* DRate            */

    MEIDataTypeFLOAT, /* OutputLimit      */
    MEIDataTypeFLOAT, /* OutputLimitHigh  */
    MEIDataTypeFLOAT, /* OutputLimitLow   */
    MEIDataTypeFLOAT, /* OutputOffset     */
    MEIDataTypeFLOAT, /* Ka0              */
    MEIDataTypeFLOAT, /* Ka1              */
    MEIDataTypeFLOAT, /* Ka2              */
};
```

Change History: Modified in the 03.04.00; added MEIDataTypeINT64 and MEIDataTypeUINT64. Modified in the 03.02.00.

Description

MEIDataType is an enumeration of data types for the filter coefficients.

MEIDataTypeCHAR	character filter data type
MEIDataTypeSHORT	short integer filter data type
MEIDataTypeUSHORT	unsigned short integer filter data type
MEIDataTypeLONG	long integer filter data type
MEIDataTypeULONG	unsigned long filter data type
MEIDataTypeFLOAT	floating point filter data type
MEIDataTypeDOUBLE	double precision floating point filter data type
MEIDataTypeINT64	
MEIDataTypeUINT64	

See Also

MPIIoSource

Definition

```
typedef union {
    MPIHandle    motor; /* MOTOR */
    long         index;  /* USER */
} MPIIoSource;
```

Description

MPIIoSource is an enumeration of data types for the filter coefficients.

motor	Handle to a motor object that is the source for the I/O.
index	Value of the index for a user input. User I/O's are no longer supported by the xmp (user I/O's are handled through the motor object).

See Also

MPIIoType

Definition

```
typedef enum {
    MPIIoTypeINVALID,
    MPIIoTypeMOTOR_DEDICATED,
    MPIIoTypeMOTOR_GENERAL,
    MPIIoTypeUSER,
} MPIIoType;
```

Change History: Modified in the 03.03.00

Description

MPIIoType is an enumeration of data types for the digital I/O. It is used in Sequences. For digital I/O commands within sequences, the user can specify the Io type. Both Dedicated and General Motor I/O are 32-bit words. The DEDICATED and GENERAL Io types are used to specify which word in the motor structure is to be used in the command.

MPIIoTypeMotor_DEDICATED	Specifies the I/O type as DEDICATED.
MPIIoTypeMotor_GENERAL	Specifies the I/O type as GENERAL.
MPIIoTypeUSER	Value specifies the I/O type as user (User I/O types are currently not supported. User I/O is available through the motor objects).

See Also

MPICommandParams | MPICommandType

MPIIoTrigger

Definition

```
typedef struct MPIIoTrigger {
    MPIIoType         type;
    MPIIoSource       source;
    unsigned long     mask;
    unsigned long     pattern;
} MPIIoTrigger;
```

Description

type	See MPIIoType.
source	See MPIIoSource.
mask	Value that specifies the mask to be applied to the I/O.
pattern	Value that specifies the pattern to be compared to the masked I/O.

See Also

MEIMaxBiQuadSections

Definition

```
#define MEIMaxBiQuadSections    (6)
```

Description

MEIMaxBiQuadSections defines the maximum number of biquad sections in an MPIFilter's postfilter. Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

NOTE: The PIV algorithm uses the last biquad section internally and so the user can only use (MEIMaxBiQuadSections-1) sections if the PIV algorithm is the current control algorithm.

See Also

meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet

MPIModuleId / MEIModuleId

Definition: MPIModuleId

```
typedef enum {
    MPIModuleIdINVALID,


    MPIModuleIdMESSAGE,
    MPIModuleIdAXIS,
    MPIModuleIdCAPTURE,
    MPIModuleIdCOMMAND,
    MPIModuleIdCOMPARE,
    MPIModuleIdCOMPENSATOR,
    MPIModuleIdCONTROL,

    MPIModuleIdEVENT,
    MPIModuleIdEVENTMGR,
    MPIModuleIdFILTER,

    MPIModuleIdMOTION,
    MPIModuleIdMOTOR,

    MPIModuleIdNOTIFY,
    MPIModuleIdPATH,
    MPIModuleIdPROBE,
    MPIModuleIdRECORDER,
    MPIModuleIdSEQUENCE,


    MPIModuleIdEXTERNAL = 0x80,

    MPIModuleIdMAX = 0xFF
} MPIModuleId;
```

Description

MPIModuleId is used to identify what module a particular MPIHandle belongs to. If the handle is an external memory pointer instead of an MPI object handle, MPIModuleIdEXTERNAL will be returned by MPI methods.

Definition: MEIModuleId

```
typedef enum {
    MEIModuleIdPLATFORM,


    MEIModuleIdCAN,

    MEIModuleIdCLIENT,
    MEIModuleIdELEMENT,
    MEIModuleIdFLASH,
    MEIModuleIdLIST,
    MEIModuleIdMAP,
    MEIModuleIdPACKET,
    MEIModuleIdSERVER,

    MEIModuleIdSYNQNET,
    MEIModuleIdSQNODE,
    MEIModuleIdDRIVE_MAP,



    MEIModuleIdBLOCK = MEIModuleIdSQNODE,
}MEIModuleId;
```

Description

MEIModuleId is used to identify what module a particular MPIHandle belongs to. If the handle is an external memory pointer instead of an MPI object handle, MPIModuleIdEXTERNAL will be returned by MPI methods.

See Also

mpiObjectModuleId | mpiObjectValidate

MEINetworkObjectInfo

Definition

```
typedef struct MEINetworkObjectInfo {
    MEINetworkObjectType  type;
    long                  number;
} MEINetworkObjectInfo;
```

Change History: Added in the 03.03.00

Description

MEINetworkObjectInfo contains the information about a network object.

type	is the type of network object.
number	is the object number of the network object type.

See Also

meiSynqNetNetworkObjectNext | meiSqNodeNetworkObjectNext

MEINetworkObjectType

Definition

```
typedef enum MEINetworkObjectType {
    MEINetworkObjectTypeNONE,       /* no object */
    MEINetworkObjectTypeTERMINATOR,
    MEINetworkObjectTypeSQNODE,
    MEINetworkObjectTypeSYNQNET,
} MEINetworkObjectType;
```

Change History: Added in the 03.03.00

Description

MEINetworkObjectType enumerations list all network object types supported by the MPI release. These defines are mostly used for network traversal routines.

MEINetworkObjectTypeNONE	No object was found.
MEINetworkObjectTypeTERMINATOR	A Loop-back Connector was found (end of a string).
MEINetworkObjectTypeSQNODE	A node object was found.
MEINetworkObjectTypeSYNQNET	A SynqNet object was found (start or end of a network).

See Also

MEINetworkObjectInfo | meiSyqnNetNetworkObjectNext | meiSqNodeNetworkObjectNext

MEINetworkPort

Definition

```
typedef enum MEINetworkPort {
    MEINetworkPortIN0,
    MEINetworkPortOUT0,
} MEINetworkPort;
```

Description

MEINetworkPort enumerations are used to specify a network port. Network ports represent the physical network connections into which the cables are plugged. These ports are commonly found in pairs on SynqNet controllers and SynqNet nodes. OUT ports from the controller or an upstream node connect to the IN port of a downstream node. When the OUT port of the last downstream node is connected back to the IN port of the controller, the network is considered to be configured as a RING network type.

MEINetworkPortIN0	"IN" Network port 0
MEINetworkPortOUT0	"OUT" Network port 0

See Also

MEINetworkType

MEINetworkType

Definition

```
typedef enum MEINetworkType {
    MEINetworkTypeINVALID = -1,    /* no nodes found */
    MEINetworkTypeSTRING,
    MEINetworkTypeSTRING_DUAL,
    MEINetworkTypeRING,
} MEINetworkType;
```

Change History: Modified in the 03.03.00

Description

MEINetworkType enumerations list all network topologies supported by this MPI release.

MEINetworkTypeINVALID	No nodes were found on the network
MEINetworkTypeSTRING	The network topology type is a string of nodes
MEINetworkTypeSTRING_DUAL	The network topology type is two strings of nodes.
MEINetworkTypeRING	The network topology type is a ring of nodes.

See Also

MEISynqNetInfo | meiSynqNetInfo

SynqNet Topologies

MPIState

Definition

```
typedef enum {
    MPIStateIDLE,
    MPIStateMOVING,
    MPIStateSTOPPING,
    MPIStateSTOPPED,
    MPIStateSTOPPING_ERROR,
    MPIStateERROR,
} MPIState;
```

Description

MPIState is an MPI enum that is used to describe the current state of the controller's motion state machine. MPIState resides in the MPIStatus structure. Currently MPIState is only used with motion module.

MPIStateIDLE	The state of motion is idle and waiting to resume motion.
MPIStateMOVING	The state of the motion is moving.
MPIStateSTOPPING	The state of the motion is stopping. This occurs from a Stop event, but not an E_Stop, E_Stop Abort, or Abort events. The stop command could have come from the firmware or MPI.
MPIStateSTOPPED	The move has stopped due to a STOP command. It requires a MotionReset to go back to IDLE OR a MotionResume to resume stopped motion. MotionStart can also be called at any time to start a new move.
MPIStateSTOPPING_ERROR	The state of the motion is performing an emergency stop and/ or abort on all axes. The move is stopping due to an error (ESTOP, ABORT, etc.).
MPIStateERROR	The state of the motion is in error. The error state is generated from an E_Stop or Abort event. It requires a MotionReset to get back to IDLE before loading another move.

See Also

MPIStatus

MPIStatus

Definition

```
typedef struct MPIStatus {
    MPIState        state;
    MPIAction       action;
    MPIEventMask    eventMask;

    MPI_BOOL        settled;
    MPI_BOOL        atTarget;

    MPIActionSource actionSource;

    MPIStatusMask   statusMask;
} MPIStatus;
```

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

Description

MPIStatus is an MPI enum that is used to describe the current state of the controller's motion state machine. MPIState resides in the MPIStatus structure. Currently MPIState is only used with motion module.

state	Value that indicates the state of an xmp controller's motion supervisor.
action	Value that indicates the action to perform for a motion supervisor.
eventMask	A bit mask that holds the current events that are active or latched for a particular object. This is often used to help determine the source of an error.
settled	Value that indicates if an axis associated with a motion supervisor has settled (is in fine position).
atTarget	Value that indicates if an axis associated with a motion supervisor has completed its command trajectory (i. e. the command position has reached the targeted end point of the move).
actionSource	Value that indicates whether the state in the **state** parameter was caused by a user action or by an action created by a controller event.
statusMask	A bit mask that hold the current extended status flags for a particular object. This is often used to help determine the source of an error.

Sample Code

```
long printAxisErrorSources(MPIAxis axis)

{
    MPIEventType  eventType;
    MPIStatus     status;
    MPIStatusFlag flag;
    long          axisNumber;
    long          returnValue;

    /* Read the axis number */
    returnValue = mpiAxisNumber(axis, &axisNumber);
    if (returnValue != MPIMessageOK) return returnValue;
```

```
    /* Read the axis status */
    returnValue = mpiAxisStatus(axis, &status, NULL);

    if (returnValue != MPIMessageOK) return returnValue;

    /* Print the events currently active on the axis */

    for (eventType=MPIEventTypeFIRST; eventType<MEIEventTypeLAST; ++eventType)
    {
         if (mpiEventMaskBitGET(status.eventMask, eventType) != FALSE)
         {
            printf("Event \"%s\" is active on axis %d\n", mpiEventTypeName(eventType),
axisNumber);
         }
    }

    /* mpiStatusMaskBIT() */
    for (flag=MPIStatusFlagFIRST ; flag<MEIStatusFlagLAST; ++flag)
    {
         if (status.statusMask & mpiStatusMaskBIT(flag) != FALSE)
        {
            printf("Status flag type %d is active on axis %d\n", flag, axisNumber);
        }
    }
}
```

See Also

MPIState | MPIStatusMask

MEIStatusFlag

Definition

```
typedef enum {
    MEIStatusFlagBROKEN_WIRE,                /* 0 */
    MEIStatusFlagILLEGAL_STATE,              /* 1 */
    MEIStatusFlagABS_ENCODER_FAULT,          /* 2 */
    MEIStatusFlagABS_ENCODER_TIMEOUT,        /* 3 */
    MEIStatusFlagBROKEN_WIRE_SECONDARY,      /* 4 */
    MEIStatusFlagILLEGAL_STATE_SECONDARY,    /* 5 */
} MEIStatusFlag;
```

Change History: Modified in the 03.04.00

Description

MEIStatusFlag is an enumeration of status flags (bits) for an object. The status flags represent the present condition for an object.

Please see MEIStatusMask data type for more information.

See Also

MEIStatusMask | MPIStatusMask

MPIStatusMask / MEIStatusMask

Definition: MPIStatusMask

```
typedef enum {
    MPIStatusMaskNONE   = 0x0,
    MPIStatusMaskMOTOR  = MPIStatusMaskNONE, /* 0x00000001 */
    MPIStatusMaskALL    = mpiStatusMaskBIT(MPIStatusFlagLAST) - 1  /* 0x00000001 */
} MPIStatusMask;
```

Description

MPIStatusMask is an enumeration of bit masks for the MEIStatusFlags. The status masks represent the present condition for an object.

MPIStatusMaskMOTOR	Value specifies the motor's status mask.
MPIStatusMaskALL	Value specifies the status mask that encompasses all the possible status flags.

Definition: MEIStatusMask

```
typedef enum {
    MEIStatusMaskBROKEN_WIRE             = mpiStatusMaskBIT
(MEIStatusFlagBROKEN_WIRE),
                                           /* 0x00000002 */
    MEIStatusMaskILLEGAL_STATE           = mpiStatusMaskBIT
(MEIStatusFlagILLEGAL_STATE),\
                                           /* 0x00000004 */
    MEIStatusMaskABS_ENCODER_FAULT       = mpiStatusMaskBIT
(MEIStatusFlagABS_ENCODER_FAULT),
                                           /* 0x00000008 */
    MEIStatusMaskABS_ENCODER_TIMEOUT     = mpiStatusMaskBIT
(MEIStatusFlagABS_ENCODER_TIMEOUT),
                                           /* 0x00000010 */
    MEIStatusMaskBROKEN_WIRE_SECONDARY   = mpiStatusMaskBIT
(MEIStatusFlagBROKEN_WIRE_SECONDARY),
                                           /* 0x00000020 */
    MEIStatusMaskILLEGAL_STATE_SECONDARY = mpiStatusMaskBIT
(MEIStatusFlagILLEGAL_STATE_SECONDARY),
                                           /* 0x00000040 */

    MEIStatusMaskMOTOR =   /* 0x0000001E */
                             (MEIStatusMaskBROKEN_WIRE  |
                              MEIStatusMaskILLEGAL_STATE |
                              MEIStatusMaskABS_ENCODER_FAULT |
                              MEIStatusMaskABS_ENCODER_TIMEOUT),

    MEIStatusMaskALL   =   /* 0x0000001E */
         (mpiStatusMaskBIT(MEIStatusFlagLAST) - 1) & ~MPIStatusMaskALL
} ;
```

Description

MEIStatusMask is an enumeration of bit masks for the MEIStatusFlags. The status masks represent the present condition for an object.

MEIStatusMaskBROKEN_WIRE	Broken wire on the primary encoder input signals. Occurs when any of the differential encoder input channels (A+ and A-, B+ and B-, or I+ and I-), have the same logic state. This mask indicates either a floating or shorted encoder input signal.
MEIStatusMaskILLEGAL_STATE	Illegal encoder logic state on the primary encoder input signals. Occurs when the A and B encoder input channels transition simultaneously. This mask indicates either faulty encoder signal logic, encoder frequencies that are too high, or noisy encoder signals.
MEIStatusMaskABS_ENCODER_FAULT	Absolute encoder initialization failure. Occurs when the hardware fails to read the absolute position information from an encoder.
MEIStatusMaskABS_ENCODER_TIMEOUT	Absolute encoder response timeout. Occurs when the encoder fails to respond to a request for absolute position data.
MEIStatusMaskBROKEN_WIRE_SECONDARY	Broken wire on the secondary encoder input signals. Occurs when any of the differential encoder input channels (A+ and A-, B+ and B-, or I+ and I-), have the same logic state. This mask indicates either a floating or shorted encoder input signal.
MEIStatusMaskILLEGAL_STATE_SECONDARY	Illegal encoder logic state on the secondary encoder inputs. Occurs when the A and B encoder input channels transition simultaneously. This flag indicates either faulty encoder signal logic, encoder frequencies that are too high, or noisy encoder signals.
MEIStatusMaskMOTOR	Bit mask containing all of the motor specific MEIStatusFlags set.
MEIStatusMaskALL	Bit mask containing all of the MEIStatusFlags set.

See Also

MPIStatus | MEIStatusFlag | MPIStatusMask

MPITrajectory

Definition

```
typedef struct MPITrajectory {
    double    velocity;
    double    acceleration;
    double    deceleration;
    double    jerkPercent;
    double    accelerationJerk;
    double    decelerationJerk;
} MPITrajectory;
```

Description

MPITrajectory contains the motion profile parameters for simple point to point type motion.

NOTE: Not all firmware binaries support S_CURVE_JERK and VELOCITY_JERK motion types due to code space limitations. mpiMotionStart(...) and mpiMotionModify(...) will return MPIMotionMessagePROFILE_NOT_SUPPORTED if the controller does not support the requested move type.

velocity	Rate of change of position. Specifies the constant slew rate for S_CURVE, TRAPEZOIDAL, S_CURVE_JERK, VELOCITY, and VELOCITY_JERK motion types. Units are counts per second.
acceleration	Rate of change of velocity. Specifies the initial ramp to reach constant velocity for TRAPEZOIDAL and VELOCITY motion types. Also, specifies the ramp from the initial jerk to the next jerk before constant velocity for S_CURVE, S_CURVE_JERK and VELOCITY_JERK motion types. Units are counts per second * second.
deceleration	Rate of change of velocity. Specifies the final ramp to reach zero velocity for TRAPEZOIDAL motion types. Also, specifies the ramp from the jerk after constant velocity to the final jerk for S_CURVE and S_CURVE_JERK motion types. Not applicable for VELOCITY and VELOCITY_JERK motion types. Units are counts per second * second.
jerkPercent	Portion of acceleration and deceleration ramp to perform jerk profile for S_CURVE, VELOCITY, S_CURVE_JERK, and VELOCITY_JERK motion types. Units are in percent. Range is 0.0 to 100.0.

accelerationJerk	Rate of change of acceleration. Specifies the initial jerk rates to reach constant velocity for S_CURVE_JERK and VELOCITY_JERK motion types. Units are counts per second * second * second.
decelerationJerk	Rate of change of deceleration. Specifies the final jerk rate to reach zero velocity for S_CURVE_JERK motion types. Not applicable for VELOCITY and VELOCITY_JERK motion types. Units are counts per second * second * second.

Sample Code

```
MPITrajectory trajectory;

    mpiAxisTrajectory(axis, &trajectory);

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

See Also

MPIMotionType | mpiMotionTrajectory | mpiMotionStart | mpiMotionModify

MPIWait

Definition

```
typedef enum {
    MPIWaitFOREVER = -1,
    MPIWaitPOLL = 0,
    MPIWaitMSEC
} MPIWait;
```

Description

MPIWait enumerations define basic wait times for certain MPI methods.

MPIWaitFOREVER	Makes MPI methods wait forever for an event to occur before returning.
MPIWaitPOLL	Makes MPI methods see if a certain event has occurred. If an event has not occurred, then the MPI method will generally return immediately returning the value MPIMessageTIMEOUT.
MPIWaitMSEC	Defines a period of one millisecond. If used alone, this will make MPI methods wait for one millisecond for an event occurs before returning. One can pass an an agument a multiple of MPIWaitMSEC to make MPI methods wait longer periods of time. For example, the following statement will make mpiPlatformKey wait 5 milliseconds for a user keystroke: mpiPlatformKey(5 * MPIWaitMSEC); If an event does not occur within the specified time, MPI methods will generally return the value MPIMessageTIMEOUT.

Warning

The MPI depends on the ability of the operating system it is running on to be able to activate threads or put threads to sleep for a specified period of time in order for these times to be accurate. Microsoft Windows platforms are not real-time operating systems and are known to be unable to activate threads any quicker than 10 milliseconds. If you encounter a timing problem, it is likely an operating system timing issue.

See Also

mpiControlInterruptWait | mpiNotifyEventWait | mpiObjectTimeoutGet
mpiObjectTimeoutSet | meiPlatformKey

MPI_INTERFACE_VERSION

Definition

```
#define  MPI_INTERFACE_VERSION
         MPI_VERSION_MAJOR "." MPI_VERSION_MINOR MPI_VERSION_BRANCH
```

Description

MPI_INTERFACE_VERSION defines a string that represents the MPI library's interface version. The MPI library compares the MPI_VERSION with the MPI_INTERFACE_VERSION during mpiControlInit to check for compatibility between your application binary and the MPI library.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_VERSION | mpiControlInit

MPI_VERSION

Definition

```
#define  MPI_VERSION    MPI_INTERFACE_VERSION "." MPI_VERSION_RELEASE
```

Description

MPI_VERSION defines a string that represents the MPI library version. The MPI library compares the MPI_VERSION with the MPI_INTERFACE_VERSION during mpiControlInit to check for compatibility between your application binary and the MPI library.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_INTERFACE_VERSION | mpiControlInit

MPI_VERSION_MAJOR

Definition

```
#define  MPI_VERSION_MAJOR  MPI_VERSION_STRINGIZE(MPI_VERSION_MAJOR_ID)
```

Change History: Modified in the 03.03.00

Description

MPI_VERSION_MAJOR defines a string that represents the major portion of the MPI_VERSION and MPI_INTERFACE_VERSION. The MPI library compares the MPI_VERSION with the MPI_INTERFACE_VERSION during mpiControlInit(...) to check for compatibility between your application binary and the MPI library.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_VERSION | MPI_INTERFACE_VERSION | mpiControlInit

MPI_VERSION_MINOR

Definition

```
#define  MPI_VERSION_MINOR  MPI_VERSION_STRINGIZE(MPI_VERSION_MINOR_ID)
```

Change History: Modified in the 03.03.00

Description

MPI_VERSION_MINOR defines a string that represents the minor portion of the MPI_VERSION and MPI_INTERFACE_VERSION. The MPI library compares the MPI_VERSION with the MPI_INTERFACE_VERSION during mpiControlInit(...) to check for compatibility between your application binary and the MPI library.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_VERSION | MPI_INTERFACE_VERSION | mpiControlInit

MPI_VERSION_MAJOR_ID

Definition

```
#define  MPI_VERSION_MAJOR_ID  03
```

Change History: Added in the 03.03.00

Description

MPI_VERSION_MAJOR_ID defines the major portion of the MPI_VERSION and MPI_INTERFACE_VERSION. The MPI library compares the MPI_VERSION with the MPI_INTERFACE_VERSION during mpiControlInit to check for compatibility between your application binary and the MPI library.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_VERSION | MPI_INTERFACE_VERSION | mpiControlInit

MPI_VERSION_MINOR_ID

Definition

```
#define  MPI_VERSION_MINOR_ID  04
```

Change History: Modified in the 03.04.00

Description

MPI_VERSION_MINOR_ID defines the minor portion of the MPI_VERSION and MPI_INTERFACE_VERSION. The MPI library compares the MPI_VERSION with the MPI_INTERFACE_VERSION during mpiControlInit to check for compatibility between your application binary and the MPI library.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_VERSION | MPI_INTERFACE_VERSION | mpiControlInit

MPI_VERSION_RELEASE

Definition

```
#define  MPI_VERSION_RELEASE  "ReleaseVersion"
```

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

Description

MPI_VERSION_RELEASE defines the release portion of the MPI_VERSION.

The MPI Version Numbering scheme is formally defined to represent the major, minor, and release level for a particular library version.

See Also

MPI_VERSION

Branch Example of Software Life Cycle

MPI_VERSION_STRINGIZE

Definition

```
#define  MPI_VERSION_STRINGIZE(id) MPI_VERSION_STRINGIZE_INTERNALX(id)
```

Change History: Added in the 03.03.00

Description

MPI_VERSION_STRINGIZE is a macro to convert id from a value to a string. This is used internally by the MPI to convert the major and minor version values to strings.

See Also

MPI_INTERFACE_VERSION | mpiControlInit

mpiStatusMaskBIT

Declaration

```
#define   mpiStatusMaskBIT(flag) (0x1 << (flag))
```

Required Header: stdmpi.h

Description

mpiStatusMaskBIT converts the status flag into the status mask.

See Also

MPIStatusMask

HostService Objects

Introduction

A **HostService** object allows users to synchronize their application to the "heartbeat" of the controller. It can be useful to monitor the health of the DSP, as a deterministic non-busy application delay, or even to modify SynqNet packets in real-time.

XMP and ZMP-SynqNet series motion controllers have the ability to interrupt the host at a specified frequency, where the frequency is a multiple of the controller's sample rate. This board-to-host interrupt is referred to as the Sync Interrupt. This object spawns a thread, which waits for the Sync Interrupt and then executes a user defined *Sync Interrupt Service Routine* (or SISR) in the user space.

The primary use for the HostService object is for host systems that need to process incoming SynqNet data from the controller and then write data back to the controller before a SynqNet transmission occurs. This is a convenient way to calculate control loops or inject values into SynqNet data packets in real-time. This type of operation will usually require a real-time operating system (VxWorks and RTX/RTSS).

The controller also has an optional watchdog timer, named the HostProcessFlag, which verifies that data written by the SISR is valid before it's sent over the SynqNet network.

The hostService object is provided as an interface to the Sync Interrupt feature. It provides code for thread-creation/deletion, setting/clearing the HostProcessFlag, and other common operations required for proper Synq Interrupt processing.

For more information about Sync Interrupt, please see the Synq Interrupt page.

NOTE: The HostService object is not part of the standard MPI. In order to use the Service Object, the *apputil.h* file needs to be included by your code and the apputil library needs to be linked to your application. In addition, thread handling is something that is different on every operating system. Therefore, HostService objects may have different behaviors on different operating systems. Programmers that are experienced in multi-threaded application programming may want to program their own host service SISR threads (see the hostService1.c and hostService2.c sample applications for a Win32 example). The source-code for this object is available in the (c:\MEI)\apputil project directory.

WARNING: Using the hostService object simultaneously with an interrupt-driven EventMgr service thread under Windows is not currently supported. However, using a polling EventMgr service thread at the same time as the hostService object is supported.

Methods

Create, Delete, Validate Methods

hostService**Create**	Create a HostService object, an SISR thread, and attach a serviceFunction for a given Control object.
hostService**Delete**	Stop SISR thread and delete the HostService object.

Configuration and Information Methods

hostService**Enable**	Enable or disable the HostService SIST thread from executing the serviceFunction. Disabled by default.
hostService**Status**	Returns status structure for HostService object.

Data Types

HostService**Function**

HostService**Status**

hostServiceCreate

Declaration

```
const HostService hostServiceCreate(MPIControl           control,
                                    HostServiceFunction  serviceFunction,
                                    void                 *params,
                                    long                 interruptPeriod,
                                    MPI_BOOL             enableWatchdog,
                                    long                 priority)
```

Required Header: service.h

Description

hostServiceCreate creates an SISR thread running at the specified ***priority*** for the given ***control*** object that executes the ***serviceFunction*** every ***interruptPeriod*** samples. After the create is called, the Sync interrupt occurs. By default, the service function is disabled from executing. Use hostServiceEnable(...) to enable the execution of the ***serviceFunction***.

control	a handle to a Control object.
serviceFunction	a handle to a user defined Sync Interrupt Service Routine (SISR) function.
***params**	points to a user defined structure that will be passed to **serviceFunction**.
interruptPeriod	specifies the frequency of the Sync Interrupt (where the frequency is a multiple of the controller's sample rate). This value must be greater that zero. A value of **1** generates an interrupt every controller sample. A value of **2** generates an interrupt every other sample, etc.
enableWatchdog	if TRUE, the SISR thread sets the HostProcessFlag watchdog flag in the firmware just before executing the **serviceFunction**. When the **serviceFunction** is complete, the SISR thread will clear the watchdog flag. If the watchdog flag is still set when the firmware starts SynqNet transmission, the controller will generate a MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED event. The event indicates that the host's serviceFunction did not complete before SynqNet transmission. This should be used if the host is trying to (for example) process feedback and calculate a new DAC command for a drive located on the network.

priority	is a platform specific variable. **If "priority" is** / **Then** -1 — hostServiceCreate will attempt to assign the maximum priority to the hostService thread. >0 — hostServiceCreate will attempt to assign the priority to the hostservice thread. **NOTE**: To avoid CPU usage competition, it is recommended that you run this hostService thread at a priority slightly higher than the apputil service thread.

Return Values	
handle	to a HostService object.
MPIHandleVOID	if the HostService could not be created

See Also

hostServiceDelete | hostServiceEnable | hostServiceStatus

hostService1.c | hostService2.c

hostServiceDelete

Declaration

```
long hostServiceDelete(HostService  service)
```

Required Header: apputil.h

Description

hostServiceDelete alerts the SISR thread to end, polls for the thread to exit, and frees the memory allocated to ***service***.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

hostServiceCreate | hostServiceEnable | hostServiceStatus

hostService1.c | hostService2.c

hostServiceEnable

Declaration

```
long hostServiceEnable(HostService    service,
                       MPI_BOOL       enable)
```

Required Header: apputil.h

Description

hostServiceEnable enables and/or disables the host service SISR thread from executing the ***serviceFunction***. This does not kill the hostService SISR thread. This method is provided to enable or temporarily disable the serviceFunction from servicing host interrupt events.

NOTE: By default, the SISR thread runs with the serviceFunction disabled. This method needs to be called after a hostServiceCreate(...) to enable the execution of the serviceFunction.

If "enabled" is	*Then*
FALSE	HostServiceEnable will disable service.
TRUE	HostServiceEnable will enable service.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

hostServiceCreate | hostServiceDelete | hostServiceStatus

hostService1.c | hostService2.c

hostServiceStatus

Declaration

```
long hostServiceStatus(HostService        service,
                       HostServiceStatus  *status)
```

Required Header: apputil.h

Description

hostServiceStatus returns the status structure for the hostService object.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

hostServiceCreate | hostServiceDelete | hostServiceEnabled

hostService1.c | hostService2.c

HostServiceFunction

Definition

```
typedef long (MPI_DEF2 *HostServiceFunction)(void *params);
```

Description

Interface prototype for a hostService service function. **HostServiceFunction** can be used to define a custom routine to service sync interrupts. HostServiceFunction function must take a pointer to a structure as a parameter and must return a *long*.

See Also

hostServiceCreate | hostServiceDelete | hostServiceStatus

hostService1.c | hostService2.c

HostServiceStatus

Definition

```
typedef struct HostServiceStatus {
    MPI_BOOL    enabled;
} HostServiceStatus;
```

Description

HostServiceStatus is a structure used to return the status of the hostService object.

enabled

If "enabled" is	*Then*
FALSE (0)	The hostService SISR function is disabled from servicing host interrupts.
TRUE (1)	The hostService SISR function is enabled and is servicing host interrupts.

See Also

hostServiceCreate | hostServiceDelete

hostService1.c | hostService2.c

Message Objects

Introduction

The **Message** module manages text strings associated with the error/warning code return values that can be returned from MPI methods. Each method return value has a unique value which can be decoded using the Message methods and macros.

For example, a return value of MEISynqNetMessagePLL_ERROR has a defined value of 0x192E. Passing this value to the mpiMessage(...) function returns the following text string:

"SynqNet: node PLL unable to lock with drive"

If MEISynqNetMessagePLL_ERROR was the last return value received from an MPI method, the string returned from mpiMessage(...) would also contain extended message information. The extended message information provides greater insight into the problem that has just occurred. In this case, the extended message information would indicate which node had the PLL locking problem:

"SynqNet: node PLL unable to lock with drive : Node 3"

Notice that the extended message information is delimited from the more generic message by a colon surrounded by spaces " : "

Extended message information may not always be returned. Extended information is only available for the very last return value from the MPI -- if a second (non-zero) return value has been returned between the time when the first return value was returned and the call to mpiMessage(...), then no extended information would be returned. In addition, not all messages return extended error information.

Methods

Configuration and Information Methods

mpiMessage	
mpiMessage**Function**	Associate message text with a function

Data Types

MPIMessage / MEIMessage

MPIMessage**Function**

Macros

mpiMessage**ID**

mpiMessage**MODULE**

mpiMessage**NUMBER**

mpiMessage

Declaration

```
const char *mpiMessage(long  messageId,
                       char  *messageText)
```

Required Header: stdmpi.h

Description

mpiMessage returns the text message associated with ***messageId*** and copies the text to ***messageText*** if ***messageText*** is not NULL.

If "messageText" is	Then
NULL	*Message* returns a pointer to message text resident in the library
not NULL	*Message* returns the pointer ***messageText***, which must point to a buffer large enough to hold the message text (that will be copied into it)

Return Values	
NULL	if messageId is invalid
Pointer to an empty string (" ")	if the messge utility is not enabled or the message function pointer associated with *messageId* is NULL[1].
Pointer to a non-zero-length string	if the mpiMessage successsfully retrieves the message associated with *messageId*.

1 - *ObjectCreate(...)* methods register the appropriate message function associated with *Object's* module. If the MPI returns a certain ***messageId***, then the appropriate message function should already be loaded and mpiMessage should return the appropriate error code.

Only an application whose job is to translate message IDs specified by some source other than the MPI (user or a log file) to error messages will need to call the create methods of all MPI objects. MEI recommends using the message.exe utility for this task, since it is already designed to manage these issues.

See Also

mpiMessageFunction | message.exe utility

mpiMessageFunction

Declaration

```
long mpiMessageFunction(MPIModuleId        moduleId,
                        MPIMessageFunction function)
```

Required Header: stdmpi.h

Description

mpiMessageFunction registers ***function*** as the function to be called by mpiMessage(...) (in order to obtain the text for a message associated with module ***moduleId***).

MessageFunction is typically called internally by object create methods. Applications generally do not need to call *MessageFunction* directly.

Return Values	
MPIMessageOK	

See Also

mpiMessage

MPIMessage / MEIMessage

Definition: MPIMotorMessage

```
typedef enum {
    MPIMessageOK,

    MPIMessageARG_INVALID,
    MPIMessagePARAM_INVALID,
    MPIMessageHANDLE_INVALID,

    MPIMessageNO_MEMORY,

    MPIMessageOBJECT_FREED,
    MPIMessageOBJECT_NOT_ENABLED,
    MPIMessageOBJECT_NOT_FOUND,
    MPIMessageOBJECT_ON_LIST,
    MPIMessageOBJECT_IN_USE,

    MPIMessageTIMEOUT,

    MPIMessageUNSUPPORTED,
    MPIMessageFATAL_ERROR,

    MPIMessageFILE_CLOSE_ERROR,
    MPIMessageFILE_OPEN_ERROR,
    MPIMessageFILE_READ_ERROR,
    MPIMessageFILE_WRITE_ERROR,
    MPIMessageFILE_MISMATCH,
} MPIMessage;
```

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

Description

MPIMessageOK

OK

MPIMessageARG_INVALID

Argument invalid

This message code is returned by a method when one or more arguments fail an integrity check. The following integrity checks are made, depending on the argument types:
- Pointers and addresses are not NULL.
- Memory addresses are within a valid range.
- Object maps have a valid number of members.

Possible Causes:
Number of axes on the motion commanded exceeds MEIXmpMAX_COORD_AXES.

MPIMessagePARAM_INVALID

Parameter invalid.

Parameters are the data inside an argument. This message code is returned by a method when one or more parameters fail an integrity check. The following integrity checks are made, depending on the parameter types:
- Pointers and addresses in structures are not NULL.
- Memory addresses in structures are within a valid range.
- Object numbers are within a valid range.
- Values in structures are within a valid range.

Please see possible causes for receiving this message.

MPIMessageHANDLE_INVALID

An object handle is not valid. This message code is returned by a method when the object handle argument is NULL. To correct this problem, create an object using an object create method. For example, to create an axis object use mpiAxisCreate(...).

MPIMessageNO_MEMORY

Memory is not available. This message code is returned by a method when an object, thread or buffer is not created due to a memory allocation failure. All host memory allocation occurs through meiPlatformAlloc(...). To correct this problem, check your host computer resources and remove unused components or add more memory. Note: Some methods may make several memory allocations or may call other methods that allocate memory. If a memory allocation fails, the previous allocations will not be freed.

MPIMessageOBJECT_FREED

The object has been freed. This message code is set within the object when the object is deleted. This message code is used internally. This message code is returned by an object delete method if the object has already been deleted. To prevent this problem, make sure to delete objects only one time.

MPIMessageOBJECT_NOT_ENABLED

The object is not active in the controller. This message code is returned by a method if real-time data or a handshake is required between the MPI and the controller, and the specified object is not enabled in the controller. To correct this problem, use mpiControlConfigSet(...) to enable the object, by setting the object count to greater than the specified object number. For example, to enable motor objects 0 to 3, set motorCount to 4.

MPIMessageOBJECT_NOT_FOUND

The object is not found on an object list. This message code is returned by a method when an object does not exist on the specified list or the object list does not exist. This message code can be returned from methods that insert or remove objects from a list. To correct this problem, specify an object that exists on the list or specify a different object list containing the object. To prevent object creation/ deletion problems, delete objects in the reverse order they were created.

MPIMessageOBJECT_ON_LIST

The object is a member of a list. This message code is returned when a method tries to add an object to a list and the object already exists on the list. It is also returned when a method tries to delete an object when the object exists on a list. For example, if an axis object is a member of a list of axis objects or an axis object is associated with a motion object, calling mpiAxisDelete(...) will return the object on list message code. To correct this problem when adding an object to a list, check to make sure the object is not already a member. To correct this problem when deleting an object, remove the object from the list and delete the parent object. To prevent object creation/deletion problems, delete objects in the reverse order they were created.

MPIMessageOBJECT_IN_USE

This message is returned when an operation cannot be completed because the object is currently in use. For example, when threadDelete(…) from the apputil library attempts to delete a thread that is currently running, MPIMessageOBJECT_IN_USE will be returned. To correct this problem, make sure the object is not in use before you attempt the operation.

MPIMessageTIMEOUT

The wait time has expired. This message code is returned by a method waiting for a response from the controller, a hardware resource, or a semaphore lock and the maximum wait time value expired. The library uses timeout values to prevent lock-up conditions when unexpected behavior occurs. To correct this problem, check the hardware health, power, and network connections.

MPIMessageUNSUPPORTED

The software or controller does not support a feature. This message code is returned by a method if the MPI library, controller, or network device does not support a feature.

Possible Causes:
Unsupported motion type specified.

List of supported motion types:
MPIMotionTypePT
MPIMotionTypePTF
MPIMotionTypePVT
MPIMotionTypePVTF
MPIMotionTypeSPLINE
MPIMotionTypeBESSEL
MPIMotionTypeBSPLINE
MPIMotionTypeBSPLINE2
MPIMotionTypeS_CURVE
MPIMotionTypeS_CURVE_JERK
MPIMotionTypeTRAPEZOIDAL
MPIMotionTypeVELOCITY
MPIMotionTypeVELOCITY_JERK
MEIMotionTypeFRAME

See MPIMotionType/MEIMotionType.

MPIMessageFATAL_ERROR

The software or hardware failed. This message code is returned by a method when an inexplicable data value is read from the controller or host memory. This message code indicates a serious problem with the host computer, memory, or controller. Contact an MEI applications engineer if you receive this message code.

MPIMessageFILE_CLOSE_ERROR

An error occurred when closing a file. This message code is returned by a method that fails to close a file. Internally, files are closed using meiPlatformClose(...) which makes a platform specific call. File closing errors indicate a missing file, bad file, or other file system problems.

MPIMessageFILE_OPEN_ERROR

An error occurred when opening a file. This message code is returned by a method that fails to open a file. Internally, files are opened using meiPlatformOpen(...) which makes a platform specific call. File opening errors indicate a missing file, bad file, or other file system problems.

MPIMessageFILE_READ_ERROR

An error occurred when reading from a file. This message code is returned by a method that fails to read from a file. Internally, files are read using meiPlatformFileRead(...) which makes a platform specific call. File reading errors indicate a bad file or other file system problems.

MPIMessageFILE_WRITE_ERROR

An error occurred when writing to a file. This message code is returned by a method that fails to write to a file. Internally, files are written using meiPlatformFileWrite(...) which makes a platform specific call. File writing errors indicate a bad file or other file system problems.

MPIMessageFILE_MISMATCH

The file being downloaded does not match the installed hardware. For example, if a user attempts to download an XMP file to a ZMP controller or a ZMP file to an XMP controller , the MPI will return error code MPIMessageFILE_MISMATCH.

Definition: MEIMotorMessage

```
typedef enum {
    MEIMotorMessageMOTOR_NOT_ENABLED,
    MEIMotorMessageSECONDARY_ENCODER_NA,
    MEIMotorMessageHARDWARE_NOT_FOUND,
    MEIMotorMessageSTEPPER_INVALID,
    MEIMotorMessageDISABLE_ACTION_INVALID,
    MEIMotorMessagePULSE_WIDTH_INVALID,
    MEIMotorMessageFEEDBACK_REVERSAL_NA,
    MEIMotorMessageFILTER_DISABLE_NA,
} MEIMotorMessage;
```

Required Header: stdmei.h

Description

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

MEIMotorMessageMOTOR_NOT_ENABLED

The motor number is not active in the controller. This message code is returned by MPIMotorEventConfig if the specified motor is not enabled in the controller. To correct the problem, use mpiControlConfigSet(…) to enable the motor object, by setting the motorCount to greater than the motor number. For example, to enable motor 0 to 3, set motorCount to 4.

MEIMotorMessageSECONDARY_ENCODER_NA

The motor's secondary encoder is not available. This message code is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the encoder fault trigger is configured for a secondary encoder when the hardware does not support a secondary encoder. To correct the problem, do not select the secondary encoder when configuring the encoder fault conditions.

MEIMotorMessageHARDWARE_NOT_FOUND

The motor object's hardware resource is not available. This message code is returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the node hardware for the motor is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor objects. An application should not configure a motor object if there is no mapped hardware to receive the service commands. To correct this problem, verify that all expected nodes were found. Use meiSqNodeInfo(…) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEIMotorMessageSTEPPER_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessageDISABLE_ACTION_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessagePULSE_WIDTH_INVALID

The motor object stepper pulse width is not valid. The pulse width must be no greater than 1 millisecond and no less than 100 nanoseconds.

MEIMotorMessageFEEDBACK_REVERSAL_NA

Feedback reversal is not applicable or is not supported for the feedback type specified.

MEIMotorMessageFILTER_DISABLE_NA

Disabling of filters is not applicable or is not supported for the feedback type specified.

See Also

MPIMessageFunction

Definition

```
typedef const char *(*MPIMessageFunction)(long);
```

Description

MPIMessageFunction is the type definition for the callback function used by mpiMessage(...). A default callback function is provided internally to all MPI/MEI modules, but an application can also be written to override it and provide a custom message function instead.

See Also

mpiMessage | mpiMessageFunction

mpiMessageID

Declaration

```
#define mpiMessageID(module number) \
((long)((((module) & MPIModuleIdMAX) << 8) | \ ((number) & 0xFF)))
```

Required Header: stdmpi.h

Description

mpiMessageID converts the message module value and number to a unique message identification value.

See Also

mpiMessage

mpiMessageMODULE

Declaration

```
#define mpiMessageMODULE(messageId) \
(((messageId) & (MPIModuleIdMAX << 16)) >> 16)
```

Required Header: stdmpi.h

Description

mpiMessageMODULE converts the message identification value to the message module value.

See Also

mpiMessage

mpiMessageNUMBER

Declaration

```
#define mpiMessageNUMBER(messageId) ((messageId) & 0xFF)
```

Required Header: stdmpi.h

Description

mpiMessageNUMBER converts the message identification value to the message number.

See Also

mpiMessage

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

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.

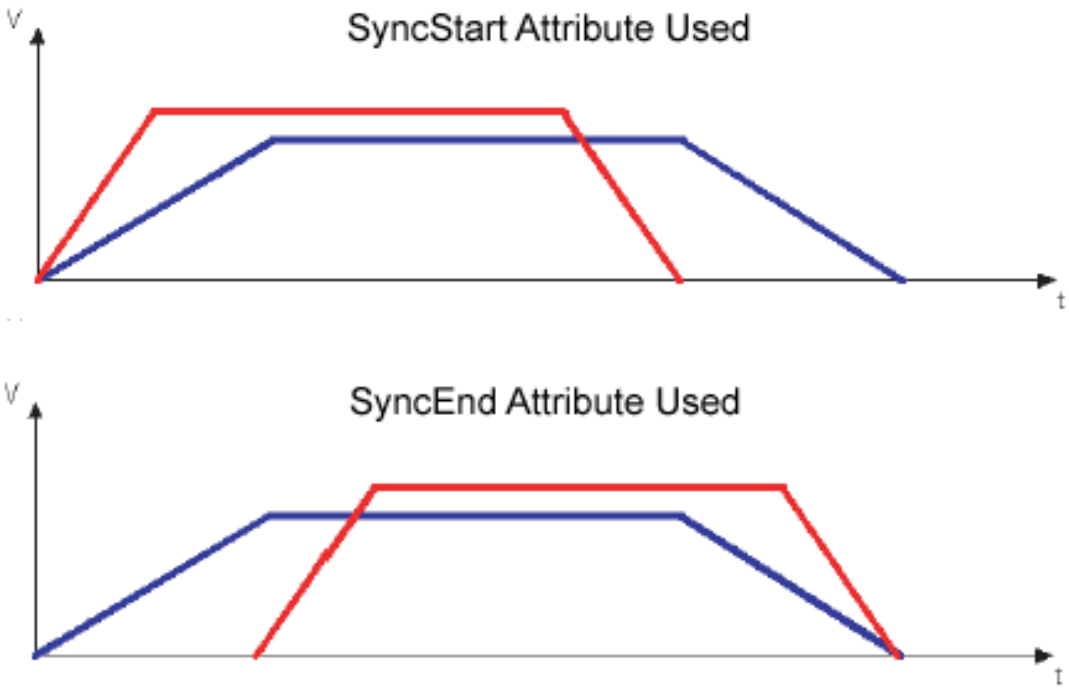
MPIMotionAttrMaskNO_BACKTRACK_HOLD	This attribute modifies a cam motion so that when the master changes direction the slave axes will hold that position until the master returns to the point where the direction changed.This attribute cannot be specified with the MPIMotionAttrFORWARD_ONLY attribute. See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

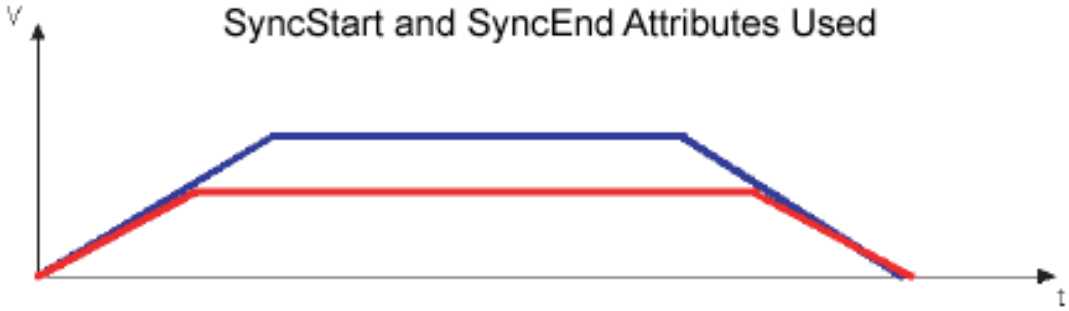
Remarks

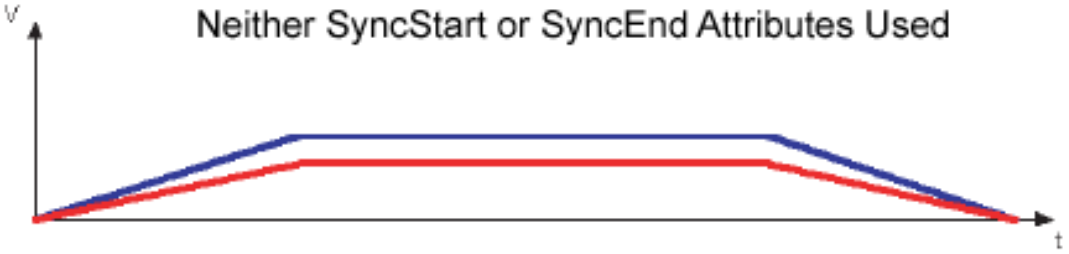
The motion attribute masks are used to enable features with mpiMotionStart(...) and mpiMotionModify (...). The masks are **OR**ed with the MPIMotionType to enable each feature.

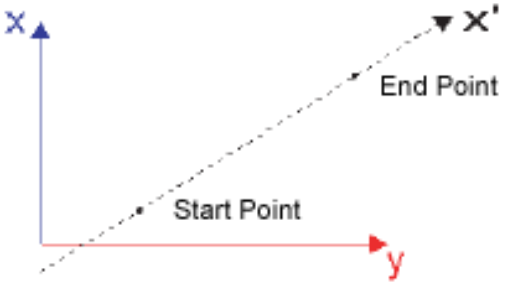
For the motion types MPIMotionTypeS_CURVE_JERK, MPIMotionTypeS_CURVE, MPIMotionTypeTRAPEZOIDAL, if neither MPIMotionAttrMaskSYNC_START nor MPIMotionAttrMaskSYNC_END are specified, then only one MPITrajectory structure will be used for by mpiMotionStart() and mpiMotionModify(). Please refer to MPIMotionAttr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

Motor Objects

Introduction

A **Motor** object manages a single motor on a controller. It represents the physical connections between the motor, drive, and associated I/O. The Motor object contains encoder data, limit switch, home sensor, amp fault and amp enable states, DAC outputs, and other status information.

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

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiMotor**Create**	Create Motor object
mpiMotor**Delete**	Delete Motor object
mpiMotor**Validate**	Validate Motor object

Configuration and Information Methods

mpiMotor**AmpEnableGet**	Get state of amp enable output
mpiMotor**AmpEnableSet**	Set state of amp enable output
meiMotor**AmpFault**	Writes the amp fault buffer data of a motor.
meiMotor**AmpFaultClear**	Clears the amp fault buffer data of a motor.
meiMotor**AmpWarning**	Writes the amp warning buffer of the motor.
meiMotor**AmpWarningClear**	Clears the motor's amp wanring message buffer.
mpiMotor**AxisMapGet**	Get object map of axes
meiMotor**CommutationModeGet**	Gets the commutation mode of a motor.
meiMotor**CommutationModeSet**	Sets the commutation mode of a motor.
mpiMotor**ConfigGet**	Get motor configuration
mpiMotor**ConfigSet**	Set motor configuration
meiMotor**ConfigStepper**	Configures a motor for stepper mode.
meiMotor**DacConfigGet**	Get a Motor's (motor) Dac configuration
meiMotor**DacConfigSet**	Set a Motor's (motor) Dac configuration
meiMotor**DacFlashConfigGet**	
meiMotor**DacFlashConfigSet**	
mpiMotor**DedicatedIn**	
mpiMotor**DedicatedOutGet**	
mpiMotor**Feedback**	Get feedback position
mpiMotor**FlashConfigGet**	Get flash config of motor

mpiMotor**FlashConfigSet**	Set flash config of motor
mpiMotor**GeneralIn**	
mpiMotor**GeneralOutGet**	
mpiMotor**GeneralOutSet**	
meiMotor**Info**	Get information about the network, node, and drive interface
meiMotor**PhaseFindStatus**	
mpiMotor**Status** / meiMotor**Status**	Get motor status

Event Methods

mpiMotor**EventConfigGet**	Get motor's event configuration
mpiMotor**EventConfigSet**	Set motor's event configuration
mpiMotor**EventNotifyGet**	Get motor's event mask for host notification.
mpiMotor**EventNotifySet**	Set motor's event mask for host notification.
mpiMotor**EventReset**	Reset events specified in event mask

Memory Methods

mpiMotor**Memory**	Get address of motor memory
mpiMotor**MemoryGet**	Copy motor memory to application memory
mpiMotor**MemorySet**	Copy application memory to motor memory

Action Methods

meiMotor**EncoderReset**	Clears encoder faults.
meiMotor**MultiTurnReset**	Clears the SynqNet drive multi-turn data for absolute type encoders.
meiMotor**PhaseFindAbort**	
meiMotor**PhaseFindStart**	

Relational Methods

mpiMotor**Control**	Get handle to associated Control object
mpiMotor**FilterMapGet**	Get object map of associated Filters
mpiMotor**FilterMapSet**	Set the Filters using object map
mpiMotor**Number**	Get index number of motor (in Control list)

Other Methods

meiMotor**EncoderRatio**	Get encoder ratio from the XMP.

Data Types

MEIMotor**AmpFaults**

MEIMotor**AmpFaultMsg**

MEIMotor**AmpWarnings**

MEIMotor**AmpWarningMsg**

MPIMotor**Brake**

MPIMotor**BrakeMode**

MPIMotor**Config** / MEIMotor**Config**

MEIMotor**DacConfig**

MEIMotor**DacChannelConfig**

MEIMotor**DacChannelStatus**

MEIMotor**DacStatus**

MPIMotor**DedicatedIn**

MPIMotor**DedicatedOut**

MEIMotor**DemandMode**

MEIMotor**DisableAction**

MPIMotor**Encoder** / MEIMotor**Encoder**

MPIMotor**EncoderFault**

MPIMotor**EncoderFaultMask**

MEIMotor**EncoderModulo**

MEIMotor**EncoderRatio**

MEIMotor**EncoderReverseModulo**

MEIMotor**EncoderSsiConfig**

MEIMotor**EncoderType**

MPIMotor**EventConfig** / MEIMotor**EventConfig**

MPIMotor**EventTrigger**

MEIMotor**FaultBit**

MEIMotor**FaultConfig**

MEIMotor**FaultMask**

MPIMotor**Feedback**

MPIMotor**GeneralIo**

MEIMotor**Info**

MEIMotor**InfoDedicatedIn**

MEIMotor**InfoDedicatedOut**

MEIMotor**InfoGeneralIo**

MEIMotor**InfoNodeType**

MEIMotor**IoConfig**

MEIMotor**IoConfigIndex**

MEIMotor**IoType**

MEIMotor**IoTypeMask**
MPIMotor**Message** / MEIMotor**Message**
MEIMotor**PhaseFindDriveMsg**
MEIMotor**PhaseFindState**
MEIMotor**PhaseFindStatus**
MEIMotor**SsiInput**
MEIMotor**Status**
MEIMotor**StatusOutput**
MEIMotor**Stepper**
MEIMotor**StepperPulse**
MEIMotor**StepperPulseType**
MEIMotor**StepperStatus**
MPIMotor**Type**

Macros

mpiMotor**EncoderFaultMaskBIT**

Constants

MEIMotor**AmpFaultsMAX**
MEIMotor**AmpMsgMAX**
MEIMotor**AmpWarningsMAX**

mpiMotorCreate

Declaration

```
const MPIMotor mpiMotorCreate(MPIControl control,
                              long       number);
```

Required Header: stdmpi.h

Description

mpiMotorCreate creates a Motor object associated with the motor identified by ***number***, and located on the motion controller (***control***). MotorCreate is the equivalent of a C++ constructor.

Return Values	
handle	to a Motor object.
MPIHandleVOID	if the Motor object could not be created.

See Also

mpiMotorDelete | mpiMotorValidate

mpiMotorDelete

Declaration

```
long mpiMotorDelete(MPIMotor motor)
```

Required Header: stdmpi.h

Description

mpiMotorDelete deletes a Motor object and invalidates its handle (***motor***). *MotorDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiMotorCreate | mpiMotorValidate

mpiMotorValidate

Declaration

```
long mpiMotorValidate(MPIMotor motor)
```

Required Header: stdmpi.h

Description

mpiMotorValidate validates a Motor object and its handle (***motor***).

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

mpiMotorCreate | mpiMotorDelete

mpiMotorAmpEnableGet

Declaration

```
long mpiMotorAmpEnableGet(MPIMotor  motor,
                          MPI_BOOL*ampEnable)
```

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

Description

mpiMotorAmpEnableGet gets the state of the amp enable output for a Motor (***motor***) and writes it in the location pointed to by ***ampEnable***. Note that the actual state of amp enable output also depends upon the actual wiring and the polarity chosen in the instance of the MPIMotorConfig structure.

If "ampEnable" is	*Then*
FALSE (0)	the amp is disabled
TRUE (1)	the amp is enabled

Return Values	
MPIMessageOK	

See Also

MPIMotorConfig | mpiMotorAmpEnableSet

mpiMotorAmpEnableSet

Declaration

```
long mpiMotorAmpEnableSet(MPIMotor     motor,
                          MPI_BOOL     ampEnable)
```

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

Description

mpiMotorAmpEnableSet sets the state of the amp enable output for a Motor (***motor***) to ***ampEnable***. Note that the actual state of amp enable output also depends upon the actual wiring and the polarity chosen in the instance of the MPIMotorConfig structure.

If "ampEnable" is	*Then*
FALSE (0)	the amp will be disabled
TRUE (1)	the amp will be enabled

Return Values	
MPIMessageOK	

See Also

MPIMotorConfig | mpiMotorAmpEnableGet

meiMotorAmpFault

Declaration

```
long meiMotorAmpFault(MPIMotor           motor,
                      MEIMotorAmpFaults  *fault);
```

Required Header: stdmei.h

Description

meiMotorAmpFault reads the amp fault buffer from a motor and writes the data into a structure pointed to by fault.

motor	a handle to the Motor object
***fault**	a pointer to a structure containing the number of amp faults, their coded values and message strings. See MEIMotorAmpFaults.

Return Values	
MPIMessageOK	

See Also

meiMotorAmpFaultClear | meiMotorAmpWarning | meiMotorWarningClear

meiMotorAmpFaultClear

Declaration

```
long meiMotorAmpFaultClear(MPIMotor    motor);
```

Required Header: stdmei.h

Description

meiMotorAmpFaultClear flushes the motor's amp fault message buffer. The number of amp faults is set to zero, the coded values are set to zero, and the messages are cleared.

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

meiMotorAmpFault | meiMotorAmpWarning | meiMotorWarningClear

meiMotorAmpWarning

Declaration

```
long meiMotorAmpWarning(MPIMotor              motor,
                        MEIMotorAmpWarnings   *warning);
```

Required Header: stdmei.h

Description

meiMotorAmpWarning reads the amp warning buffer from a motor and writes the data into a structure pointed to by warning.

motor	a handle to the Motor object
***warning**	a pointer to a structure containing the number of amp warnings, their coded values and message strings. See MEIMotorAmpWarnings.

Return Values	
MPIMessageOK	

See Also

meiMotorAmpWarningClear | meiMotorAmpFault | meiMotorAmpFaultClear

meiMotorAmpWarningClear

Declaration

```
long meiMotorAmpWarningClear(MPIMotor    motor);
```

Required Header: stdmei.h

Description

meiMotorAmpWarningClear flushes the motor's amp warning message buffer. The number of amp warnings is set to zero, the coded values are set to zero, and the messages are cleared.

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

meiMotorAmpWarning | meiMotorAmpFault | meiMotorAmpFaultClear

mpiMotorAxisMapGet

Declaration

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

Required Header: stdmpi.h

Description

mpiMotorAxisMapGet gets the object map of the Axes associated with a Motor (**motor**) and writes it into the structure pointed to by **axismap**.

motor	a handle to the Motor object
***axismap**	a pointer to an object map. An ObjectMap is a bitmap, where each numbered bit represents the presence or absence of the correspondingly numbered object.

Return Values	
MPIMessageOK	

See Also

meiMotorCommutationModeGet

Declaration

```
long meiMotorCommutationModeGet(MPIMotor       motor,
                                MEIXmpCommMode *mode)
```

Required Header: stdmei.h

Description

meiMotorCommutationModeGet gets the commutation mode of a Motor (***motor***) and writes it to the location pointed to by ***mode***.

Return Values	
MPIMessageOK	

See Also

meiMotorCommutationModeSet

meiMotorCommutationModeSet

Declaration

```
long meiMotorCommutationModeSet(MPIMotor       motor,
                                MEIXmpCommMode mode)
```

Required Header: stdmei.h

Description

meiMotorCommutationModeSet sets the commutation mode of a Motor (***motor***) to ***mode***.

Return Values	
MPIMessageOK	

See Also

meiMotorCommutationModeGet

mpiMotorConfigGet

Declaration

```
long mpiMotorConfigGet(MPIMotor       motor,
                       MPIMotorConfig *config,
                       void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorConfigGet gets a Motor's (***motor***) configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

Return Values	
MPIMessageOK	
MEIMotorMessageDEMAND_MODE_NOT_SET	
MEIMotorMessageDEMAND_MODE_UNSUPPORTED	

Sample Code

```
MEIMotorConfig motorConfig;
mpiMotorConfigGet( motor0, NULL, &motorConfig );

motorConfig.Io[0].Type = MEIMotorIoTypeBRAKE;

mpiMotorConfigSet( motor0, NULL, &motorConfig );
```

See Also

MEIMotorConfig | mpiMotorConfigSet

mpiMotorConfigSet

Declaration

```
long mpiMotorConfigSet(MPIMotor       motor,
                       MPIMotorConfig *config,
                       void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorConfigSet sets a Motor's (***motor***) configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

Return Values	
MPIMessageOK	
MEIMotorMessageDEMAND_MODE_NOT_SET	
MEIMotorMessageDEMAND_MODE_UNSUPPORTED	

Sample Code

```
MEIMotorConfig motorConfig;
mpiMotorConfigGet( motor0, NULL, &motorConfig );

motorConfig.Io[0].Type = MEIMotorIoTypeBRAKE;

mpiMotorConfigSet( motor0, NULL, &motorConfig );
```

See Also

mpiMotorConfigGet | MEIMotorConfig
Special Note: Using mpiMotorConfigSet with Absolute Encoders

meiMotorConfigStepper

Declaration

```
long meiMotorConfigStepper(MPIMotor       motor,
                           MEIMotorConfig *config,
                           long           stepperNumber)
```

Required Header: stdmei.h

Description

meiMotorConfigStepper modifies the motor configuration structure pointed to by config, to use a step engine (stepperNumber) from another motor. By default, each motor uses its own step engine. Do NOT use more than one motor per step engine. Use the methods mpiMotorConfigGet/Set(...) to read/write the motor configuration from/to the controller.

motor	a handle to the Motion object
***config**	a pointer to the motion frame buffer status structure returned by the method
stepperNumber	index to a step engine

Return Values	
MPIMessageOK	

See Also

mpiMotorConfigGet | mpiMotorConfigSet

meiMotorDacConfigGet

Declaration

```
long meiMotorDacConfigGet(MPIMotor           motor,
                          MEIMotorDacConfig  *config);
```

Required Header: stdmei.h

Description

meiMotorDacConfigGet gets a Motor's (***motor***) DAC configuration and writes it to the structure pointed to by ***config***.

motor	a handle to the Motor object.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacConfigSet | MEIMotorDacConfig

meiMotorDacConfigSet

Declaration

```
long meiMotorDacConfigSet(MPIMotor          motor,
                          MEIMotorDacConfig *config);
```

Required Header: stdmei.h

Description

meiMotorDacConfigSet configures a Motor's (***motor***) DAC using data from the structure pointed to by ***config***.

motor	a handle to the Motor object.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacConfigGet | MEIMotorDacConfig

meiMotorDacFlashConfigGet

Declaration

```
long meiMotorDacFlashConfigGet(MPIMotor           motor,
                               void               *flash,
                               MEIMotorDacConfig  *config);
```

Required Header: stdmei.h

Description

meiMotorDacFlashConfigGet gets a Motor's (***motor***) DAC configuration from flash memory and writes it to the structure pointed to by ***config***.

motor	a handle to the Motor 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 MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacFlashConfigSet | meiMotorDacConfigGet | MEIMotorDacConfig

meiMotorDacFlashConfigSet

Declaration

```
long meiMotorDacFlashConfigSet(MPIMotor           motor,
                               void               *flash,
                               MEIMotorDacConfig  *config);
```

Required Header: stdmei.h

Description

meiMotorDacFlashConfigSet sets a Motor's (***motor***) DAC configuration to flash memory using data from the structure pointed to by ***config***.

motor	a handle to the Motor 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 MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacFlashConfigGet

mpiMotorDedicatedIn

Declaration

```
long mpiMotorDedicatedIn(MPIMotor        motor,
                         long            startBit,
                         long            bitCount,
                         unsigned long   *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00. mpiMotorDedicatedIn replaced mpiMotorIoGet.

Description

mpiMotorDedicatedIn function reads the current state of one or more dedicated input bits.

NOTE: mpiMotorDedicatedIn replaced mpiMotorIoGet in the MPI library.

motor	a handle to the Motor object
startBit	the first dedicated in bit that will be returned by the function.
bitCount	the number of dedicated in bits that will be returned by the function.
***state**	the address of the current state of the inputs that is returned.

Return Values	
MPIMessageOK	

See Also

Dedicated Motor I/O | MPIMotorDedicatedIn

mpiMotorDedicatedOutGet

Declaration

```
long mpiMotorDedicatedOutGet(MPIMotor       motor,
                             long           startBit,
                             long           bitCount,
                             unsigned long  *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00

Description

mpiMotorDedicatedOutGet gets the current state of one or more of the dedicated outputs.

motor	a handle to the Motor object
startBit	the first dedicated out bit that will be returned by the function.
bitCount	the number of dedicated out bits that will be returned by the function.
***state**	the address of where the current state of the outputs will be returned.

Return Values	
MPIMessageOK	

See Also

Dedicated Motor I/O | MPIMotorDedicatedOut

mpiMotorFeedback

Declaration

```
long mpiMotorFeedback(MPIMotor           motor,
                      MPIMotorFeedback   *feedback)
```

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiMotorFlashConfigGet

Declaration

```
long mpiMotorFlashConfigGet(MPIMotor       motor,
                            void           *flash,
                            MPIMotorConfig *config,
                            void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorFlashConfigGet gets a Motor's (***motor***) flash configuration and writes it in the structure pointed to by ***config***, and also writes it in the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

motor	a handle to a Motor 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 motor object of type MPIMotorConfig.
***external**	a pointer to a configuration structure for the motor object of type MEIMotorConfig.

Return Values	
MPIMessageOK	

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigSet

mpiMotorFlashConfigSet

Declaration

```
long mpiMotorFlashConfigSet(MPIMotor       motor,
                            void           *flash,
                            MPIMotorConfig *config,
                            void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorFlashConfigSet sets a Motor's (***motor***) flash configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

motor	a handle to a Motor 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 motor object of type MPIMotorConfig.
***external**	a pointer to a configuration structure for the motor object of type MEIMotorConfig.

Return Values	
MPIMessageOK	
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigGet | meiSynqNetFlashTopologySave

mpiMotorGeneralIn

Declaration

```
long mpiMotorGeneralIn(MPIMotor         motor,
                       long             startBit,
                       long             bitCount,
                       unsigned long    *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00. mpiMotorGeneralIn replaced mpiMotorIoGet.

Description

mpiMotorGeneralIn reads the current input state of one or more general purpose bits.

NOTE: mpiMotorGeneralIn replaced mpiMotorIoGet in the MPI library.

motor	a handle to the Motor object
startBit	the first general purpose bit that will be returned by the function.
bitCount	the number of general purpose bits that will be returned by the function.
***state**	the address of the current state of the inputs that is returned.

Return Values	
MPIMessageOK	

Sample Code

```
long x;
meiMotorGeneralIn( motor0, 0, 1, &x );
```

See Also

General Purpose Motor I/O

mpiMotorGeneralOutGet

Declaration

```
long mpiMotorGeneralOutGet(MPIMotor         motor,
                           long             startBit,
                           long             bitCount,
                           unsigned long    *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00. mpiMotorGeneralOutGet replaced mpiMotorIoGet.

Description

mpiMotorGeneralOutGet function reads the current output state of one or more general purpose bits.

NOTE: mpiMotorGeneralOutGet replaced mpiMotorIoGet in the MPI library.

motor	a handle to the Motor object
startBit	the first general purpose bit that will be returned by the function.
bitCount	the number of general purpose bits that will be returned by the function.
***state**	the address of the current state of the general purpose bits will be returned.

Return Values	
MPIMessageOK	

Sample Code

```
long x;
meiMotorGeneralOutGet( motor0, 0, MEIMotorGeneralIoLAST, &x );
```

See Also

General Purpose Motor I/O

mpiMotorGeneralOutSet

Declaration

```
long mpiMotorGeneralOutSet(MPIMotor         motor,
                           long             startBit,
                           long             bitCount,
                           unsigned long    state,
                           MPI_BOOL         wait);
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00
Added in 03.02.00 (mpiMotorGeneralOutSet replaced mpiMotorIoSet).

Description

mpiMotorGeneralOutSet function changes the state of one or more general purpose bits.

NOTE: mpiMotorGeneralOutSet replaced mpiMotorIoSet in the MPI library.

motor	a handle to the Motor object.
startBit	the first general purpose bit that will be set by the function.
bitCount	the number of general purpose bits that will be set by the function.
state	the new state of the general purpose bits.
wait	See Motor Digital Output Waits.

Return Values	
MPIMessageOK	

See Also

General Purpose Motor I/O | Motor Digital Output Waits

meiMotorInfo

Declaration

```
long meiMotorInfo(MPIMotor      motor,
                  MEIMotorInfo  *info);
```

Required Header: stdmei.h

Description

meiMotorInfo reads the static information about the network, node, and drive interface associated with the motor object, and writes it into the structure pointed to by ***info***.

motor	a handle to the Motor object
***info**	a pointer to a motor information structure

Return Values	
MPIMessageOK	

See Also

meiMotorStatus

meiMotorPhaseFindStatus

Declaration

```
long  meiMotorPhaseFindStatus(MPIMotor                    motor,
                              MEIMotorPhaseFindStatus*    status);
```

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

Description

meiMotorPhaseFindStatus provides the user with information which reflects the status of the drive's Phase Finding Procedure.

motor	a handle to the Motor object.
status	the current state of the phase finding process: in progress, failed, success. It also contains drive specific information pertaining to the current state of the phase finding procedure.

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindStart | meiMotorPhaseFindAbort

Motor Phase Finding

mpiMotorStatus / meiMotorStatus

Declaration: mpiMotorStatus

```
long mpiMotorStatus(MPIMotor    motor,
                    MPIStatus   *status,
                    void        *external)
```

Required Header: stdmpi.h

Description

mpiMotorStatus writes a Motor's (***motor***) status into the structure pointed to by ***status***, and also into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

motor	a handle to the Motor object
***status**	a pointer to the motor status structure returned by the method
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

Declaration: meiMotorStatus

```
long meiMotorStatus(MPIControl   control,
                    long         motorNumber
                    MPIStatus    *status,
                    void         *external)
```

Required Header: stdmei.h

Description

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

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

Remarks

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

control	a handle to the Control object
motorNumber	index to the motor
***status**	a pointer to the motor status structure returned by the method
***external**	pointer to an implementation-specific structure

Return Values	
MPIMessageOK	if *MotorStatus* successfully gets the status of a Motor object.
MPIMessageARG_INVALID	if the *status* pointer is NULL.

Sample Code

```
void readDACOutputs(MPIMotor motor)
{
        long returnValue;
        MPIStatus status;
        MEIMotorStatus motorStatus;

   returnValue = mpiMotorStatus(motor, &status, &motorStatus);
   msgCHECK(returnValue);

   printf("Output CMD = %.4f\n", motorStatus.dac.cmd.level);
   printf("Output AUX = %.4f\n", motorStatus.dac.aux.level);
}
```

See Also

MPIStatus | MEIMotorStatus

mpiMotorEventConfigGet

Declaration

```
long mpiMotorEventConfigGet(MPIMotor              motor,
                            MPIEventType          eventType,
                            MPIMotorEventConfig   *src,
                            void                  *external)
```

Required Header: stdmpi.h

Description

mpiMotorEventConfigGet gets the Motor's (***motor***) configuration for the event specified by ***eventType*** and writes it into the structure pointed to by ***eventConfig***, and also writes it to the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The event configuration information in ***external*** is in addition to the event configuration information in ***eventConfig***, i.e, the event configuration information in ***eventConfig*** and in ***external*** is not the same information.

NOTE: Set ***eventConfig*** or ***external*** to NULL. One must be NULL, the other must be passed a pointer.

Remarks

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

Return Values	
MPIMessageOK	

Sample Code

```
for(index = 0; index < AXIS_COUNT; index++)
  { // turn off error limit and limit switch actions for
    motors 0 to AXIS_COUNT
      returnValue = mpiMotorEventConfigGet(motor[index],
         MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
      msgCHECK(returnValue);

      eventConfig.action = MPIActionNONE;

      returnValue = mpiMotorEventConfigSet(motor[index],
         MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
      msgCHECK(returnValue);

      returnValue = mpiMotorEventConfigGet(motor[index],
         MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
      msgCHECK(returnValue);

      eventConfig.action = MPIActionNONE;

      returnValue = mpiMotorEventConfigSet(motor[index],
         MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
      msgCHECK(returnValue);

      returnValue = mpiMotorEventConfigGet(motor[index],
         MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
      msgCHECK(returnValue);

      eventConfig.action = MPIActionNONE;

      returnValue = mpiMotorEventConfigSet(motor[index],
         MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
      msgCHECK(returnValue);
}
```

See Also

MEIMotorEventConfig | mpiMotorEventConfigSet | Error Limit and Limit Switch Errors

mpiMotorEventConfigSet

Declaration

```
long mpiMotorEventConfigSet(MPIMotor            motor,
                            MPIEventType        eventType,
                            MPIMotorEventConfig *eventConfig,
                            void                *external)
```

Required Header: stdmpi.h

Description

mpiMotorEventConfigSet reads the structure pointed to by ***eventConfig*** and sets the Motor's (***motor***) configuration for the event specified by ***eventType***.

The event configuration information in ***external*** is in addition to the event configuration information in ***eventConfig***, i.e, the event configuration information in ***eventConfig*** and in ***external*** is not the same information.

NOTE: Set ***eventConfig*** or ***external*** to NULL. One must be NULL, the other must be passed a pointer.

Remarks

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

Return Values	
MPIMessageOK	

See Also

MEIMotorEventConfig | mpiMotorEventConfigGet | Error Limit and Limit Switch Errors

mpiMotorEventNotifyGet

Declaration

```
long mpiMotorEventNotifyGet(MPIMotor      motor,
                            MPIEventMask  *eventMask,
                            void          *external)
```

Required Header: stdmpi.h

Description

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

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

Event notification is enabled for event types specified in ***eventmask***, which is a bit mask of MPIEventMask bits associated with the desired MPIEventType values. Event notification is disabled for event types not specified in ***eventmask***. The MPIEventMask bits must be set of cleared using the MPIEventMask macros.

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 | mpiMotorEventNotifySet

mpiMotorEventNotifySet

Declaration

```
long mpiMotorEventNotifySet(MPIMotor     motor,
                            MPIEventMask eventMask,
                            void         *external)
```

Required Header: stdmpi.h

Description

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

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

Event notification is enabled for event types specified in ***eventMask***, a bit mask of MPIEventMask bits associated with the desired MPIEventType values. Event notification is disabled for event types that are not specified in ***eventMask***. The MPIEventMask bits must be set of cleared using the MPIEventMask macros.

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

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	

Sample Code

```
MPIEventMask eventMask;

mpiEventMaskCLEAR( eventMask );
mpiEventMaskALL( eventMask );
meiEventMaskALL( eventMask );

returnValue = mpiMotorEventNotifySet( motor, eventMask, NULL );
msgCHECK( returnValue );
```

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo | mpiMotorEventNotifyGet

mpiMotorEventReset

Declaration

```
long mpiMotorEventReset(MPIMotor      motor,
                        MPIEventMask  eventMask)
```

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiControlEventReset | mpiMotionEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | meiSqNodeEventReset | mpiAxisEventReset

Event Notification Methods

mpiMotorMemory

Declaration

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

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiMotorMemoryGet | mpiMotorMemorySet

mpiMotorMemoryGet

Declaration

```
long mpiMotorMemoryGet(MPIMotor      motor,
                       void          *dst,
                       const void    *src,
                       long          count)
```

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

Description

mpiMotorMemoryGet copies ***count*** bytes of a Motor's (***motor***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiMotorMemorySet | mpiMotorMemory

mpiMotorMemorySet

Declaration

```
long mpiMotorMemorySet(MPIMotor    motor,
                       void        *dst,
                       const void  *src,
                       long        count)
```

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

Description

mpiMotorMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Motor's (***motor***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiMotorMemoryGet | mpiMotorMemory

meiMotorEncoderReset

Declaration

```
long meiMotorEncoderReset(MPIMotor motor)
```

Required Header: stdmei.h

Description

meiMotorEncoderReset clears the encoder fault status registers for the primary and secondary encoder associated with the motor object.

Return Values	
MPIMessageOK	

See Also

meiMotorMultiTurnReset

Declaration

```
long meiMotorMultiTurnReset(MPIMotor    motor);
```

Required Header: stdmei.h

Description

meiMotorMultiTurnReset clears the SynqNet drive multi-turn data for absolute type encoders. This is only needed when configuring the zero location for an absolute encoder or when the absolute encoder's battery is replaced. meiMotorMultiTurnReset is an offline operation. Make sure all motors (amp enables) are disabled before executing a multi-turn reset. The SynqNet network may be shutdown (dropped from SYNQ mode) due to specific drive limitations.

Not all SynqNet drives support or require this feature. Please see the drive manufacturer's documentation for details.

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindAbort

Declaration

```
long meiMotorPhaseFindAbort(MPIMotor   motor);
```

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

Description

meiMotorPhaseFindAbort stops the phase finding process and disables the amplifier.

motor	a handle to the Motor object.

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindStart | meiMotorPhaseFindStatus

Motor Phase Finding

meiMotorPhaseFindStart

Declaration

```
long meiMotorPhaseFindStart(MPIMotor   motor);
```

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

Description

meiMotorPhaseFindStart activates a drive's phase finding process.

motor	a handle to the Motor object.

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

mpiMotorControl

Declaration

```
const MPIControl mpiMotorControl(MPIMotor motor)
```

Required Header: stdmpi.h

Description

mpiMotorControl returns a handle to the Control object with which the motor is associated.

motor	a handle to the Motor object

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

See Also

mpiMotorCreate | mpiControlCreate

mpiMotorFilterMapGet

Declaration

```
long mpiMotorFilterMapGet(MPIMotor     motor,
                          MPIObjectMap *filtermap)
```

Required Header: stdmpi.h

Description

mpiMotorFilterMapGet gets the object map of the Filters [that are associated with a Motor (***motor***)] and writes it into the structure pointed to by ***filtermap***.

Return Values	
MPIMessageOK	

See Also

mpiMotorFilterMapSet

mpiMotorFilterMapSet

Declaration

```
long mpiMotorFilterMapSet(MPIMotor     motor,
                          MPIObjectMap filtermap)
```

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiMotorFilterMapGet

mpiMotorNumber

Declaration

```
long mpiMotorNumber(MPIMotor motor,
                    long     *number)
```

Required Header: stdmpi.h

Description

mpiMotorNumber writes the index of a Motor (***motor***, on the motion controller that ***motor*** is associated with) to the contents of ***number***.

Return Values	
MPIMessageOK	

See Also

meiMotorEncoderRatio

Declaration

```
long meiMotorEncoderRatio(MPIControl           control,
                          long                 motorNumber,
                          long                 encoderNumber,
                          MEIMotorEncoderRatio *ratio)
```

Required Header: stdmei.h

Description

meiMotorEncoderRatio gets encoder ratio from the XMP.

WARNING: This is a customer-specific method that is only supported with custom firmware. To inquire about using this method, please contact an MEI Applications Engineer.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEIMotorAmpFaults

Definition

```
typedef struct MEIMotorAmpFaults {
    long                 count;
    long                 code[MEIMotorAmpFaultsMAX];
    MEIMotorAmpFaultMsg  message[MEIMotorAmpFaultsMAX];
} MEIMotorAmpFaults;
```

Description

MPIAxisInPosition contains the amp fault information from a SynqNet drive. The amp fault messages are drive specific. Not all drives support amp fault messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

count	The number of amp faults in the buffer.
code	An array of drive specific amp fault coded values.
message	An array of drive specific amp fault message strings.

See Also

meiMotorAmpFault | meiMotorAmpFaultClear

MEIMotorAmpFaultsMsg

Definition

```
typedef char   MEIMotorAmpFaultMsg[MEIMotorAmpMsgMAX];
```

Description

MEIMotorAmpFaultsMsg defines the amp fault message string definition.

See Also

MEIMotorAmpMsgMAX

MEIMotorAmpWarnings

Definition

```
typedef struct MEIMotorAmpWarnings {
    long                    count;
    long                    code[MEIMotorAmpWarningsMAX];
    MEIMotorAmpWarningMsg   message[MEIMotorAmpWarningsMAX];
} MEIMotorAmpWarnings;
```

Description

MEIMotorAmpWarnings contains the amp warning information from a SynqNet drive. The amp warning messages are drive specific. Not all drives support amp warning messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

count	The number of amp faults in the buffer.
code	An array of drive specific amp fault coded values.
message	An array of drive specific amp fault message strings.

See Also

meiMotorAmpWarning | meiMotorAmpWarningClear

MEIMotorAmpWarningsMsg

Definition

```
typedef  char  MEIMotorAmpWarningsMsg[MEIMotorAmpMsgMAX];
```

Description

MEIMotorAmpWarningsMsg defines the maximum number of amp warning messages per motor.

See Also

MPIMotorBrake

Definition

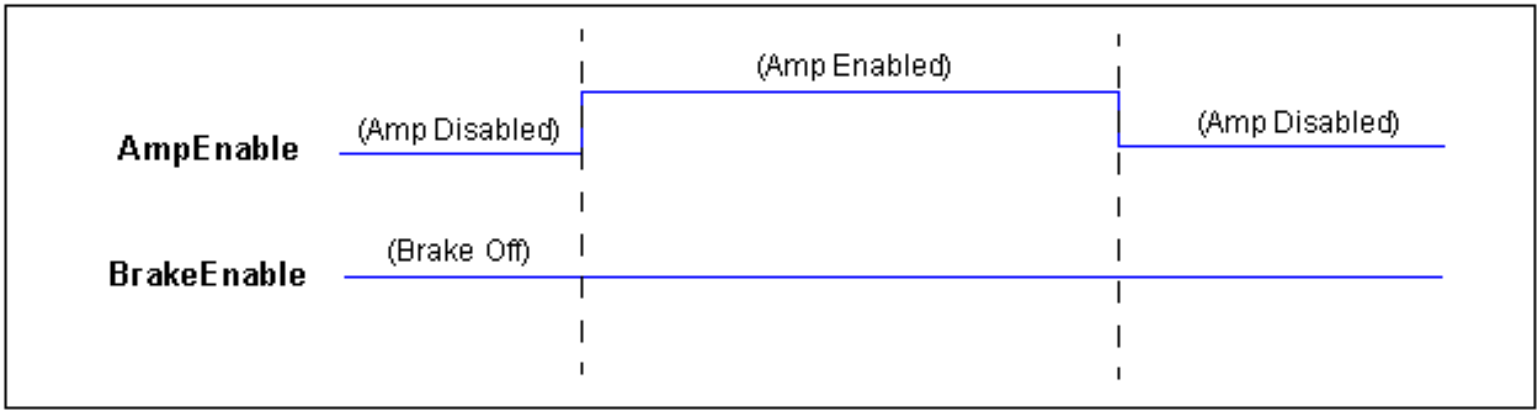
```
typedef struct MPIMotorBrake {
    MPIMotorBrakeMode   mode;
    float               applyDelay;
    float               releaseDelay;
} MPIMotorBrake;
```

Description

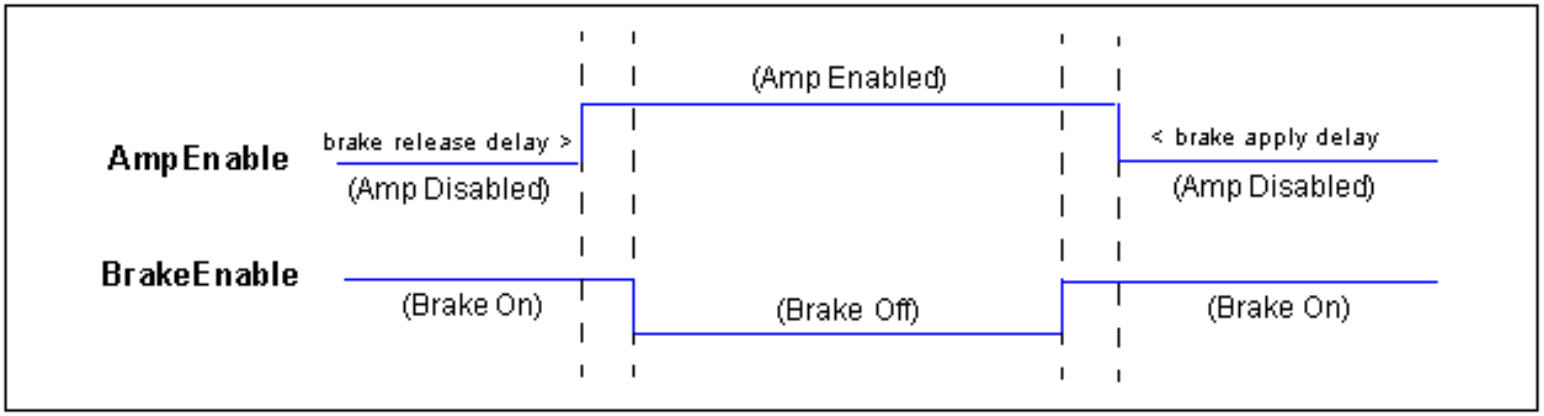
MPIMotorBrake specifies the configuration for a motor's dedicated brake logic. Each motor object has a dedicated brake output. The controller enables/disables the brake depending on the amp enable state and the brake configuration. When the amp enable is disabled, the brake is set to an active state. When the amp enable is enabled, the brake is set to an inactive state.

mode	An enumerated brake mode. See MPIMotorBrakeMode.
applyDelay	The time between when the brake is active and the amp enable is disabled. The units are in seconds.
releaseDelay	The time between when the amp enable is enabled and the brake is inactive. The units are in seconds.

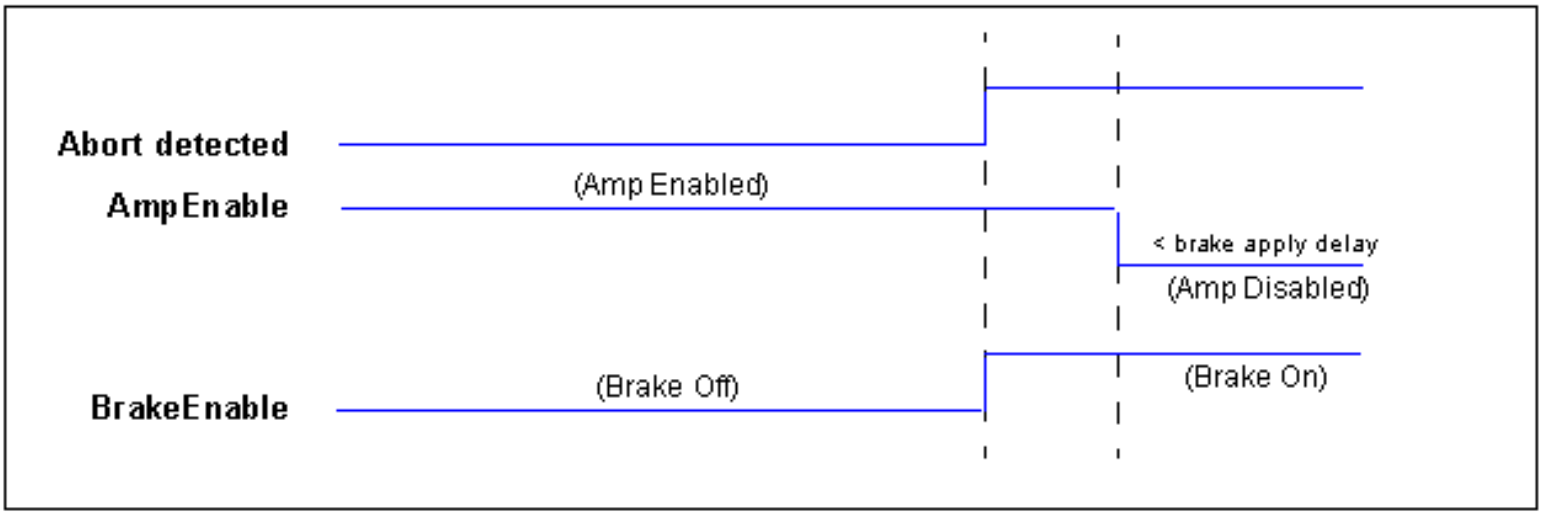
See the diagrams below for the brake logic details:



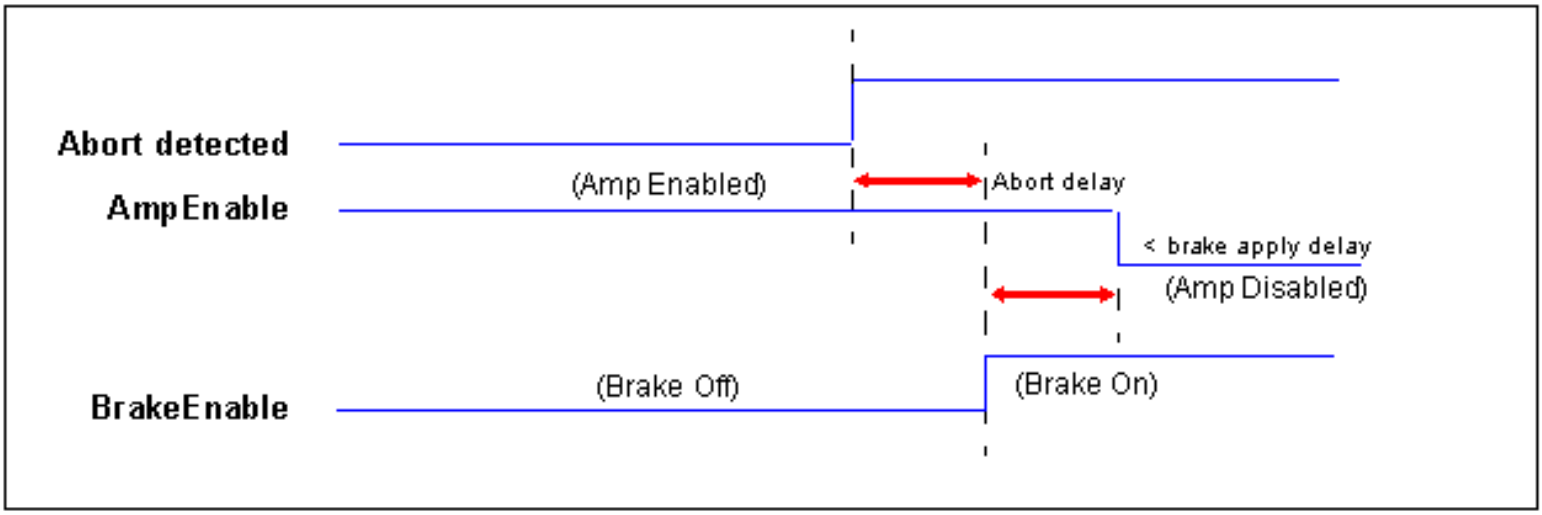
Case 1. Amp On/Off with NO Brake



Case 2. Amp On/Off with Brake



Case 3. Amp Off after ABORT Detection (in system with NO abort delay)



Case 4. Amp Off after ABORT Detection (in system WITH abort delay)

See Also

mpiMotorConfigGet | mpiMotorConfigSet | MPIMotorConfig | MPIMotorDedicatedOut

MPIMotorBrakeMode

Definition

```
typedef enum{
    MPIMotorBrakeModeNONE,
    MPIMotorBrakeModeDELAY,
} MPIMotorBrakeMode;
```

Description

MPIMotorBrakeMode is an enumeration of modes for the dedicated brake signal.

MPIMotorBrakeModeNONE	Brake feature is disabled.
MPIMotorBrakeModeDELAY	Brake is enabled/disabled with specified delays.

See Also

MPIMotorBrake | MPIMotorConfig

MPIMotorConfig / MEIMotorConfig

Definition: MPIMotorConfig

```
typedef struct MPIMotorConfig {
    MPIMotorType     type;

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

    float           abortDelay;
    float           enableDelay;
    MPIMotorBrake   brake;

    MPIObjectMap    filterMap;

} MPIMotorConfig;
```

Description

event	Structure to configure various Motor Events. See MPIMotorEventConfig description.
abortDelay	Sets time value, in seconds, to delay Abort action after Event has occured.
enableDelay	Sets time value, in seconds, to delay Enabling of the amplifier after commanded.
brake	Configures the dedicated brake logic. See MPIMotorBrake.
filterMap	Get/Set a map of Filter Objects to which the Motor is mapped. Default mapping is Filter 0 to Motor 0, Filter 1 to Motor 1, etc. See also MPIObjectMap description in Object section.

Definition: MEIMotorConfig

```
typedef struct MEIMotorConfig {
    char                    userLabel[MEIObjectLabelCharMAX+1];
                              /* +1 for NULL terminator */
    MEIMotorDemandMode      demandMode;
    MEIMotorEncoder         Encoder[MEIXmpMotorEncoders];
    MEIMotorStatusOutput    StatusOutput;

    MEIMotorIoConfig        Io[MEIMotorIoConfigIndexLAST];

    MEIMotorFaultConfig     faultConfig;

    MEIMotorStepper         Stepper;
    MEIMotorDacConfig       Dac;

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

    MEIXmpLimitData         Limit[MEIXmpLimitLAST];

    MPIAction               nodeFailureAction;
    MPIAction               userFaultAction;
                              /* see MEISqNodeConfigUserFault{}
                              structure */
    MEIMotorDisableAction   disableAction;

} MEIMotorConfig;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00.

Description

MEIMotorConfig contains configurations for the motor.

userLabel	This field consists of 16 characters and is used to label the motot object for user identification purposes. The userLabel field is NOT used by the controller.
demandMode	The demand mode determines the data type(s) transmitted from the controller to the node for servo control. The default value is automatically configured during SynqNet network initialization, based on the particular node model. The demand mode cannot be changed when the motor's **ampEnable** is enabled for safety. See MEIMotorDemandMode description.
Encoder	Structure to configure Motor Encoder type and parameters.
StatusOutput	A structure to configure a motor's digital outputs to monitor axis status bits. Requires custom firmware.
Io	An array of motor I/O configuration structures.
faultConfig	A structure to configure a motor's fault bits. Support for motor fault bits is node/drive specific.
Stepper	Structure to configure Motor Stepper parameters. See MEIMotorStepper description.
Dac	Structure that includes Command and Auxiliary DAC configuration for each motor. See MEIMotorDacConfig description.
Commutation	A structure to configure controller sinusoidal commutation. This structure is controller specific. Please see Sinusoidal Commutation for more details.
Limit	Structure used to configure custom motor limits and events. See User Limits for more information.
nodeFailureAction	Action applied to the motor when a Node failure occurs.
userFaultAction	Action applied to the motor when a User Fault occurs.
disableAction	Used to configure the controller to set the command position equal to the actual position while the motor is disabled.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorDacConfig

Definition

```
typedef struct MEIMotorDacConfig {
    MEIXmpDACPhase              Phase;
    MEIMotorDacChannelConfig    Cmd;
    MEIMotorDacChannelConfig    Aux;
} MEIMotorDacConfig;
```

Description

MEIMotorDacConfig is a structure that includes Command and Auxiliary DAC configuration for each motor.

See Also

meiMotorDacConfigGet | meiMotorDacConfigSet

MEIMotorDacChannelConfig

Definition

```
typedef struct MEIMotorDacChannelConfig {
    float               Offset; /* volts */
    float               Scale;
    MEIXmpDACInputType  InputType;
    MEIXmpGenericValue  *Input;
} MEIMotorDacChannelConfig;
```

Description

MEIMotorDacChannelConfig is a structure used to configure the DAC settings.

Offset	Set DAC Offset value. Valid values range from -10 Volts to +10 Volts.
Scale	Multiplier for the Dac channel. Default value is 1.0.
InputType	An enumerated value to define whether the Dac channel input is a float or a long. The input type is reserved for special or custom configurations.
Input	A pointer to a MEIXmpGenericValue, which is a union of a long and float value. The input pointer is reserved for special or custom configurations.

See Also

MEIMotorDacChannelStatus

Definition

```
typedef struct MEIMotorDacChannelStatus {
    float       level;  /* volts */
} MEIMotorDacChannelStatus;
```

Description

MEIMotorDacChannelConfig is a structure used to configure the DAC settings.

level	*level* reflects the DAC output value. Valid values range from -10 Volts to +10 Volts.

See Also

MEIMotorDacStatus

Definition

```
typedef struct MEIMotorDacStatus {
    MEIMotorDacChannelStatus    cmd;
    MEIMotorDacChannelStatus    aux;
} MEIMotorDacStatus;
```

Description

MEIMotorDacStatus is a structure that returns the Status for both Command and Auxiliary DACs. It is used to read the ***cmd*** and ***aux*** DAC level (in volts) from the controller.

See Also

MPIMotorDedicatedIn

Definition

```
typedef enum {
   MPIMotorDedicatedInAMP_FAULT                =  0,
   MPIMotorDedicatedInBRAKE_APPLIED            =  1,
   MPIMotorDedicatedInHOME                     =  2,
   MPIMotorDedicatedInLIMIT_HW_POS             =  3,
   MPIMotorDedicatedInLIMIT_HW_NEG             =  4,
   MPIMotorDedicatedInINDEX_PRIMARY            =  5,
   MPIMotorDedicatedInFEEDBACK_FAULT           =  6,
   MPIMotorDedicatedInCAPTURED                 =  7,
   MPIMotorDedicatedInHALL_A                   =  8,
   MPIMotorDedicatedInHALL_B                   =  9,
   MPIMotorDedicatedInHALL_C                   =  10,
   MPIMotorDedicatedInAMP_ACTIVE               =  11,
   MPIMotorDedicatedInINDEX_SECONDARY          =  12,
   MPIMotorDedicatedInWARNING                  =  13,
   MPIMotorDedicatedInDRIVE_STATUS_9           =  14,
   MPIMotorDedicatedInDRIVE_STATUS_10          =  15,
   MPIMotorDedicatedInFEEDBACK_FAULT_PRIMARY   =  16,
   MPIMotorDedicatedInFEEDBACK_FAULT_SECONDARY =  17,
} MPIMotorDedicatedIn;
```

Change History: Modified in the 03.03.00
Added in the 03.02.00 (MPIMotorDedicatedIn replaced MEIMotorDedicatedIn).

Description

MPIMotorDedicatedIn is an enumeration of bit masks for the motor's dedicated inputs. The support for dedicated inputs is node/drive specific. See the node/drive manufacturer's documentation for details.

NOTE: MPIMotorDedicatedIn replaced MEIMotorDedicatedIn in the MPI library.

MPIMotorDedicatedInAMP_FAULT	Generated by the masked motor fault bits. Active when one or more masked motor faults bits are active. See MEIMotorFaultConfig.
MPIMotorDedicatedInBRAKE_APPLIED	Mechanical brake state.
MPIMotorDedicatedInHOME	Position calibration sensor.
MPIMotorDedicatedInLIMIT_HW_POS	Hardware limit for the positive direction.
MPIMotorDedicatedInLIMIT_HW_NEG	Hardware limit for the negative direction.
MPIMotorDedicatedInINDEX_PRIMARY	Primary encoder index input signal.
MPIMotorDedicatedInFEEDBACK_FAULT	Position feedback status. TRUE when position feedback fails, FALSE when operating properly.
MPIMotorDedicatedInCAPTURED	Currently not supported.
MPIMotorDedicatedInHALL_A	Reflects the state of Hall Sensor A
MPIMotorDedicatedInHALL_B	Reflects the state of Hall Sensor B
MPIMotorDedicatedInHALL_C	Reflects the state of Hall Sensor C
MPIMotorDedicatedInAMP_ACTIVE	A bit set by the drive that indicates the amplifier's state. 1 = Amplifier is closing the current loop and the motor winding are energized. 0 = Amplifier is not closing the current loop and the motor windings are not energized. Support for this bit varies depending on the drive type.
MPIMotorDedicatedInINDEX_SECONDARY	Secondary encoder index input signal.

MPIMotorDedicatedInWARNING	Drive warning state. 1 = drive warning status bit is active and warning message is available 0 = drive warning status bit is not active. Support for this bit varies depending on the drive type.
MPIMotorDedicatedInDRIVE_STATUS_9	State of bit 9 in the SynqNet drive specific status register.
MPIMotorDedicatedInDRIVE_STATUS_10	State of bit 10 in the SynqNet drive specific status register.
MPIMotorDedicatedInFEEDBACK_FAULT_PRIMARY	Indicates that the drive/motor primary position feedback system has detected a fault.
MPIMotorDedicatedInFEEDBACK_FAULT_SECONDARY	Indicates that the drive/motor secondary position feedback system has detected a fault.

See Also

MPIMotorDedicatedOut | mpiMotorDedicatedIn

MPIMotorDedicatedOut

Definition

```
typedef enum {
    MPIMotorDedicatedOutAMP_ENABLE
          = MEIXmpMotorDedicatedFlagsMaskAMP_ENABLE,    /* bit 0 */
    MPIMotorDedicatedOutBRAKE_RELEASE
          = MEIXmpMotorDedicatedFlagsMaskBRAKE_RELEASE, /* bit 1 */
} MPIMotorDedicatedOut;
```

Change History: Added in the 03.02.00. MPIMotorDedicatedOut replaced MEIMotorDedicatedOut.

Description

MPIMotorDedicatedOut is an enumeration of bit masks for the motor's dedicated outputs. The support for dedicated outputs is node/drive specific. See the node/drive manufacturer's documentation for details.

NOTE: MPIMotorDedicatedOut replaced MEIMotorDedicatedOut in the MPI library.

MPIMotorDedicatedOutAMP_ENABLE	Enable/disable drive or amplifier. Drive is enabled when TRUE, disabled when FALSE.
MPIMotorDedicatedOutBRAKE_RELEASE	Enable/disable mechanical brake. Brake is released (motor shaft is free) when TRUE, engaged when FALSE.

See Also

MPIMotorDedicatedIn | mpiMotorDedicatedOutGet

MEIMotorDemandMode

Definition

```
typedef enum MEIMotorDemandMode {
   MEIMotorDemandModeTORQUE,
   MEIMotorDemandModeVELOCITY,
   MEIMotorDemandModeANALOG,
   MEIMotorDemandModeANALOG_DUAL_DAC,
} MEIMotorDemandMode;
```

Change History: Added in the 03.04.00

Description

MEIMotorDemandMode is an enumeration of demand modes for a motor. The demand mode determines the data type(s) transmitted from the controller to the node for servo control.

During SynqNet network initialization, the nodes are discovered and the demand mode is automatically set to a default based on the particular drive model. The number of demand fields (1, 2, or 3) per motor are enabled and connected to the controller's filter (closed-loop servo algorithm). The user can change the demand mode using mpiMotorConfigGet/Set(…).

Please consult the drive or RMB (Remote Motion Block) documentation to determine which modes are supported.

MEIMotorDemandModeTORQUE	The controller sends a 16-bit motor control value representing torque. For SynqNet drives only.
MEIMotorDemandModeVELOCITY	The controller sends a 16-bit motor control value representing velocity. For SynqNet drives only.
MEIMotorDemandModeANALOG	The controller sends a 16-bit motor control value representing torque. For RMBs only.
MEIMotorDemandModeANALOG_DUAL_DAC	The controller sends two 16-bit motor control values. Use this mode for auxiliary DACs or sinusoidal commutation. For RMBs only.

Remarks

For optimum controller performance with RMBs, set **demandMode** = MEIMotorDemandModeANALOG_DUAL_DAC ONLY for motors that actually use the auxiliary DACs.

Sample Code

To configure the SynqNet demand mode for velocity:

```
MEIMotorConfig config;
mpiMotorConfigGet(motor, NULL, &config);

config.demandMode = MEIMotorDemandModeVELOCITY;

mpiMotorConfigSet(motor, NULL, &config);
```

See Also

MEIMotorConfig | mpiMotorConfigSet | mpiMotorConfigGet

SynqNet Demand Modes

MEIMotorDisableAction

Definition

```
typedef enum MEIMotorDisableAction {
    MEIMotorDisableActionNONE,
    MEIMotorDisableActionCMD_EQ_ACT,
} MEIMotorDisableAction;
```

Description

MEIMotorDisableAction is an enumeration of controller actions to be applied when the Amp Enable is disabled. This feature can be configured by calling mpiMotorConfigSet(...) with the disableAction element in the MEIMotorConfig structure set to one of the values defined in the MEIMotorDisableAction enumeration.

The default configuration is MEIMotorDisableActionCMD_EQ_ACT. This configuration applies some safety features which eliminate motor jumps when the Amp Enable is enabled. This is the recommended configuration for all servo motor types.

The CMD_EQ_ACT action does not operate with stepper motors. If the motor type is a stepper, make sure to set the disable action to NONE. The pulse output is based on the command position, so if the controller sets the command position equal to the actual position during motion, it will cause very unusual motion profiles.

MEIMotorDisableActionNONE	No action. When the Amp Enable is disabled (or enabled), the controller continues to calculate and apply the torque demand output value.
MEIMotorDisableActionCMD_EQ_ACT	Command position equals actual position action (default). When the Amp Enable is disabled, the controller will: 1) Disable the servo loop output (except for the offset). 2) Set the command position equal to the actual position every sample. 3) Clear the integrator error. When the Amp Enable is enabled, the controller will operate the servo loop normally.

See Also

mpiMotorConfigSet | MEIMotorConfig | MEIMotorDisableAction

MPIMotorEncoder / MEIMotorEncoder

Definition: MPIMotorEncoder

```
typedef enum {
    MPIMotorEncoderPRIMARY,
    MPIMotorEncoderSECONDARY,
} MPIMotorEncoder;
```

Description

MPIMotorEncoder is an enumeration of encoder feedback inputs for a motor.

MPIMotorEncoderPRIMARY	The first encoder feedback for a motor.
MPIMotorEncoderSECONDARY	The second encoder feedback for a motor.

Definition: MEIMotorEncoder

```
typedef struct MEIMotorEncoder {
    MEIMotorEncoderType             type;
    long                            encoderPhase;
                                    /* 0 => normal, else reversed */
    long                            filterDisable;
                                    /* 0 => quad filter enabled,
                                    else not enabled */
    MEIMotorEncoderRatio            ratio;
    MEIMotorEncoderModulo           modulo;
    MEIMotorEncoderSsiConfig        ssiConfig;
} MEIMotorEncoder;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00

Description

MEIMotorEncoder is an enumeration of encoder feedback inputs for a motor.

For **standard modulo**, set modulo.value to the modulo value and set modulo.reverse to FALSE.

For **reverse modulo**, set modulo.value to the modulo value from the raw encoder and set modulo.reverse to TRUE.

type†	The type of feedback being used by the motor.
encoderPhase†	Controls the direction of the encoder counts (only applicable with quadrature encoder feedback).
filterDisable†	Enables/Disables the use of filters on the encoder signal.
ratio	Custom firmware required. Encoder feedback ratio for scaling.
modulo	Custom firmware required. Reverse modulo value for the encoder to handle rollover.
ssiConfig	A structure to configure the motor feedback for SSI (Synchronous-Serial Interface). Only use **ssiConfig** when the MEIMotorEncoderType is configured for **MEIMotorEncoderTypeSSI**.

† - If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

Error Limit and Limit Switch Errors | mpiMotorCreate | MEIMotorEncoderType

MPIMotorEncoderFault

Definition

```
typedef enum {
    MPIMotorEncoderFaultPRIMARY,
    MPIMotorEncoderFaultSECONDARY,
    MPIMotorEncoderFaultPRIMARY_OR_SECONDARY,
} MPIMotorEncoderFault;
```

Description

MPIMotorEncoderFault is an enumeration of encoder fault sources for motor encoder fault events. Each motor object supports a primary and secondary encoder. The hardware may or may not support a secondary encoder.

MPIMotorEncoderFaultPRIMARY	Sets the Motor Event (Encoder Fault) to trigger from the primary encoder.
MPIMotorEncoderFaultSECONDARY	Sets the Motor Event (Encoder Fault) to trigger from the secondary encoder.
MPIMotorEncoderFaultPRIMARY_OR_SECONDARY	Sets the Motor Event (Encoder Fault) to trigger from either the primary or secondary encoder.

See Also

MPIMotorEventConfig | MPIMotorEventTrigger

MPIMotorEncoderFaultMask

Definition

```
typedef enum {
     MPIMotorEncoderFaultMaskNONE,
     MPIMotorEncoderFaultMaskBW_DET,
     MPIMotorEncoderFaultMaskILL_DET,
     MPIMotorEncoderFaultMaskABS_ERR,
     MPIMotorEncoderFaultMaskALL
} MPIMotorEncoderFaultMask;
```

Description

MPIMotorEncoderFaultMask is an enumeration to mask bits from MPIMotorEncoderFault register.

MPIMotorEncoderFaultMaskBW_DET	Mask for Broken Wire detection
MPIMotorEncoderFaultMaskILL_DET	Mask for Illegal State detection
MPIMotorEncoderFaultMaskABS_ERR	Mask for Absolute Encoder Error

See Also

MEIMotorEncoderModulo

Definition

```
typedef struct MEIMotorEncoderModulo {
   MPI_BOOL   reverse;  /* set to TRUE if external encoder is
                           already moduloed at a non 32 bit boundary */
   long       value;
} MEIMotorEncoderModulo;
```

Change History: Added in the 03.04.00.

Description

The controller can modulo incomming encoder feedback. In cases where the you want to keep track of how many times the motor has passed a certain position (ex: how many times a motor has gone through one revolution), the modulo function can be used. If you want to keep track of how many times a motor has gone through one revolution, set the modulo ***value*** to the number of encoder counts per revolution and set ***reverse*** to FALSE. The lower 32 bits of the motor feedback will represent the position within one revolution of the motor. The upper 32 bits will represent how many times the motor has gone through one revolution.

For motors that do not have 32 bits of feedback, the reverse modulo function MUST be used. Set the modulo ***value*** equal to the encoder resolution and set ***reverse*** to TRUE. The controller will create a 64-bit encoder position in the motor object from the encoder feedback.

reverse	If the encoder provides 32 bits of feedback, set the reverse field to FALSE. In the case where an encoder does not provide 32 bits of feedback, the reverse field is used. For example, if an encoder has 24 bits of feedback, set the modulo value to 2^{24} and set the reverse field to TRUE. The controller will create a full 64-bit encoder position based on the 24-bit feedback from the motor.
value	Set to the Modulo value. Modulo is always active. The modulo value can be any number from 0 to 4294967295 (2^{32}-1). A value of 0 (default) is equivalent to 2^{32} (0x100000000) and causes a rollover at 32 bits. The 64-bit motor feedback value is created from the 32-bit position feedback from the motor. Modulo is generally used on encoders that provide 32 bits of feedback and the user needs to keep track of the number of times a certain value is exceeded. For example, a motor with 2000 counts/rev could use a modulo value of 2000 so that the number of revolutions is counted. In this case, the upper 32 bits of the motor feedback value represents the number of revolutions and the lower 32 bits of the motor feedback value represents the position (0 – 1999) of the motor within one revolution.

	In the case where an encoder does not provide 32 bits of feedback, the reverse field is used. For example, if an encoder has 24 bits of feedback, set the modulo value to 2^{24} and set the reverse field to TRUE. The controller will create a full 64-bit encoder position based on the 24-bit feedback from the motor.

See Also

MEIMotorEncoderRatio

Definition

```
typedef struct MEIMotorEncoderRatio {
    long    A;    /* denominator */
    long    B;    /* numerator */
} MEIMotorEncoderRatio;
```

Description

MEIMotorEncoderRatio is used to set the Encoder ratio. Ratio is programmed into the firmware via a denominator (A) and a numerator (B). B/A = ratio.

NOTE: Custom Firmware is required to support this data type.

A	the denominator.
B	the numerator.

See Also

MEIMotorEncoderReverseModulo

Definition

```
typedef struct MEIMotorEncoderReverseModulo {
    long    *Ptr;   /* XMP Address */
    long    Rollover;
} MEIMotorEncoderReverseModulo;
```

Description

MEIMotorEncoderReverseModulo is used to program the firmware to calculate a 32-bit encoder position based off of the data that is pointed to by ***Ptr***. Each sample, the firmware calculates a delta from the data that is pointed to by ***Ptr*** and the ***Rollover*** value. This delta is added to a 32-bit counter to create a 32-bit position.

NOTE: Custom Firmware is required to support this data type.

***Ptr**	a pointer to the XMP address.
Rollover	the value is the resolution of the incoming data pointed to by *Ptr.

See Also

MEIMotorEncoderSsiConfig

Definition

```
typedef struct MEIMotorEncoderSsiConfig {
    MEIMotorSsiInput   input;
    long               bitCount;
    long               baudRate;  /* units = hertz */
    MPI_BOOL           brokenWireEnable;
} MEIMotorEncoderSsiConfig;
```

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

Description

MEIMotorEncoderSsiConfig contains configurations for SSI (Synchronous-Serial Interface) feedback devices. Use this structure to configure the SSI when the MEIMotorEncoderType is configured for **MEIMotorEncoderTypeSSI**. Be sure to verify that the SynqNet node FPGA supports SSI feedback devices. See the Node FPGA Images: Features Table.

input	Source pin for the serial input from the SSI feedback device.
bitCount	Resolution in bits for the SSI feedback device. The valid range is 1 to 32 bits.
baudRate	Serial communication rate in bits per second. The valid range is 10000 to 500000.
brokenWireEnable	Enable (TRUE) or disable (FALSE) broken wire detection.

Sample Code

Configure the motor's primary feedback for an SSI device by using general purpose bit #0 to drive the clock output at 500 kHz. The serial input is decoded via the encoder input channel A.

```
returnValue =
    mpiMotorConfigGet(motor,
                      NULL,
                      &motorConfig);

 /* Configure the IO to output the SSI Clock */
   motorConfig.Io[MEIMotorIoConfigIndex0}.Type = MEIMotorIoTypeSSI_CLOCK0;

   /* Configure the primary feedback for SSI */
   motorConfig.Encoder[0].type = MEIMotorEncoderTypeSSI;
   motorConfig.Encoder[0].ssiConfig.baudRate = 500000; /* 500 KHZ */
   motorConfig.Encoder[0].ssiConfig.bitCount = 32;
   motorConfig.Encoder[0].ssiConfig.input = MEIMotorSsiInputENC_A;
   motorConfig.Encoder[0].ssiConfig.brokenWireEnable = TRUE;

returnValue =
   mpiMotorConfigSet(motor,
                     NULL,
                     &motorConfig);
```

See Also

MEIMotorSsiInput | MEIMotorEncoderType | mpiMotorConfigSet | mpiMotorConfigGet

Sample Application
ssiEncCfg.c

MEIMotorEncoderType

Definition

```
typedef enum MEIMotorEncoderType{
    MEIMotorEncoderTypeQUAD_AB = MEIXmpMotorEncoderConfigTypeQUAD_AB,
    MEIMotorEncoderTypeDRIVE   = MEIXmpMotorEncoderConfigTypeDRIVE,
    MEIMotorEncoderTypeSSI     = MEIXmpMotorEncoderConfigTypeSSI,
} MEIMotorEncoderType;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00

Description

MEIMotorEncoderType is the type of encoder being used for feedback. The encoder type is drive specific. Therefore, an appropriate default value should be set by the MPI.

MEIMotorEncoderTypeQUAD_AB	Quadrature encoder feedback.
MEIMotorEncoderTypeDRIVE	Drive memory interface feedback. This includes all feedback types supported by the drive.
MEIMotorEncoderTypeSSI	Synchronous Serial Interface (SSI) absolute encoder feedback.

See Also

MPIMotorEventConfig / MEIMotorEventConfig

Definition: MPIMotorEventConfig

```
typedef struct MPIMotorEventConfig {
    MPIAction            action;
    MPIMotorEventTrigger trigger;
    MPI_BOOL             direction;
    float                duration;   /* seconds */
} MPIMotorEventConfig;
```

Change History: Modified in the 03.03.00

Description

MPIMotorEventConfig is a structure used to configure Motor Events.

action	The action to be taken on the associated axis when the event is triggered (when the event status changes from FALSE to TRUE).
trigger	The trigger configuration for an event.
direction	Only used for Hardware and Software limits. Setting ***direction*** to TRUE requires the direction of motion to be in direction of the Limit to trigger the event. If ***direction*** is set to FALSE, a limit event can be generated regardless of the direction of motion.
duration	time that Limit (e.g. Home, Pos. and Neg. Limits, User Limit) must be asserted before Event is generated. Value in seconds. **NOTE**: The duration parameter for the home limit should be zero for standard configurations. A non-zero value will result in the home limit being missed.

Definition: MEIMotorEventConfig

```
typedef MEIXmpLimitData MEIMotorEventConfig;

typedef struct {
    MEIXmpLimitCondition    Condition[MEIXmpLimitConditions];
    MEIXmpStatus            Status;
    MEIXmpLogic             Logic;
    MEIXmpLimitOutput       Output;
    long                    Count;
    long                    State;
} MEIXmpLimitData;
```

Description

MEIMotorEventConfig is an enumeration of encoder feedback inputs for a motor.

condition	is a structure that configures the conditional statements evaluated to generate a Limit Event. Each limit may have up to two conditions (MEIXmpLimitConditions = 2). This structure is described in further detail on the User Limits page.
status	an enum that defines what actions the XMP will take when a user limit evaluates TRUE. Always set Status to at least MEIXmpStatusLIMIT to notify the motor object that a limit has occurred. Valid **Status** values are listed in the Values of Status table below.
logic	an enum that sets the logic applied between the two condition block outputs, Condition[0] and Condition[1]. Valid **Logic** values are listed in the Values of Logic table below.
output	is a structure that allows specific action to be taken when the Limit Event is generated. In addition to generating a Motion Action, a Limit can write to any other valid XMP Firmware register defined in the *OutputPtr with value described by AndMask and OrMask. This structure is described in further detail on the User Limits page.
count	For internal use only. The MPI method, mpiMotorEventConfigSet(...) will not write these values.
state	For internal use only. The MPI method, mpiMotorEventConfigSet(...) will not write these values.

Values of Status	Action to be taken
MEIXmpStatusLIMIT	None
MEIXmpStatusLIMIT | MEIXmpStatusPAUSE	Axes attached to the motor will be Paused
MEIXmpStatusLIMIT | MEIXmpStatusSTOP	Axes attached to the motor will be Stopped
MEIXmpStatusLIMIT | MEIXmpStatusABORT	Axes attached to the motor will be Aborted
MEIXmpStatusLIMIT | MEIXmpStatusESTOP	Axes attached to the motor will be E-Stopped

MEIXmpStatusLIMIT | MEIXmpStatusESTOP_ABORT	Axes attached to the motor will be E-Stopped and Aborted

Values of Logic	Evaluates	Motor object notified that a limit has occurred if...
MEIXmpLogicNEVER	Nothing	No event is generated
MEIXmpLogicSINGLE	Condition[0]	Condition[0] = = TRUE
MEIXmpLogicOR	Condition[0], Condition[1]	(Condition[0] || Condition[1]) = = TRUE
MEIXmpLogicAND	Condition[0], Condition[1]	(Condition[0] && Condition[1]) = = TRUE
other **MEIXmpLogic** enums	For internal use only.	

See Also

MPIMotorEventConfig and Motor Limit Configuration
Error Limit and Limit Switch Errors
User Limits

mpiMotorEventConfigGet | mpiMotorEventConfigSet | MPIEncoderFault

MPIMotorEventTrigger

Definition

```
typedef union {
    long                 polarity; /* 0 => active low, else active high */
    double               position; /* MPIEventTypeLIMIT_SW_[POS|NEG] */
    float                error;    /* MPIEventTypeLIMIT_ERROR */
    MPIMotorEncoderFault encoder;  /* MPIEventTypeENCODER_FAULT */
} MPIMotorEventTrigger;
```

Change History: Modified in the 03.04.00. Changed **position** info to a double.

Description

polarity	Configures the polarity for the motor event trigger. If polarity = 0 (FALSE), the event will trigger on an active low signal. If polarity = 1 (TRUE), the event will trigger on an active high signal.
position	Configures the positive and negative software position limits. The controller monitors the actual position and compares it to the positive and negative software position limits. If the positive limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_POS event. If the negative limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_NEG event. **MPI version 03.04.xx (and newer)** If the actual position value minus the negative limit value is greater than 2^{63} counts, then the negative limit will trip. Similarly, if the actual position value minus the positive limit value is less than -2^{63} counts, then the positive limit will trip. This is a result of a wraparound with the 64-bit signed value. When these conditions occur, the comparisons will not work correctly. **MPI version 03.03.xx (and older)** If the actual position value minus the negative limit value is greater than 2^{31} counts, then the negative limit will trip. Similarly, if the actual position value minus the positive limit value is less than -2^{31} counts, then the positive limit will trip. This is a result of a wraparound with the 32-bit signed value. When these conditions occur, the comparisons will not work correctly.
error	Configures the position error limit. The controller calculates the position error each sample period: (command position - actual position) = position error. If the position error exceeds the range of the specified error limit, the controller will generate a MPIEventTypeLIMIT_ERROR.

encoder	Configures the controller to monitor the primary or secondary for faults. The encoder should be set to one of the values defined by the MPIMotorEncoderFault {...} enumeration. When configured, if the primary, secondary, or either encoder generates a fault, the controller will generate the MPIEventTypeENCODER_FAULT.

See Also

MPIMotorEventConfig | MPIMotorEncoderFault | MPIEventType | MEIEventType | Error Limit and Limit Switch Errors

MEIMotorFaultBit

Definition

```
typedef enum MEIMotorFaultBit {
    MEIMotorFaultBitINVALID,
    MEIMotorFaultBitAMP_FAULT,
    MEIMotorFaultBitDRIVE_FAULT,
    MEIMotorFaultBitWATCHDOG_FAULT,
    MEIMotorFaultBitCHECKSUM_ERROR,
    MEIMotorFaultBitFEEDBACK_FAULT,
    MEIMotorFaultBitAMP_NOT_POWERED,
    MEIMotorFaultBitDRIVE_NOT_READY,
    MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY,
} MEIMotorFaultBit;
```

Description

MEIMotorFaultBit is an enumeration of motor fault bits for the SynqNet node drive interface. The support for motor fault bits is node/drive specific. Please see the node\drive manufacturer's documentation for details.

MEIMotorFaultBitAMP_FAULT

External amp fault input pin. Indicates a fault condition in the amplifier.

Available on remote motion blocks.

MEIMotorFaultBitDRIVE_FAULT

Bit in the drive interface status register. Indicates a fault condition in the drive.

NOT available on remote motion blocks.

MEIMotorFaultBitWATCHDOG_FAULT

Bit in the drive interface status register. Indicates a drive failure due to a watchdog timeout. The node alternately sets/clears the watchdog each sample, the drive's processor then clears/sets the watchdog. If the drive's processor does not respond, the watchdog fault bit is set.

NOT available on remote motion blocks.

MEIMotorFaultBitCHECKSUM_ERROR

Bit in the drive interface status register. Indicates the data transfer (via serial or parallel memory interface) between the failed SynqNet node FPGA and drive with a checksum error.

NOT available on remote motion blocks.

MEIMotorFaultBitFEEDBACK_FAULT

Bit in the drive interface status register. Active when one or more of the feedback status bits is triggered. Indicates that the drive/motor position feedback system has failed.

Available on remote motion blocks.

MEIMotorFaultBitAMP_NOT_POWERED

Bit in the drive interface status register. The SynqNet drive amplifier power stage does not have sufficient voltage. The motor windings cannot be energized properly until this bit is clear.

NOT available on remote motion blocks.

MEIMotorFaultBitDRIVE_NOT_READY

Bit in the drive interface status register. The SynqNet drive is not ready to receive commands or servo motors. This fault indicates the drive has not completed its processor initialization or there is some other serious drive processor problem.

NOT available on remote motion blocks.

MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY

Bit in the drive interface status register. Active when one or more of the secondary feedback status bits is triggered. Indicates that the drive/motor auxilury position feedback system has failed.

Available on remote motion blocks.

See Also

MEIMotorFaultMask | MEIMotorFaultConfig | meiMotorStatus

MEIMotorFaultConfig

Definition

```
typedef struct MEIMotorFaultConfig {
    MEIMotorFaultMask   faultMask; /* sets the
                                      MEIMotorDedicatedInAMP_FAULT state */
} MEIMotorFaultConfig;
```

Description

MEIMotorFaultConfig specifies the motor fault bit configuration.

faultMask†	A mask of motor fault bits. The masked motor fault bits will be monitored by the controller's motor object. If any masked motor fault bit is active (TRUE), the motor's dedicated amp fault input (MEIMotorDedicatedInAMP_FAULT) is set active (TRUE). The support for motor fault bits is node/drive specific. During SynqNet network initialization, the SqNodeLib automatically configures the faultMask based on the node type. See the node/drive manufacturer's documentation for details.

† - If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

MEIMotorFaultBit | MEIMotorFaultMask | meiMotorStatus

MEIMotorFaultMask

Definition

```
typedef enum MEIMotorFaultMask {
      MEIMotorFaultMaskAMP             = (1<<
MEIMotorFaultBitAMP_FAULT),
      MEIMotorFaultMaskDRIVE           = (1<< MEIMotorFaultBitDRIVE_FAULT),
      MEIMotorFaultMaskWATCHDOG        = (1<< MEIMotorFaultBitWATCHDOG_FAULT),
      MEIMotorFaultMaskCHECKSUM        = (1<< MEIMotorFaultBitCHECKSUM_ERROR),
      MEIMotorFaultMaskFEEDBACK        = (1<< MEIMotorFaultBitFEEDBACK_FAULT),
      MEIMotorFaultMaskAMP_NOT_POWERED = (1<< MEIMotorFaultBitAMP_NOT_POWERED),
      MEIMotorFaultMaskDRIVE_NOT_READY = (1<< MEIMotorFaultBitDRIVE_NOT_READY),
      MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY = (1<<
MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY),
  } MEIMotorFaultMask;
```

Description

MEIMotorFaultMask is an enumeration of motor fault bit masks for the SynqNet node drive interface. The support for motor fault bits is node/drive specific. Please see the node\drive manufacturer's documentation for details.

MEIMotorFaultMaskAMP	External amp fault input pin. Indicates a fault condition in the amplifier. Available on remote motion blocks.
MEIMotorFaultMaskDRIVE	Bit in the drive interface status register. Indicates a fault condition in the drive. NOT available on remote motion blocks.
MEIMotorFaultMaskWATCHDOG	Bit in the drive interface status register. Indicates a drive failure due to a watchdog timeout. The node alternately sets/clears the watchdog each sample, the drive's processor then clears/sets the watchdog. If the drive's processor does not respond, the watchdog fault bit is set. NOT available on remote motion blocks.
MEIMotorFaultMaskCHECKSUM	Bit in the drive interface status register. Indicates the data transfer (via serial or parallel memory interface) between the failed SynqNet node FPGA and drive with a checksum error. NOT available on remote motion blocks.

MEIMotorFaultMaskFEEDBACK	Bit in the drive interface status register. Active when one or more of the feedback status bits is triggered. Indicates that the drive/motor position feedback system has failed. Available on remote motion blocks.
MEIMotorFaultMaskAMP_NOT_POWERED	Bit in the drive interface status register. The SynqNet drive amplifier power stage does not have sufficient voltage. The motor windings cannot be energized properly until this bit is clear. NOT available on remote motion blocks.
MEIMotorFaultMaskDRIVE_NOT_READY	Bit in the drive interface status register. The SynqNet drive is not ready to receive commands or servo motors. This fault indicates the drive has not completed its processor initialization or there is some other serious drive processor problem. NOT available on remote motion blocks.
MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY	Bit in the drive interface status register. Active when one or more of the secondary feedback status bits is triggered. Indicates that the drive/motor auxilury position feedback system has failed. Available on remote motion blocks.

See Also

MEIMotorFaultBit | MEIMotorFaultConfig | meiMotorStatus

MPIMotorFeedback

Definition

```
typedef   double   MPIMotorFeedback[MPIMotorEncoderLAST];
```

Description

MPIMotorFeedback is an array of doubles used to store motor feedback (both primary and secondary).

See Also

MPIMotorEncoder | mpiMotorFeedback

MPIMotorGeneralIo

Definition

```
typedef enum MPIMotorGeneralIo {
    MPIMotorGeneralIo0,
    MPIMotorGeneralIo1,
    MPIMotorGeneralIo2,
    MPIMotorGeneralIo3,
    MPIMotorGeneralIo4,
    MPIMotorGeneralIo5,
    MPIMotorGeneralIo6,
    MPIMotorGeneralIo7,
    MPIMotorGeneralIo8,
    MPIMotorGeneralIo9,
    MPIMotorGeneralIo10,
    MPIMotorGeneralIo11,
    MPIMotorGeneralIo12,
    MPIMotorGeneralIo13,
    MPIMotorGeneralIo14,
    MPIMotorGeneralIo15,
    MPIMotorGeneralIo16,
    MPIMotorGeneralIo17,
    MPIMotorGeneralIo18,
    MPIMotorGeneralIo19,
    MPIMotorGeneralIo20,
    MPIMotorGeneralIo21,
    MPIMotorGeneralIo22,
    MPIMotorGeneralIo23,
    MPIMotorGeneralIo24,
    MPIMotorGeneralIo25,
    MPIMotorGeneralIo26,
    MPIMotorGeneralIo27,
    MPIMotorGeneralIo28,
    MPIMotorGeneralIo29,
    MPIMotorGeneralIo30,
    MPIMotorGeneralIo31,
} MPIMotorGeneralIo;
```

Change History: Modified in the 03.04.00. Added in the 03.02.00.
MPIMotorGeneralIo replaced MEIMotorIoMask.

Description

MPIMotorGeneralIo enumeration gives labels for each of the general purpose outputs that a motor can support.

NOTE: MPIMotorGeneralIo replaced MEIMotorIoMask in the MPI library.

MPIMotorGeneralIo0	General I/O bit 0.
MPIMotorGeneralIo1	General I/O bit 1.
MPIMotorGeneralIo2	General I/O bit 2.
MPIMotorGeneralIo3	General I/O bit 3.
MPIMotorGeneralIo4	General I/O bit 4.
MPIMotorGeneralIo5	General I/O bit 5.
MPIMotorGeneralIo6	General I/O bit 6.
MPIMotorGeneralIo7	General I/O bit 7.
MPIMotorGeneralIo8	General I/O bit 8.
MPIMotorGeneralIo9	General I/O bit 9.
MPIMotorGeneralIo10	General I/O bit 10.
MPIMotorGeneralIo11	General I/O bit 11.
MPIMotorGeneralIo12	General I/O bit 12.
MPIMotorGeneralIo13	General I/O bit 13.
MPIMotorGeneralIo14	General I/O bit 14.
MPIMotorGeneralIo15	General I/O bit 15.
MPIMotorGeneralIo16	General I/O bit 16.
MPIMotorGeneralIo17	General I/O bit 17.
MPIMotorGeneralIo18	General I/O bit 18.
MPIMotorGeneralIo19	General I/O bit 19.
MPIMotorGeneralIo20	General I/O bit 20.
MPIMotorGeneralIo21	General I/O bit 21.
MPIMotorGeneralIo22	General I/O bit 22.
MPIMotorGeneralIo23	General I/O bit 23.
MPIMotorGeneralIo24	General I/O bit 24.
MPIMotorGeneralIo25	General I/O bit 25.

MPIMotorGeneralIo26	General I/O bit 26.
MPIMotorGeneralIo27	General I/O bit 27.
MPIMotorGeneralIo28	General I/O bit 28.
MPIMotorGeneralIo29	General I/O bit 29.
MPIMotorGeneralIo30	General I/O bit 30.
MPIMotorGeneralIo31	General I/O bit 31.

See Also

Please see General Purpose I/O

MEIMotorInfo

Definition

```
typedef struct MEIMotorInfo {
    MEIMotorInfoNodeType   nodeType;
    struct {
        long    networkNumber;
        long    nodeNumber;
        long    driveIndex;
    } sqNode;

    long    captureCount;
    long    encoderCount;
    long    probeCount;

    MEIMotorInfoDedicatedIn    dedicatedIn[MPIMotorDedicatedInLAST];
    MEIMotorInfoDedicatedOut   dedicatedOut[MPIMotorDedicatedOutLAST];
    MEIMotorInfoGeneralIo      generalIo[MPIMotorGeneralIoLAST];
} MEIMotorInfo;
```

Change History: Modified in the 03.02.00

Description

MEIMotorInfo structure contains static data determined during network initialization. It identifies the network, node, and drive interface associated with the motor object.

nodeType	Identifies the type of node. See MEIMotorInfoNodeType.
networkNumber	An index to the SynqNet network (0, 1, 2 etc.).
nodeNumber	An index to the node (0, 1, 2, etc.).
driveIndex	An index to the drive interface (0, 1, 2, etc.).
captureCount	Counts the number of captures for the motor.
encoderCount	Counts the number of encoders for the motor.
probeCount	Counts the number of hardware Probe engines for the motor.
dedicatedIn	Details about the dedicated inputs.
dedicatedOut	Details about the dedicated outputs.

generalIo	Details about the general purpose I/O.

See Also

meiMotorInfo

MEIMotorInfoDedicatedIn

Definition

```
typedef struct MEIMotorInfoDedicatedIn {
   MPI_BOOL    supported;
} MEIMotorInfoDedicatedIn;
```

Change History: Changed in the 03.03.00. Added in the 03.02.00.

Description

supported	0 = This dedicated input is NOT supported by the hardware. 1 = This dedicated input is supported by the hardware.

See Also

MEIMotorInfoDedicatedOut

MEIMotorInfoDedicatedOut

Definition

```
typedef struct MEIMotorInfoDedicatedOut {
  MPI_BOOL   supported;
} MEIMotorInfoDedicatedOut;
```

Change History: Modified in the 03.03.00. Added in the 03.02.00

Description

supported	0 = This dedicated output is NOT supported by the hardware. 1 = This dedicated output is supported by the hardware.

See Also

MEIMotorInfoDedicatedIn

MEIMotorInfoGeneralIo

Definition

```
typedef struct MEIMotorInfoGeneralIo {
   MPI_BOOL             supported;
   char                 *name;
   MEIMotorIoTypeMask   validTypes;
} MEIMotorInfoGeneralIo;
```

Change History: Modified in the 03.03.00. Added in the 03.02.00.

Description

supported	0 = This dedicated output is NOT supported by the hardware. 1 = This dedicated output is supported by the hardware.
***name**	The name the node uses for this I/O pin
validTypes	This is a bit field, each bit indicates a valid source/input that the node supports.

See Also

MEIMotorInfoNodeType

Definition

```
typedef enum MEIMotorInfoNodeType {
    MEIMotorInfoNodeTypeNONE,
    MEIMotorInfoNodeTypeSQNODE,
} MEIMotorInfoNodeType;
```

Description

MEIMotorInfoNodeType is an enumeration of node types. It specifies the node type associated with the motor.

MEIMotorInfoNodeTypeNONE	Not a network node. The node is local to the controller.
MEIMotorInfoNodeTypeSQNODE	A SynqNet node type.

See Also

MEIMotorInfo

MEIMotorIoConfig

Definition

```
typedef struct MEIMotorIoConfig {
    MEIMotorIoType     Type;
} MEIMotorIoConfig;
```

Description

MEIMotorIoConfig specifies the configuration for the motor's digital I/O.

Type	The I/O bit function. See MEIMotorIoType. If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorIoConfigIndex

Definition

```
typedef enum MEIMotorIoConfigIndex {
    MEIMotorIoConfigIndex0,
    MEIMotorIoConfigIndex1,
    MEIMotorIoConfigIndex2,
    MEIMotorIoConfigIndex3,
    MEIMotorIoConfigIndex4,
    MEIMotorIoConfigIndex5,
    MEIMotorIoConfigIndex6,
    MEIMotorIoConfigIndex7,
    MEIMotorIoConfigIndex8,
    MEIMotorIoConfigIndex9,
    MEIMotorIoConfigIndex10,
    MEIMotorIoConfigIndex11,
    MEIMotorIoConfigIndex12,
    MEIMotorIoConfigIndex13,
    MEIMotorIoConfigIndex14,
    MEIMotorIoConfigIndex15,
    MEIMotorIoConfigIndex16,
    MEIMotorIoConfigIndex17,
    MEIMotorIoConfigIndex18,
    MEIMotorIoConfigIndex19,
    MEIMotorIoConfigIndex20,
    MEIMotorIoConfigIndex21,
    MEIMotorIoConfigIndex22,
    MEIMotorIoConfigIndex23,
    MEIMotorIoConfigIndex24,
    MEIMotorIoConfigIndex25,
    MEIMotorIoConfigIndex26,
    MEIMotorIoConfigIndex27,
    MEIMotorIoConfigIndex28,
    MEIMotorIoConfigIndex29,
    MEIMotorIoConfigIndex30,
    MEIMotorIoConfigIndex31,

} MEIMotorIoConfigIndex;
```

Change History: Modified in the 03.04.00.

Description

MEIMotorIoConfigIndex is an enumeration of configurable motor I/O, indexed by a number.

MEIMotorIoConfigIndex0	motor I/O number 0
MEIMotorIoConfigIndex1	motor I/O number 1
MEIMotorIoConfigIndex2	motor I/O number 2
MEIMotorIoConfigIndex3	motor I/O number 3
MEIMotorIoConfigIndex4	motor I/O number 4
MEIMotorIoConfigIndex5	motor I/O number 5
MEIMotorIoConfigIndex6	motor I/O number 6
MEIMotorIoConfigIndex7	motor I/O number 7
MEIMotorIoConfigIndex8	motor I/O number 8
MEIMotorIoConfigIndex9	motor I/O number 9
MEIMotorIoConfigIndex10	motor I/O number 10
MEIMotorIoConfigIndex11	motor I/O number 11
MEIMotorIoConfigIndex12	motor I/O number 12
MEIMotorIoConfigIndex13	motor I/O number 13
MEIMotorIoConfigIndex14	motor I/O number 14
MEIMotorIoConfigIndex15	motor I/O number 15
MEIMotorIoConfigIndex16	motor I/O number 16
MEIMotorIoConfigIndex17	motor I/O number 17
MEIMotorIoConfigIndex18	motor I/O number 18
MEIMotorIoConfigIndex19	motor I/O number 19
MEIMotorIoConfigIndex20	motor I/O number 20
MEIMotorIoConfigIndex21	motor I/O number 21
MEIMotorIoConfigIndex22	motor I/O number 22
MEIMotorIoConfigIndex23	motor I/O number 23
MEIMotorIoConfigIndex24	motor I/O number 24
MEIMotorIoConfigIndex25	motor I/O number 25
MEIMotorIoConfigIndex26	motor I/O number 26

MEIMotorIoConfigIndex27	motor I/O number 27
MEIMotorIoConfigIndex28	motor I/O number 28
MEIMotorIoConfigIndex29	motor I/O number 29
MEIMotorIoConfigIndex30	motor I/O number 30
MEIMotorIoConfigIndex31	motor I/O number 31

See Also

MEIMotorIoConfig

MEIMotorIoType

Definition

```
typedef enum MEIMotorIoType {
   MEIMotorIoTypeOUTPUT,

   MEIMotorIoTypePULSE_A,
   MEIMotorIoTypePULSE_B,
   MEIMotorIoTypeCOMPARE_0,
   MEIMotorIoTypeCOMPARE_1,
   MEIMotorIoTypeSOURCE5
   MEIMotorIoTypeBRAKE,

   MEIMotorIoTypeSSI_CLOCK0,
   MEIMotorIoTypeSSI_CLOCK1,

   MEIMotorIoTypeSOURCE9,
   MEIMotorIoTypeSOURCE10,
   MEIMotorIoTypeSOURCE11,
   MEIMotorIoTypeSOURCE12,
   MEIMotorIoTypeSOURCE13,
   MEIMotorIoTypeSOURCE14,
   MEIMotorIoTypeSOURCE15,

   MEIMotorIoTypeINPUT,

   MEIMotorIoTypeSOURCE1 = MEIMotorIoTypePULSE_A,
   MEIMotorIoTypeSOURCE2 = MEIMotorIoTypePULSE_B,
   MEIMotorIoTypeSOURCE3 = MEIMotorIoTypeCOMPARE_0,
   MEIMotorIoTypeSOURCE4 = MEIMotorIoTypeCOMPARE_1,

   MEIMotorIoTypeSOURCE6 = MEIMotorIoTypeBRAKE,
   MEIMotorIoTypeSOURCE7 = MEIMotorIoTypeSSI_CLOCK0,
   MEIMotorIoTypeSOURCE8 = MEIMotorIoTypeSSI_CLOCK1,
} MEIMotorIoType;
```

Change History: Modified in the 03.04.00. Modified in the 03.02.00.

Description

MEIMotorIoType is an enumeration of motor I/O functions. A SynqNet node's motor I/O may support one or more features depending on the node hardware and FPGA. MotorIoType contains the generic

enumerations for all nodes. For the node specific enumerations, please see the individual SqNode modules for details.

MEIMotorIoTypeOUTPUT	discrete digital output
MEIMotorIoTypeSOURCE1	motor I/O source number 1 for node-specific feature.
MEIMotorIoTypeSOURCE2	motor I/O source number 2 for node-specific feature.
MEIMotorIoTypeSOURCE3	motor I/O source number 3 for node-specific feature.
MEIMotorIoTypeSOURCE4	motor I/O source number 4 for node-specific feature.
MEIMotorIoTypeSOURCE5	motor I/O source number 5 for node-specific feature.
MEIMotorIoTypeSOURCE6	motor I/O source number 6 for node-specific feature.
MEIMotorIoTypeSOURCE7	motor I/O source number 7 for node-specific feature.
MEIMotorIoTypeSOURCE8	motor I/O source number 8 for node-specific feature.
MEIMotorIoTypeSOURCE9	motor I/O source number 9 for node-specific feature.
MEIMotorIoTypeSOURCE10	motor I/O source number 10 for node-specific feature.
MEIMotorIoTypeSOURCE11	motor I/O source number 11 for node-specific feature.
MEIMotorIoTypeSOURCE12	motor I/O source number 12 for node-specific feature.
MEIMotorIoTypeSOURCE13	motor I/O source number 13 for node-specific feature.
MEIMotorIoTypeSOURCE14	motor I/O source number 14 for node-specific feature.
MEIMotorIoTypeSOURCE15	motor I/O source number 15 for node-specific feature.
MEIMotorIoTypeINPUT	discrete digital input

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorIoTypeMask

Definition

```
typedef enum MEIMotorIoTypeMask {
  MEIMotorIoTypeMaskOUTPUT   = (1<<MEIMotorIoTypeOUTPUT),
  MEIMotorIoTypeMaskSOURCE1  = (1<<MEIMotorIoTypeSOURCE1),
  MEIMotorIoTypeMaskSOURCE2  = (1<<MEIMotorIoTypeSOURCE2),
  MEIMotorIoTypeMaskSOURCE3  = (1<<MEIMotorIoTypeSOURCE3),
  MEIMotorIoTypeMaskSOURCE4  = (1<<MEIMotorIoTypeSOURCE4),
  MEIMotorIoTypeMaskSOURCE5  = (1<<MEIMotorIoTypeSOURCE5),
  MEIMotorIoTypeMaskSOURCE6  = (1<<MEIMotorIoTypeSOURCE6),
  MEIMotorIoTypeMaskSOURCE7  = (1<<MEIMotorIoTypeSOURCE7),
  MEIMotorIoTypeMaskSOURCE8  = (1<<MEIMotorIoTypeSOURCE8),
  MEIMotorIoTypeMaskSOURCE9  = (1<<MEIMotorIoTypeSOURCE9),
  MEIMotorIoTypeMaskSOURCE10 = (1<<MEIMotorIoTypeSOURCE10),
  MEIMotorIoTypeMaskSOURCE11 = (1<<MEIMotorIoTypeSOURCE11),
  MEIMotorIoTypeMaskSOURCE12 = (1<<MEIMotorIoTypeSOURCE12),
  MEIMotorIoTypeMaskSOURCE13 = (1<<MEIMotorIoTypeSOURCE13),
  MEIMotorIoTypeMaskSOURCE14 = (1<<MEIMotorIoTypeSOURCE14),
  MEIMotorIoTypeMaskSOURCE15 = (1<<MEIMotorIoTypeSOURCE15),

  MEIMotorIoTypeMaskINPUT = (1<<MEIMotorIoTypeINPUT),

  MEIMotorIoTypeMaskPULSE_A = (1<<MEIMotorIoTypePULSE_A),
  MEIMotorIoTypeMaskPULSE_B = (1<<MEIMotorIoTypePULSE_B),

  MEIMotorIoTypeMaskCOMPARE_0 = (1<<MEIMotorIoTypeCOMPARE_0),
  MEIMotorIoTypeMaskCOMPARE_1 = (1<<MEIMotorIoTypeCOMPARE_1),

  MEIMotorIoTypeMaskBRAKE = (1<<MEIMotorIoTypeBRAKE),

  MEIMotorIoTypeMaskSSI_CLOCK0 = (1<<MEIMotorIoTypeSSI_CLOCK0),
  MEIMotorIoTypeMaskSSI_CLOCK1 = (1<<MEIMotorIoTypeSSI_CLOCK1),
} MEIMotorIoTypeMask;
```

Change History: Added in the 03.02.00

Description

MEIMotorIoTypeMask is an enumeration to mask bits from the MEIMotorInfo structure.

MEIMotorIoTypeMaskOUTPUT	Supports the output configuration.
MEIMotorIoTypeMaskPULSE_A	Supports the Pulse A configuration.
MEIMotorIoTypeMaskPULSE_B	Supports the Pulse B configuration.
MEIMotorIoTypeMaskCOMPARE_0	Currently not supported.
MEIMotorIoTypeMaskCOMPARE_1	Currently not supported.
MEIMotorIoTypeMaskBRAKE	See MPIMotorBrake
MEIMotorIoTypeMaskSSI_CLOCK0	Supports the SSI_CLOCK0 configuration.
MEIMotorIoTypeMaskSSI_CLOCK1	Supports the SSI_CLOCK1 configuration.
MEIMotorIoTypeMaskINPUT	Supports the input configuration.

See Also

MPIMotorBrake | MEIMotorInfo | mpiMotorConfigGet | mpiMotorConfigSet | MEIMotorIoType

MPIMotorMessage / MEIMotorMessage

Definition: MPIMotorMessage

```
typedef enum {
     MPIMotorMessageMOTOR_INVALID,
     MPIMotorMessageTYPE_INVALID,
     MPIMotorMessageSTEPPER_NA,
} MPIMotorMessage;
```

Required Header: stdmpi.h
Change History: Modified in the 03.04.00: Addition of STEPPER_NA. Modified in the 03.02.00.

Description

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

MPIMotorMessageMOTOR_INVALID

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

MPIMotorMessageTYPE_INVALID

The motor type is not valid. This message code is returned by mpiMotorConfigGet / Set(...) if the motor type is not a value defined in the enumeration MPIMotorType. To avoid this error, use one of the defined motor types in MPIMotorType.

MPIMotorMessageSTEPPER_NA

The hardware does not support the Stepper motor type. This message code is returned by mpiMotorConfigSet(...) if the motor type is configured for MPIMotorTypeSTEPPER when the node hardware does not support steppers. To correct the problem, do not select the stepper motor type.

Definition: MEIMotorMessage

```
typedef enum {
    MEIMotorMessageMOTOR_NOT_ENABLED,
    MEIMotorMessageSECONDARY_ENCODER_NA,
    MEIMotorMessageHARDWARE_NOT_FOUND,
    MEIMotorMessageSTEPPER_INVALID,
    MEIMotorMessageDISABLE_ACTION_INVALID,
    MEIMotorMessagePULSE_WIDTH_INVALID,
    MEIMotorMessageFEEDBACK_REVERSAL_NA,
    MEIMotorMessageFILTER_DISABLE_NA,
    MEIMotorMessagePHASE_FINDING_FAILED,
    MEIMotorMessageDEMAND_MODE_UNSUPPORTED,
    MEIMotorMessageDEMAND_MODE_NOT_SET,
} MEIMotorMessage;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00.

Description

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

MEIMotorMessageMOTOR_NOT_ENABLED

The motor number is not active in the controller. This message code is returned by mpiMotorEventConfigGet(…) if the specified motor is not enabled in the controller. To correct the problem, use mpiControlConfigSet(…) to enable the motor object, by setting the motorCount to greater than the motor number. For example, to enable motor 0 to 3, set motorCount to 4.

MEIMotorMessageSECONDARY_ENCODER_NA

The motor's secondary encoder is not available. This message code is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the encoder fault trigger is configured for a secondary encoder when the hardware does not support a secondary encoder. To correct the problem, do not select the secondary encoder when configuring the encoder fault conditions.

MEIMotorMessageHARDWARE_NOT_FOUND

The motor object's hardware resource is not available. This message code is returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the node hardware for the motor is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor objects. An application should not configure a motor object if there is no mapped hardware to receive the service commands. To correct this problem, verify that all expected nodes were found. Use meiSqNodeInfo(…) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEIMotorMessageSTEPPER_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessageDISABLE_ACTION_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessagePULSE_WIDTH_INVALID

The motor stepper pulse width is not valid. This message code is returned by mpiMotorConfigSet(...) if the pulseWidth is out of range. To correct the problem, specify the the pulse width value between 100 nanoseconds and 1 millisecond.

MEIMotorMessageFEEDBACK_REVERSAL_NA

The feedback reversal is not applicable for the feedback type specified. This message code is returned by mpiMotorConfigSet(...) if the feedback type is not MEIMotorEncoderTypeQUAD_AB and the encoderPhase is set to TRUE. To correct the problem, set encoderPhase to FALSE. Some drives may support feedback reversal via drive parameters. Please consult the drive manufacturer's documentation for details.

MEIMotorMessageFILTER_DISABLE_NA

The feedback filter disable is not applicable for the feedback type specified. This message code is returned by mpiMotorConfigSet(...) if the feedback type is not MEIMotorEncoderTypeQUAD_AB and the filterDisable is set to TRUE. To correct the problem, set filterDisable to FALSE.

MEIMotorMessagePHASE_FINDING_FAILED

The drive failed to complete the phase finding procedure. This message code is returned by meiMotorPhaseFindStart(…) if the drive failed to successfully initailized commutation. To determine the cause of the failure, check the fault and warning codes from the drive with meiMotorAmpFault(…) and meiMotorAmpWarning(…).

MEIMotorMessageDEMAND_MODE_UNSUPPORTED

The motor does not support the specified demand mode. This message is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the demand mode is configured for a mode that the node/drive does not support. To correct this problem, use the default demand mode or specify a different demand mode.

MEIMotorMessageDEMAND_MODE_NOT_SET

The demand mode cannot be changed while the amplifier is enabled. This message is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the demand mode is changed while the motor's amplifier is enabled. To avoid this error message, disable the motor's amp enable with mpiMotorAmpEnableSet(…) before changing the demand mode.

See Also

mpiMotorCreate | MEIMotorEncoderType

MEIMotorPhaseFindDriveMsg

Definition

```
typedef struct MEIMotorPhaseFindDriveMsg {
/* drive specific code and message reporting information
   on the success or failure, see drive module for details */
   long   code;
   char*  message;
} MEIMotorPhaseFindDriveMsg;
```

Change History: Added in the 03.03.00

Description

MEIMotorPhaseFindDriveMsg drive specific information which reflects the quality or failure status of the phase finding routine.

NOTE: Not all drives will support this field.

code	A value retrieved from the drive which reports the status or quality of the Phase Finding Procedure. This value is drive manufacturer specific and is used for informational purposes only and is not evaluated by the controller.
message	A string provided by the drive manufacturer that describes what “code” represents.

See Also

MEIMotorPhaseFindState | MEIMotorPhaseFindStatus | meiMotorPhaseFindStart | meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

MEIMotorPhaseFindState

Definition

```
typedef enum MEIMotorPhaseFindState  {
   MEIMotorPhaseFindStateIN_PROGRESS,
   MEIMotorPhaseFindStateSUCCESS,
   MEIMotorPhaseFindStateFAILURE,
}MEIMotorPhaseFindState;
```

Change History: Added in the 03.03.00

Description

MEIMotorPhaseFindState is used to report the state of the phase finding procedure.

MEIMotorPhaseFindStateIN_PROGRESS	The drive is still in the process of phase finding.
MEIMotorPhaseFindStateSUCCESS	The drive has successfully completed phase finding.
MEIMotorPhaseFindStateFAILURE	The drive has finished phase finding but the process was a failure.

See Also

MEIMotorPhaseFindStatus | MEIMotorPhaseFindDriveMessage | meiMotorPhaseFindStart | meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

MEIMotorPhaseFindStatus

Definition

```
typedef struct  MEIMotorPhaseFindStatus {
   MEIMotorPhaseFindState       state;
   MEIMotorPhaseFindDriveMsg    msg;
}MEIMotorPhaseFindStatus;
```

Change History: Added in the 03.03.00

Description

MEIMotorPhaseFindStatus is a structure containing information which reflects the status of the drive Phase Finding Procedure.

state	The current state of the phase finding process: in progress, failed, success.
msg	A drive-specific code and string that describes the quality of the phase finding after the process has been completed. Or it may also include status information that describes why the phase finding failed. **NOTE**: The support for this variable is drive dependant and in some cases may not be available. Be sure to check with the drive manufacturer for support.

See Also

MEIMotorPhaseFindState | MEIMotorPhaseFindDriveMessage | meiMotorPhaseFindStart | meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

MEIMotorSsiInput

Definition

```
typedef enum MEIMotorSsiInput{
    MEIMotorSsiInputENC_A,
    MEIMotorSsiInputENC_B,
    MEIMotorSsiInputENC_I,
    MEIMotorSsiInputGPIO_0,
    MEIMotorSsiInputGPIO_1,
    MEIMotorSsiInputGPIO_2,
    MEIMotorSsiInputGPIO_3,
    MEIMotorSsiInputGPIO_4,
    MEIMotorSsiInputGPIO_5,
    MEIMotorSsiInputGPIO_6,
} MEIMotorSsiInput;
```

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

Description

MEIMotorSsiInput is an enumeration of input sources for SSI (Synchronous-Serial Interface) feedback devices. This enumeration is used to specify the input pin for the serial data when using the MEIMotorEncoderSsiConfig configuration structure.

MEIMotorSsiInputENC_A	Encoder input channel A
MEIMotorSsiInputENC_B	Encoder input channel B
MEIMotorSsiInputENC_I	Encoder input channel I
MEIMotorSsiInputGPIO_0	General purpose input bit 0
MEIMotorSsiInputGPIO_1	General purpose input bit 1
MEIMotorSsiInputGPIO_2	General purpose input bit 2
MEIMotorSsiInputGPIO_3	General purpose input bit 3
MEIMotorSsiInputGPIO_4	General purpose input bit 4
MEIMotorSsiInputGPIO_5	General purpose input bit 5
MEIMotorSsiInputGPIO_6	General purpose input bit 6

See Also

MEIMotorEncoderSsiConfig | mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorStatus

Definition

```
typedef struct MEIMotorStatus {
    MEIMotorDacStatus       dac;
    MEIMotorFaultMask       faultMask;
    MEIMotorStepperStatus   stepper;
} MEIMotorStatus;
```

Description

MEIMotorStatus returns the specific Motor Status registers of the controller.

dac	The status for both Command and Auxiliary DACs.
faultMask	The motor fault bit masks for the SynqNet node drive interface.
stepper	Shows the status of the pulse module. It is used for checking network synchronization with the pulse module. It can also be used to check if a pulse velocity/width error occurred.

See Also

MEIMotorStepperStatus

MEIMotorStatusOutput

Definition

```
typedef struct MEIMotorStatusOutput {
    long           *outPtr;
    MEIXmpIOMask   ioMask[MEIXmpMotorStatusOutputs];
} MEIMotorStatusOutput;
```

Description

MEIMotorStatusOutput specifies which motor outputs to follow the motor status bits. This feature requires custom controller firmware.

***outPtr**	a pointer to controller memory.
ioMask	a bit mask to select the motor output signals.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorStepper

Definition

```
typedef struct MEIMotorStepper {
    MPI_BOOL             loopback;   /* TRUE = FPGA pulse feedback,
                                        FALSE = encoder feedback */
    MEIMotorStepperPulse pulseA;
    MEIMotorStepperPulse pulseB;
    double               pulseWidth; /* output pulse width  (sec),
                                        10^(-7) < pulseWidth < 10^(-3) */
} MEIMotorStepper;
```

Change History: Modified in the 03.03.00

Description

MEIMotorStepper is a structure used to configure Stepper Motor parameters.

To use pulse outputs, you will need to:

1. Configure the motor type for STEPPER.
2. See the motor's stepper pulse width. Make sure it meets the drive's requirments and is not smaller than 2x the minimum pulse period.
3. Enable the motor's stepper loopback if there is no encoder feedback.
4. Configure the motor's stepper pulseA and pulseB types for STEP, DIR, CW, CCW, QUADA, or QUADB.
5. Select the motor I/O's config type for pulseA and pulseB. This will route the pulseA/B signals to the node's digital outputs.

loopback†	enables Step Loopback feature. When enabled, step output pulses counted to generate Actual Position. When disabled, external feedback device is required to generate actual position.
pulseA	Structure used to configure Step Pulse A.
pulseB	Structure used to configure Step Pulse B.
pulseWidth†	sets width of Step pulse. Valid values range from a minimum width of 0.1 usec to maximum width of 25.5 usec.

† - If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

MEIMotorStepperPulse | MEIMotorStepperPulseType

MEIMotorStepperPulse

Definition

```
typedef struct MEIMotorStepperPulse {
    MEIMotorStepperPulseType    type;
    MPI_BOOL                    invert;
} MEIMotorStepperPulse;
```

Change History: Modified in the 03.03.00

Description

MEIMotorStepperPulse is a structure used to configure Stepper Motor parameters.

To use pulse outputs, you will need to:

1. Configure the motor type for STEPPER.
2. See the motor's stepper pulse width. Make sure it meets the drive's requirments and is not smaller than 2x the minimum pulse period.
3. Enable the motor's stepper loopback if there is no encoder feedback.
4. Configure the motor's stepper pulseA and pulseB types for STEP, DIR, CW, CCW, QUADA, or QUADB.
5. Select the motor I/O's config type for pulseA and pulseB. This will route the pulseA/B signals to the node's digital outputs.

type	Configures the stepper motor's pulse type. If you attempt to change this value and hardware is not connected, an error message will be returned.
invert	If set to TRUE the actual Pulse output will be inverted by the FPGA. If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

MEIMotorStepper

MEIMotorStepperPulseType

Definition

```
typedef enum MEIMotorStepperPulseType {
    MEIMotorStepperPulseTypeSTEP,
    MEIMotorStepperPulseTypeDIR,

    MEIMotorStepperPulseTypeCW,
    MEIMotorStepperPulseTypeCCW,

    MEIMotorStepperPulseTypeQUADA,
    MEIMotorStepperPulseTypeQUADB,

} MEIMotorStepperPulseType;
```

Description

MEIMotorStepperPulseType is an enumeration used to specify the pulse output signal type.

MEIMotorStepperPulseTypeSTEP	This will enable the pulse output (either A or B) to generate a step output. Use it together with MEIMotorStepperPulseTypeDIR to provide a complete step interface.
MEIMotorStepperPulseTypeDIR	This will enable the pulse output (either A or B) to output the direction of the move. Use it together with MEIMotorStepperPulseTypeSTEP to provide a complete step interface.
MEIMotorStepperPulseTypeCW	This will enable the pulse output (either A or B) to output a clockwise pulse train. Used together with MEIMotorStepperPulseTypeCCW to provide a complete step interface.
MEIMotorStepperPulseTypeCCW	This will enable the pulse output (either A or B) to output a counter-clockwise pulse train. Use it together with MEIMotorStepperPulseTypeCW to provide a complete step interface.
MEIMotorStepperPulseTypeQUADA	This will enable the pulse output (either A or B) to output a quadrature signal (90 degree phase difference to MEIMotorStepperPulseTypeQUADB). Use it together with MEIMotorStepperPulseTypeQUADB to provide a complete quadrature interface.
MEIMotorStepperPulseTypeQUADB	This will enable the pulse output (either A or B) to output a quadrature signal (90 degree phase difference to MEIMotorStepperPulseTypeQUADA). Use it together with MEIMotorStepperPulseTypeQUADA to provide a complete quadrature interface.

See Also

MEIMotorStepperStatus

Definition

```
typedef struct MEIMotorStepperStatus {
   MPI_BOOL pulseLockLost; /* TRUE if Pulse jitter logic has lost lock,
                           otherwise FALSE */
   MPI_BOOL pulseStatus;   /* TRUE if a Pulse was generated
incorrectly,
                           otherwise FALSE */
} MEIMotorStepperStatus;
```

Change History: Modified in the 03.03.00

Description

MEIMotorStepperStatus is a structure used to check for error conditions relating to the pulse module.

pulseLockLost	Indicates that the pulse module lost timing lock with the SynqNet network.
pulseStatus	Indicates that the pulse module generated a new pulse when the previous pulse was not finished. It usually indicates that the pulse width is incorrect for the desired velocity.

See Also

MEIMotorStatus | MEIMotorStepper

MPIMotorType

Definition

```
typedef enum {
    MPIMotorTypeINVALID,

    MPIMotorTypeSERVO,
    MPIMotorTypeSTEPPER,
} MPIMotorType;
```

Description

MPIMotorType is an enumeration of valid Motor Types.

MPIMotorTypeSERVO	Motor configured as Servo
MPIMotorTypeSTEPPER	Motor configured as Stepper

See Also

mpiMotorConfigGet | mpiMotorConfigSet

mpiMotorEncoderFaultMaskBIT

Declaration

```
#define  mpiMotorEncoderFaultMaskBIT(fault)  (0x1 << (fault))
```

Required Header: stdmpi.h

Description

mpiMotorEncoderFaultMaskBIT converts the motor encoder fault into the motor encoder fault mask.

See Also

MPIMotorEncoderFault | MPIMotorEncoderFaultMask

MEIMotorAmpFaultsMAX

Definition

```
#define  MEIMotorAmpFaultsMAX  (32) /* max faults per motor */
```

Description

MEIMotorAmpFaultsMAX defines the maximum number of amp fault messages per motor.

See Also

MEIMotorAmpMsgMAX

Definition

```
#define   MEIMotorAmpMsgMAX    (200) /* max chars per motor */
```

Description

MEIMotorAmpMsgMAX defines the maximum number of characters per amp fault message.

See Also

MEIMotorAmpWarningsMAX

Definition

```
#define  MEIMotorAmpWarningsMAX  (32) /* max Warnings per motor */
```

Description

MEIMotorAmpWarningsMAX defines the maximum number of amp warning messages per motor.

See Also

Notify Objects

Introduction

A thread uses a **Notify** object to wait for event notification. For each thread intended to wait for events from an object (or objects), your application must create a Notify object. The source of firmware events are Motion, Sequence, and Recorder objects.

When it is desired to wait for event notifications from a single source, that source (i.e., object handle) can be passed as the second argument to mpiNotifyCreate. After a Notify object is appended to the EventMgr list of Notify objects, make a call to mpiNotifyEventWait to instruct the Notify object to wait for event notification.

| Error Messages |

Implementation

Notify objects maintain a FIFO ("First In, First Out") buffer of events that have occurred. Each call to mpiNotifyEventWait removes one event from the buffer. If the event buffer is empty, mpiNotifyEventWait(...) will wait for an event to be sent to the **Notify** object. This ensures that events will not be missed in cases where multiple events have occurred between calls to mpiNotifyEventWait(...). However, the danger is that an application may not call mpiNotifyEventWait(...) for a long time. In this case, the event buffer may grow rather quickly and use a large amount of system memory. In order to prevent this problem from occurring, one should use mpiNotifyEventMaskSet to enable and disable event notification at the proper times.

Methods

Create, Delete, Validate Methods

mpiNotify**Create**	Create a Notify object
mpiNotify**Delete**	Delete a Notify object
mpiNotify**Validate**	Validate a Notify object

Event Methods

mpiNotify**Event**	Check to see if events have occurred
mpiNotify**EventFlush**	Flush pending events from event queue
mpiNotify**EventMaskGet**	Get event mask
mpiNotify**EventMaskSet**	Set event mask
mpiNotify**EventWait**	Get next event from queue or wait timeout msec for it to arrive
mpiNotify**EventWake**	Wake up thread waiting for notify object

Relational Methods

List Methods for Event Sources

mpiNotify**Source**	Get the indexth event source in list
mpiNotify**SourceAppend**	Append an event source to list
mpiNotify**SourceCount**	Count number of event sources in list
mpiNotify**SourceFirst**	Get first event source in list
mpiNotify**SourceIndex**	Get index value for event source in list
mpiNotify**SourceInsert**	Place event source after source in list
mpiNotify**SourceLast**	Get last event source in list
mpiNotify**SourceListGet**	Get list of event sources
mpiNotify**SourceListSet**	Create a list of event sources
mpiNotify**SourceNext**	Get next event source after source in list
mpiNotify**SourcePrevious**	Get the event source before source in list
mpiNotify**SourceRemove**	Remove event source from list

Data Types

MPINotify**Message**
MEINotify**Trace**

mpiNotifyCreate

Declaration

```
MPINotify mpiNotifyCreate(MPIEventMask mask,
                          void         *source)
```

Required Header: stdmpi.h

Description

mpiNotifyCreate creates a Notify object that will accept event notifications for the events that are specified in ***mask***. The ***source*** argument specifies the initial element in the list of event sources, from which event notification will be accepted. If ***source*** is NULL, then event notification will be accepted from all event sources. *NotifyCreate* is the equivalent of a C++ constructor.

Return Values	
handle	to a Notify object.
MPIHandleVOID	if the object could not be created.

See Also

mpiNotifyDelete | mpiNotifyValidate

mpiNotifyEventWait

Declaration

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

Required Header: stdmpi.h

Description

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

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

Return Values	
MPIMessageOK	
MPIMessageTIMEOUT	

See Also

mpiNotifyEventWake

mpiNotifyEventMaskSet

Declaration

```
long mpiNotifyEventMaskSet(MPINotify    notify,
                           MPIEventMask mask)
```

Required Header: stdmpi.h

Description

mpiNotifyEventMaskSet sets the event type(s) for which notification will be accepted by a Notify object (***notify***), as specified by ***mask***.

Event notification is accepted for event types specified in ***mask***, a bit mask of MPIEventMask bits (associated with the desired MPIEventType values). The MPIEventMask bits must be set or cleared using the MPIEventMask macros. Event notification is denied for event types not specified in ***mask***.

Return Values	
MPIMessageOK	

Sample Code

```
/*
  Disables event notification and copies the previously used
  event mask to oldMask.  oldMask may then be used to re-enable
  event notification via another call to mpiNotifyEventMaskSet().
*/
void NotifyDisable(MPINotify notify, MPIEventMask* oldMask)
{
    MPIEventMask newMask;
    long returnValue;

    returnValue = mpiNotifyEventMaskGet(notify, oldMask);
    msgCheck(returnValue);

    mpiEventMaskCLEAR(newMask);

    returnValue = mpiNotifyEventMaskSet(notify, newMask);
    msgCheck(returnValue);
}
```

See Also

MPIEventType | mpiNotifyEventMaskGet

mpiNotifyDelete

Declaration

```
long mpiNotifyDelete(MPINotify notify)
```

Required Header: stdmpi.h

Description

mpiNotifyDelete deletes a Notify object and invalidates its handle (***notify***). *NotifyDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiNotifyCreate | mpiNotifyValidate

mpiNotifyValidate

Declaration

```
long mpiNotifyValidate(MPINotify notify)
```

Required Header: stdmpi.h

Description

mpiNotifyValidate validates a Notify object and its handle (***notify***).

Return Values	
MPIMessageOK	

See Also

mpiNotifyEvent

Declaration

```
long mpiNotifyEvent(MPINotify      notify,
                    MPIEventStatus *status)
```

Required Header: stdmpi.h

Description

mpiNotifyEvent first checks to see if the type field of ***status*** matches a bit in the event mask maintained by a Notify object (***notify***).

IF
the type field of ***status*** matches a bit in the event mask,

AND
the event source list maintained by the Notify object (***notify***) is empty or the ***source*** field of ***status*** matches a source in the event source list,

THEN
a Notify object (notify) will place the event in a FIFO event queue and signal that an event has been received.

If a thread is waiting for event notification (after having called mpiNotifyEventWait(notify)), the signal will awaken it. Otherwise, the next call to mpiNotifyEventWait(notify) will return immediately with ***status***.

Return Values	
MPIMessageOK	

See Also

mpiNotifyEventWait

mpiNotifyEventFlush

Declaration

```
long mpiNotifyEventFlush(MPINotify   notify)
```

Required Header: stdmpi.h

Description

mpiNotifyEventFlush flushes any pending events from the internal FIFO event queue maintained by a Notify object (***notify***).

Return Values	
MPIMessageOK	

See Also

mpiNotifyEventMaskGet

Declaration

```
long mpiNotifyEventMaskGet(MPINotify    notify,
                           MPIEventMask *mask)
```

Required Header: stdmpi.h

Description

mpiNotifyEventMaskGet writes an event mask (that specifies the event type(s) for which event notification is accepted by a Notify object (***notify***)) to the location pointed to by ***mask***.

Return Values	
MPIMessageOK	

Sample Code

```
/*
  Disables event notification and copies the previously used
  event mask to oldMask.  oldMask may then be used to re-enable
  event notification via another call to mpiNotifyEventMaskSet().
*/
void NotifyDisable(MPINotify notify, MPIEventMask* oldMask)
{
    MPIEventMask newMask;
    long returnValue;

    returnValue = mpiNotifyEventMaskGet(notify, oldMask);
    msgCheck(returnValue);

    mpiEventMaskCLEAR(newMask);

    returnValue = mpiNotifyEventMaskSet(notify, newMask);
    msgCheck(returnValue);
}
```

See Also

mpiNotifyEventMaskSet

mpiNotifyEventWake

Declaration

```
long mpiNotifyEventWake(MPINotify      notify,
                        MPIEventStatus *status)
```

Required Header: stdmpi.h

Description

mpiNotifyEventWake wakes a thread that is waiting for an event notification from a Notify object (***notify***). The awakened thread will return from its call to mpiNotifyEventWait(notify, status, timeout) with the contents of ***status*** set to the contents of status. If status is NULL, ***status*** will indicate an event of type MPIEventTypeNONE.

NotifyEventWake is different from *NotifyEvent*, because event notification is not accepted based on the event type or source (*NotifyEvent*); instead event notification is always accepted (*NotifyEventWake*).

Return Values	
MPIMessageOK	

See Also

mpiNotifyEventWait | MPIEventType

mpiNotifySource

Declaration

```
void* mpiNotifySource(MPINotify notify,
                      long      index)
```

Required Header: stdmpi.h

Description

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

notify	a handle to the Notify object.
index	a position in the list.

Return Values	
***index*th event source**	in the event source list maintained by a Notify object (***notify***)
NULL	if ***notify*** is invalid if ***index*** is less than 0 if ***index*** is greater than or equal to mpiNotifySourceCount(notify)
MPIMessageARG_INVALID	
MEIListMessageELEMENT_NOT_FOUND	
MPIMessageHANDLE_INVALID	

See Also

mpiNotifySourceAppend

Declaration

```
long mpiNotifySourceAppend(MPINotify notify,
                           void      *source)
```

Required Header: stdmpi.h

Description

mpiNotifySourceAppend appends an event source (***source***) to the list of event sources maintained by a Notify object (***notify***).

notify	a handle to the Notify object.
source	a pointer with an arbitrary (but non-NULL) value.

Return Values	
MPIMessageOK	
MPIMessageHANDLE_INVALID	
MPIMessageNO_MEMORY	

See Also

mpiNotifySourceCount

Declaration

```
long mpiNotifySourceCount(MPINotify notify)
```

Required Header: stdmpi.h

Description

mpiNotifySourceCount returns the number of elements on the list.

notify	a handle to the Notify object.

Return Values	
number of event sources	in the event source list maintained by a Notify object (***notify***)
-1	if ***notify*** is invalid
0	if the event source list is empty

See Also

mpiNotifySourceFirst

Declaration

```
void* mpiNotifySourceFirst(MPINotify notify)
```

Required Header: stdmpi.h

Description

mpiNotifySourceFirst returns the first element in the list. This function can be used in conjuntion with mpiNotifySourceNext() in order to iterate through the list.

notify	a handle to the Notify object.

Return Values	
first event source	in the event source list maintained by a Notify object (***notify***)
NULL	if ***notify*** is invalid or if the event source list is empty
MPIMessageHANDLE_INVALID	if ***notify*** is an invalid handle.

See Also

mpiNotifySourceNext | mpiNotifySourceLast

mpiNotifySourceIndex

Declaration

```
long mpiNotifySourceIndex(MPINotify notify,
                          void      *source)
```

Required Header: stdmpi.h

Description

mpiNotifySourceIndex returns the position of ***source*** on the list.

notify	a handle to the Notify object.
source	a pointer with an arbitrary (but non-NULL) value.

Return Values	
index of *source*	in the event source list maintained by a Notify object (***notify***)
-1	if ***notify*** is invalid if the event source (***source***) was not found in the event source list

See Also

mpiNotifySourceInsert

Declaration

```
long mpiNotifySourceInsert(MPINotify notify,
                           void      *source,
                           void      *insert)
```

Required Header: stdmpi.h

Description

mpiNotifySourceInsert places the event source (pointed to by ***insert***) after a specified event source (***source***), in the list of event sources that are maintained by a Notify object (***notify***).

notify	a handle to the Notify object.

Return Values	
MPIMessageOK	

See Also

mpiNotifySourceLast

Declaration

```
void* mpiNotifySourceLast(MPINotify notify)
```

Required Header: stdmpi.h

Description

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

notify	a handle to the Notify object.

Return Values	
last event source	in the list maintained by a Notify object (***notify***)
NULL	if ***notify*** is invalid if the event source list is empty

See Also

mpiNotifySourcePrevious | mpiNotifySourceFirst

mpiNotifySourceListGet

Declaration

```
long mpiNotifySourceListGet(MPINotify notify,
                            long      *sourceCount,
                            void      **sourceList)
```

Required Header: stdmpi.h

Description

mpiNotifySourceListGet returns the event source list for a Notify object (***notify***). *NotifySourceListGet* writes the number of event sources in the event source list to the location (pointed to by ***sourceCount***), and also writes an array of ***sourceCount*** event source pointers to the location (pointed to by ***sourceList***).

Return Values	
MPIMessageOK	

See Also

mpiNotifySourceListSet | NotifySourceAppend | NotifySourceInsert | mpiNotifyCreate

mpiNotifySourceListSet

Declaration

```
long mpiNotifySourceListSet(MPINotify notify,
                            long      sourceCount,
                            void      **sourceList)
```

Required Header: stdmpi.h

Description

mpiNotifySourceListSet creates an event source list of length ***sourceCount***, using the source pointers specified by ***sourceList***. The ***sourceList*** argument is the address of an array of ***sourceCount*** event source pointers, or is NULL (if ***sourceCount*** = 0). Any existing event source list is completely replaced after using *NotifySourceListSet*.

You can also create an event source list incrementally (i.e., created one source at a time) by using *NotifySourceAppend/Insert* methods. To specify the first event source of a list, use the ***source*** argument of mpiNotifyCreate(...). Use the *NotifySourceList* methods to examine and manipulate an event source list, regardless of how you created it.

Return Values	
MPIMessageOK	

See Also

NotifySourceAppend | NotifySourceInsert | mpiNotifyCreate | mpiNotifySourceListGet

mpiNotifySourceNext

Declaration

```
void * mpiNotifySourceNext(MPINotify  notify,
                           void       *source)
```

Required Header: stdmpi.h

Description

mpiNotifySourceNext returns the next element following "source" on the list. This function can be used in conjuntion with mpiNotifySourceFirst() in order to iterate through the list.

notify	a handle to the Notify object.
source	a pointer with an arbitrary (but non-NULL) value.

Return Values	
event source	before the event source (***source***) in the event source list maintained by a Notify object (***notify***)
NULL	if ***notify*** is invalid if the event source (***source***) is the first event source in the event source list
MPIMessageHANDLE_INVALID	

See Also

mpiNotifySourcePrevious

mpiNotifySourcePrevious

Declaration

```
void * mpiNotifySourcePrevious(MPINotify notify,
                               void      *source)
```

Required Header: stdmpi.h

Description

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

notify	a handle to the Notify object.
source	a pointer with an arbitrary (but non-NULL) value.

Return Values	
event source	before the event source (***source***) in the event source list maintained by a Notify object (***notify***)
NULL	if ***notify*** is invalid if the event source (***source***) is the first event source in the event source list
MPIMessageHANDLE_INVALID	

See Also

mpiNotifySourceNext

mpiNotifySourceRemove

Declaration

```
long mpiNotifySourceRemove(MPINotify notify,
                           void      *source)
```

Required Header: stdmpi.h

Description

mpiNotifySourceRemove removes an event source (***source***) from the list of event sources maintained by a Notify object (***notify***).

Return Values	
MPIMessageOK	

See Also

MPINotifyMessage

Definition

```
typedef enum {

    MPINotifyMessageNOTIFY_INVALID,
    MPINotifyMessageWAIT_IN_PROGRESS,
} MPINotifyMessage;
```

Description

MPINotifyMessage is an enumeration of the Notify error messages that can be returned by the MPI library.

MPINotifyMessageNOTIFY_INVALID

The notify object is not valid. This message code is returned by a notify method if the notify object handle is not valid. This problem can be caused by failed mpiNotifyCreate(...). To prevent this problem, check your notify objects after creation by using mpiNotifyValidate(...).

MPINotifyMessageWAIT_IN_PROGRESS

The notify object is waiting for an event. This message code is returned by mpiNotifyEventWait(...) if the notify object is already waiting for an event in another thread. To prevent this problem, make sure a thread does not share notify objects with other threads.

Sample Code

```
MPIControl    control;
MPINotify     notify;
long          returnValue;

...

notify =
        mpiNotifyCreate(control);
returnValue =
        mpiNotifyValidate(notify);
```

See Also

MPINotify | mpiNotifyCreate | mpiNotifyValidate

MEINotifyTrace

Definition

```
typedef enum {
    MEINotifyTraceTHREAD,
} MEINotifyTrace;
```

Description

MEINotifyTrace holds information about a particular event that was generated by the XMP.

MEINotifyTraceTHREAD	will display trace information when notify objects are set to wait, are finished waiting, and when they are signaled to wake.

See Also

Object Interface

Introduction

Each **Object** shares some common functionality with other objects. The Object module encapsulates the common object methods and macros into a single module. This makes object handling consistent and more efficient.

Methods

Create, Delete, Validate Methods

mpiObject**Validate**

Configuration and Information Methods

mpiObject**ModuleId**

mpiObject**TimeoutGet**

mpiObject**TimeoutSet**

meiObject**TraceGet**

meiObject**TraceSet**

Data Types

MPIObject**Map**

Constants

MEIObject**LabelCharMAX**

Macros

mpiObjectMap**AND_ASSIGN**

mpiObjectMap**ASSIGN**

mpiObjectValidate

Declaration

```
long mpiObjectValidate(MPIHandle   handle,
                       MPIModuleId moduleId)
```

Required Header: stdmpi.h

Description

mpiObjectValidate verifies that the object (handle) is of the type moduleId.

Return Values	
MPIMessageOK	

See Also

mpiObjectModuleId

Declaration

```
long mpiObjectModuleId(MPIHandle    handle,
                       MPIModuleId  *moduleId)
```

Required Header: stdmpi.h

Description

mpiObjectModuleId function writes the MPIModuleId of the object (***handle***) to the location pointed to by ***moduleId***.

Return Values	
MPIMessageOK	

See Also

mpiObjectTimeoutGet

Declaration

```
long mpiObjectTimeoutGet(MPIHandle handle,
                         MPIWait   *timeout)
```

Required Header: stdmpi.h

Description

mpiObjectTimeoutGet method gets the timeout value for the object (***handle***), and writes it to the location pointed to by ***timeout***.

Return Values	
MPIMessageOK	

See Also

mpiObjectTimeoutSet

mpiObjectTimeoutSet

Declaration

```
long mpiObjectTimeoutSet(MPIHandle handle,
                         MPIWait   timeout)
```

Required Header: stdmpi.h

Description

mpiObjectTimeoutSet sets the timeout value for an object (***handle***) to ***timeout***.

The ***timeout*** value is used in a multi-threaded environment when an MPI function must block to wait for a shared resource to become available. The default ***timeout*** value is MPIWaitFOREVER.

If "timeout" is	Then
MPIWaitFOREVER	*ObjectTimeoutSet* will wait until the resource becomes available
MPIWaitPOLL	*ObjectTimeoutSet* will not wait for the resource to become available. If the shared resource is not available, a timeout will be considered to have occurred.
a value	*ObjectTimeoutSet* will wait timeout milliseconds for the resource to become available

Return Values	
MPIMessageOK	
MPIMessageTIMEOUT	

See Also

mpiObjectTimeoutGet

meiObjectTraceGet / meiObjectTraceGET

Declaration: meiObjectTraceGet

```
long meiObjectTraceGet(MPIHandle    handle,
                       MEITraceMask *traceMask)
```

Required Header: stdmei.h

Description

meiObjectTraceGet gets an Object's trace mask and writes it to the value pointed to by traceMask.

handle	a handle to an object
***traceMask**	a pointer to the value of an object's trace mask

Return Values	
MPIMessageOK	if *ObjectTraceGet* successfully gets the trace mask for an object.

Declaration: meiObjectTraceGET

```
#define  meiObjectTraceGET(object) (((MEIObject)(object))->trace)
```

Required Header: stdmei.h

Description

meiObjectTraceGET gets the object's global trace mask.

See Also

meiObjectTraceSET | meiObjectTraceSet

meiObjectTraceSet / meiObjectTraceSET

Declaration: meiObjectTraceSet

```
long meiObjectTraceSet(MPIHandle    handle,
                       MEITraceMask traceMask)
```

Required Header: stdmei.h

Description

meiObjectTraceSet sets an Object's trace mask to the value specified by ***traceMask***.

handle	a handle to an object
traceMask	the value of an object's trace mask

Return Values	
MPIMessageOK	if *ObjectTraceSet* successfully sets the trace mask for an object.

Declaration: meiObjectTraceSET

```
#define  meiObjectTraceSET(object,mask)
                          (((MEIObject)(object))->trace = (mask))
```

Required Header: stdmei.h

Description

meiObjectTraceSET sets the object's global trace mask.

See Also

meiObjectTraceGET | meiObjectTraceGet

MPIObjectMap

Definition

```
typedef  unsigned  long  MPIObjectMap;
```

Description

MPIObjectMap contains the amp fault information from a SynqNet drive. The amp fault messages are drive specific. Not all drives support amp fault messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

MPIObjectMap	A map of MPI objects. An ObjectMap is a bitmap, where each numbered bit represents the presence or absence of the correspondingly numbered object. Valid maps are Axis/Filter and Filter/Motor. The Axis/Filter map can also be read, but setting this map must be done through the corresponding Filter objects.

See Also

MEIObjectLabelCharMAX

Definition

```
#define  MEIObjectLabelCharMAX    (16)
```

Change History: Added in the 03.03.00

Description

MEIObjectLabelCharMAX defines the maximum number of characters that can be stored for Object User Labels.

See Also

mpiObjectMapAND_ASSIGN

Declaration

```
#define  mpiObjectMapAND_ASSIGN(dst,src)        ((dst) &= (src))
```

Required Header: stdmpi.h

Description

Bitwise ANDs the dst object map with the ***src*** object map and assigns the result to ***dst***.

See Also

mpiObjectMapASSIGN

mpiObjectMapASSIGN

Declaration

```
#define  mpiObjectMapASSIGN(dst,src)  ((dst) = (src))
```

Required Header: stdmpi.h

Description

mpiObjectMapASSIGN assigns ***src*** object map to the ***dst*** object map.

See Also

mpiObjectMapAND_ASSIGN

mpiObjectMapBitCountMAX

Declaration

```
#define  mpiObjectMapBitCountMAX(objectMap)
                                 (sizeof(objectMap) * 8)
```

Required Header: stdmpi.h

Description

mpiObjectMapBitCountMAX calculates the maximum bit count for the specified object ***objectMap***.

See Also

mpiObjectMapBitGET

Declaration

```
#define mpiObjectMapBitGET(objectMap,bit)
                           (((objectMap) &  (0x1 << (bit))) ? 1 : 0)
```

Required Header: stdmpi.h

Description

mpiObjectMapBitGET gets the bit number's state for the specified object ***map***.

See Also

mpiObjectMapBitSET

mpiObjectMapBitSET

Declaration

```
#define  mpiObjectMapBitSET(objectMap,bit,value) \
            (((value) == 0) \
                 ? ((objectMap) &= ~(0x1 << (bit))) \
                 : ((objectMap) |=  (0x1 << (bit))))
```

Required Header: stdmpi.h

Description

mpiObjectMapBitSET sets object map's specified ***bit*** number to the specified ***value***.

See Also

mpiObjectMapBitGET

mpiObjectMapCLEAR

Declaration

```
#define mpiObjectMapCLEAR(objectMap)  ((objectmap) = 0x0)
```

Required Header: stdmpi.h

Description

mpiObjectMapCLEAR sets the object ***map*** to zero.

See Also

mpiObjectMapCOMPLEMENT

Declaration

```
#define  mpiObjectMapCOMPLEMENT(objectMap)
                                ((objectMap) = ~(objecMap))
```

Required Header: stdmpi.h

Description

mpiObjectMapCOMPLEMENT performs bitwise complement to the object map and assigns the result to ***objectMap***.

See Also

mpiObjectMapIS_CLEAR

Declaration

```
#define mpiObjectMapIS_CLEAR(objectMap) ((objectMap) == 0x0)
```

Required Header: stdmpi.h

Description

mpiObjectMapIS_CLEAR compares the map to 0x0. If the map is zero, it returns a non-zero value.

See Also

mpiObjectMapIS_EQUAL

Declaration

```
#define  mpiObjectMapIS_EQUAL(map1,map2)        ((map1) == (map2))
```

Required Header: stdmpi.h

Description

mpiObjectMapIS_EQUAL compares map1 to map2. If the maps are equal, it returns a non-zero value.

See Also

mpiObjectMapIS_VALID

Declaration

```
#define  mpiObjectMapIS_VALID(objectMap,count) \
     (((count) >= (sizeof(objectMap) * 8)) || \
     (((~((0x1 << (count)) - 1)) & (objectMap)) == 0))))
```

Required Header: stdmpi.h

Description

mpiObjectMapIS_VALID checks if the map is within the count range. If the map is valid, it returns a non-zero value.

See Also

mpiObjectMapMAX

Declaration

```
#define mpiObjectMapMAX(objectMap,count)
                        ((objectMap) = (0x1 << (count)) - 1)
```

Required Header: stdmpi.h

Description

mpiObjectMapMAX calculates the maximum object ***objectMap*** with the specified ***count***.

See Also

mpiObjectMapOR_ASSIGN

Declaration

```
#define  mpiObjectMapOR_ASSIGN(dst,src)    ((dst) |= (src))
```

Required Header: stdmpi.h

Description

Bitwise ORs the dst object map with the ***src*** object map and assigns the result to ***dst***.

See Also

mpiObjectMapASSIGN

Path Objects

Introduction

A **Path** object manages coordinated multi-axis motion profiles. It is used when the motion profiles in an N-Dimensional space are required to follow a specific coordinated trajectory. Motion paths are constructed with high level linear and arc segments and downloaded to the controller. The controller calculates the real-time individual axis profiles.

Generally, Path motion is used when the trajectory through space is more important than the final target position. Several different algorithms can be applied to convert the linear and arc segment path into an interpolated trajectory.

Path trajectory generation is now supported by PT, PVT, SPLINE, BESSEL, BSPLINE, and BSPLINE2 algorithms. Blending of the corners is only available for the 2 bspline algorithms. Blending of a corner is when the path does not hit the corner but goes through a smooth arc.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiPath**Create**	Create a Path object
mpiPath**Delete**	Delete a Path object
mpiPath**Validate**	Validate a Path object

Configuration and Information Methods

mpiPath**ParamsGet**

mpiPath**ParamsSet**

Relational Methods

mpiPath**Append**

Action Methods

mpiPath**MotionParamsGenerate**

Data Types

MPIPath**Arc**

MPIPath**ArcCenter**

MPIPath**ArcEndPoint**

MPIPath**Attr**
MPIPath**Direction**
MPIPath**Element**
MPIPath**ElementAttributes**
MPIPath**ElementAttrMask**
MPIPath**ElementType**
MPIPath**Line**
MPIPath**Message**
MPIPath**Params**
MPIPath**Point**

Macros

mpiPath**ElementTYPE**
mpiPath**ElementAttrMaskBIT**
mpiPath**ElementATTR**

Constants

MPIPathPoint**DIMENSION_MAX**
MPIPathPoint**POINTS_MAX**

mpiPathCreate

Declaration

```
MPIPath mpiPathCreate();
```

Required Header: stdmpi.h

Description

mpiPathCreate creates a Path object.

Return Values	
handle	to a Path object
MPIHandleVOID	if the object could not be created

See Also

mpiPathDelete | mpiPathValidate

mpiPathDelete

Declaration

```
long mpiPathDelete(MPIPath path);
```

Required Header: stdmpi.h

Description

mpiPathDelete deletes a Path object and invalidates its handle (***path***). PathDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiPathCreate | mpiPathValidate

mpiPathValidate

Declaration

```
long mpiPathValidate(MPIPath path);
```

Required Header: stdmpi.h

Description

mpiPathValidate validates the Path object and its handle (***command***).

Return Values	
MPIMessageOK	

See Also

mpiPathCreate | mpiPathDelete

mpiPathParamsGet

Declaration

```
long mpiPathParamsGet(MPIPath          path,
                      MPIPathParams    *params,
                      void             *external),
```

Required Header: stdmpi.h

Description

mpiPathParamsGet reads the parameters for a path object and writes them into the structure pointed to by ***params***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

path	a handle to a Path object
***params**	a pointer to a MPIPathParams structure
***external**	a pointer to a void or NULL

Return Values	
MPIMessageOK	

See Also

mpiPathParamsSet

mpiPathParamsSet

Declaration

```
long mpiPathParamsSet(MPIPath        path,
                      MPIPathParams  *params,
                      void           *external),
```

Required Header: stdmpi.h

Description

mpiPathParamsSet writes the parameters from the structure pointed to by ***params*** into the Path object. Also, it writes the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL) into the Path.

path	a handle to a Path object
***params**	a pointer to a MPIPathParams structure
***external**	a pointer to a void or NULL

Return Values	
MPIMessageOK	

See Also

mpiPathParamsGet

mpiPathAppend

Declaration

```
long mpiPathAppend(MPIPath        path,
                   MPIPathElement *element);
```

Required Header: stdmpi.h

Description

mpiPathAppend adds an array of path elements pointed to by ***element*** to the end of the ***path*** stored in the Path object.

Return Values	
MPIMessageOK	

See Also

mpiPathCreate | mpiPathMotionParamsGenerate

mpiPathMotionParamsGenerate

Declaration

```
long mpiPathMotionParamsGenerate(MPIPath          path,
                                 MPIMotionParams  *params);
```

Required Header: stdmpi.h

Description

mpiPathMotionParamsGenerate calculates a list of points from the path object and writes them into the motion params structure pointed to by params. After the motion params are generated, they can be passed to mpiMotionStart(...). Path generated params are supported with the following motion types:

MPIMotionTypePT
MPIMotionTypePTF
MPIMotionTypePVT
MPIMotionTypePVTF
MPIMotionTypeSPLINE
MPIMotionTypeBESSEL
MPIMotionTypeBSPLINE
MPIMotionTypeBSPLINE2

To create a path, use mpiPathCreate(...) to create a path object. Initialize the path parameters with mpiPathParamsGet(...) / mpiPathParamsSet(...). Then add path elements (line, arc, etc.) with mpiPathAppend(...). Before calling mpiPathMotionParamsGenerate(...) make sure to specify the MPIMotionPoint{...} values in the MPIMotionParams structure. It is very important to set point.final = TRUE or FALSE before calling mpiPathMotionParamsGenerate(...).

path	a handle to a Path object
***params**	a pointer to a MPIMotionParams structure.

Return Values	
MPIMessageOK	

See Also

mpiPathCreate | mpiPathParamsGet | mpiPathParamsSet | mpiPathAppend | mpiMotionStart

MPIPathArc

Definition

```
typedef struct MPIPathArc {
        struct {
                double  start;
                double  included;
        } angle;
        double  radius;
} MPIPathArc;
```

Description

MPIPathArc specifies the parameters for an arc path element. It supports 2 dimensional arcs only. All arcs start at the end position for the last path element added to the path or the present command position if the arc is the first element in the path.

start	This value defines the arc's starting angle. Units are in degrees.
included	This value defines the relative travel angle. Units are in degrees. Positive values specify counter-clockwise motion and negative values specify clockwise motion.
radius	This value defines the distance from the center to the arc edge. Units are in counts.

See Also

MPIPathElement | MPIPathParams | MPIPathArcCenter

MPIPathArcCenter

Definition

```
typedef struct MPIPathArcCenter {
   MPIPathPoint   center;
   double         angle;
} MPIPathArcCenter;
```

Description

MPIPathArcCenter specifies the parameters for an arc path element. It supports 2 dimensional arcs only. All arcs start at the end position for the last path element added to the path or the present command position if the arc is the first element in the path.

center	This structure defines the coordinates for the center point of the arc. Please see MPIPathPoint data type documentation for more information.
angle	This value defines the relative travel angle. Units are in degrees. Positive values specify counter-clockwise motion and negative values specify clockwise motion.

See Also

MPIPathElement | MPIPathParams | MPIPathArc

MPIPathArcEndPoint

Definition

```
typedef struct MPIPathArcEndPoint {
   MPIPathPoint       center;
   MPIPathPoint       endPoint;
   MPIPathDirection   direction;
} MPIPathArcEndPoint;
```

Description

MPIPathArcEndPoint specifies the parameters for an arc path element. It supports 2 dimensional arcs only. All arcs start at the end position for the last path element added to the path or the present command position if the arc is the first element in the path.

center	This structure defines the coordinates for the center point of the arc. Please see MPIPathPoint data type documentation for more information.
endPoint	This structure defines the coordinates for the final point of the arc. Please see MPIPathPoint data type documentation for more information.
direction	This value defines the travel direction, counter-clockwise or clockwise. Please see MPIPathDirection data type documentation for more information.

See Also

MPIPathElement | MPIPathParams | MPIPathDirection

MPIPathAttr

Definition

```
typedef enum {
   MPIPathElementAttrINVALID    = -1,
   MPIPathElementAttrRELATIVE   = MPIPathElementAttrLAST - 1,
   MPIPathElementAttrID         = MPIPathElementAttrLAST - 2,
   MPIPathElementAttrVELOCITY   = MPIPathElementAttrLAST - 3,
   MPIPathElementAttrACCEL      = MPIPathElementAttrLAST - 4,
   MPIPathElementAttrTIMESLICE  = MPIPathElementAttrLAST - 5,
   MPIPathElementAttrCOUNT      = MPIPathElementAttrLAST - MPIPathElementAttrFIRST,
} MPIPathAttr;
```

Description

In **MPIPathAttr**, the path attributes are used to generate the path attribute masks to enable features with mpiPathAppend (...). Please see MPIPathElementAttrMask data type for more information.

See Also

mpiPathAppend

MPIPathDirection

Definition

```
typedef enum {
   MPIPathDirectionCW  = -1,
   MPIPathDirectionCCW = 1,
} MPIPathDirection;
```

Description

MPIPathDirectionCW	This value defines the clockwise direction.
MPIPathDirectionCCW	This value defines the counter-clockwise direction.

See Also

MPIPathArcEndPoint

MPIPathElement

Definition

```
typedef struct MPIPathElement {
   MPIPathElementType    type;
   long                  blending;
   union {
     MPIPathArc          arc;
     MPIPathArcCenter    arcCenter;
     MPIPathArcEndPoint  arcEndPoint;
     MPIPathLine         line;
} params;

     MPIPathElementAttributes    attributes;
} MPIPathElement;
```

Description

type	This value defines the type of path element. Please see MPIPathElementType data type documentation for more information.
blending	This value determines whether the corners of the path are rounded or sharp. When set to TRUE, blending is enabled, causing rounded corners. When set to FALSE, blending is disabled, causing sharp corners.
arc	This structure defines the arc's start angle, included angle, and radius. This structure is used when the type is MPIPathElementTypeARC. Please see MPIPathArc data type documentation for more information.
arcCenter	This structure defines the arc's center and angle. This structure is used when the type is MPIPathElementTypeARC_CENTER. Please see MPIPathArcCenter data type documentation for more information.
arcEndPoint	This structure defines the arc's center, end point, and direction. This structure is used when the type is MPIPathElementTypeARC_END_POINT. Please see MPIPathArcEndPoint data type documentation for more information.
line	This structure defines the coordinates for a linear element. This structure is used when the type is MPIPathElementTypeLINE. Please see MPIPathLine data type documentation for more information.
attributes	This structure defines the attributes for a path element. Please see MPIPathElementAttributes data type documentation for more information.

See Also

mpiPathAppend

MPIPathElementAttributes

Definition

```
typedef struct MPIPathElementAttributes {
    long    id;              /* MPIPathAttrID           */
    double  velocity;        /* MPIPathAttrVELOCITY     */
    double  acceleration;    /* MPIPathAttrACCELERATION */
    double  timeSlice;       /* MPIPathAttrTIMESLICE    */
} MPIPathElementAttributes;
```

Description

In **MPIPathElementAttributes**, the path attributes define the parameters to be used when specific features are enabled with the path element attribute masks. When using these attributes, be sure to enable the feature with the appropriate MPIPathElementAttrMask{.}.

id	This value defines an identification number to be stored in the path element. During path profile execution, at the start of each element the controller loads the id into the axis' ElementID field. The application can query the controller's axis memory to monitor the path element execution. The id is limited to 16-bit resolution by the controller firmware.
velocity	This value defines the velocity for the path element.
acceleration	This value defines the acceleration for the path element.
timeSlice	This value defines the time between interpolation points for the path element. The practical range for the time slice is from 10 msec (.01) to 100 msec (.1). Larger time slice values produce smoother (lower acceleration), less accurate paths. Smaller time slice values produce more accurate (both position and velocity) paths with higher peak accelerations.

See Also

MPIPathElementAttrMask

MPIPathElementAttrMask

Definition

```
typedef enum {
   MPIPathElementAttrMaskRELATIVE,
       = mpiPathElementAttrMaskBIT(MPIPathElementAttrRELATIVE),
   MPIPathElementAttrMaskID,
       = mpiPathElementAttrMaskBIT(MPIPathElementAttrID),
   MPIPathElementAttrMaskVELOCITY,
       = mpiPathElementAttrMaskBIT(MPIPathElementAttrVELOCITY),
   MPIPathElementAttrMaskACCEL,
       = mpiPathElementAttrMaskBIT
(MPIPathElementAttrACCEL),
   MPIPathElementAttrMaskTIMESLICE,
       = mpiPathElementAttrMaskBIT(MPIPathElementAttrTIMESLICE),

   MPIPathElementAttrMaskALL   = -1 << MPIPathElementAttrFIRST,
} MPIPathElementAttrMask;
```

Description

In **MPIPathElementAttrMask**, the path attribute masks are used to enable features with mpiPathAppend(...). The masks are ORed with the MPIPathElementType to enable each feature.

MPIPathElementAttrMaskRELATIVE	This mask enables relative coordinates for path motion. This feature is not supported and is reserved for future use.
MPIPathElementAttrMaskID	This mask enables an identification tag to be stored in the path. Each element can have a unique identification. Please see MPIPathElementAttributes data type documentation for more information.
MPIPathElementAttrMaskVELOCITY	This mask enables a path velocity to be specified for each element. Please see MPIPathElementAttributes data type documentation for more information.
MPIPathElementAttrMaskACCEL	This mask enables a path acceleration to be specified for each element. Please see MPIPathElementAttributes data type documentation for more information.
MPIPathElementAttrMaskTIMESLICE	This mask enables a path time slice to be specified for each element. Please see MPIPathElementAttributes data type documentation for more information.

See Also

MPIPathElementType | mpiPathAppend

MPIPathElementType

Definition

```
typedef enum {
   MPIPathElementTypeINVALID,
   MPIPathElementTypeARC,            /* only 2D */
   MPIPathElementTypeARC_CENTER,     /* only 2D */
   MPIPathElementTypeARC_END_POINT, /* both 2D and 3D */
   MPIPathElementTypeHELIX,          /* not currently supported */
   MPIPathElementTypeIO,             /* not currently supported */
   MPIPathElementTypeLINE,           /* both 2D and 3D */

        MPIPathElementTypeMASK,
} MPIPathElementType;
```

Description

MPIPathElementTypeARC	This type generates an arc specified by the arc's start angle, included angle, and radius.
MPIPathElementTypeARC_CENTER	This type generates an arc specified by the arc's center and angle.
MPIPathElementTypeARC_END_POINT	This type generates an arc specified by the arc's center, end point, and direction.
MPIPathElementTypeLINE	This type generates a line specified by the position coordinates.

See Also

MPIPathArc | MPIPathLine

MPIPathLine

Definition

```
typedef struct MPIPathLine {
   MPIPathPoint   point;
} MPIPathLine;
```

Description

MPIPathLine specifies the parameters for a linear path element. It supports up to MPIPathPointDIMENSION_MAX dimensions. All lines start at the end position for the last path element added to the path or the present command position if the line is the first element in the path.

point	This structure defines the end point coordinates for the linear segment.

See Also

MPIPathElement | MPIPathParams | MPIPathPointDIMENSION_MAX

MPIPathMessage

Definition

```
typedef enum {
   MPIPathMessagePATH_INVALID,
   MPIPathMessageILLEGAL_DIMENSION,
   MPIPathMessageILLEGAL_ELEMENT,
   MPIPathMessageARC_ILLEGAL_DIMENSION,
   MPIPathMessageHELIX_ILLEGAL_DIMENSION,
   MPIPathMessageILLEGAL_RADIUS,
   MPIPathMessagePATH_TOO_LONG,
   MPIPathMessageILLEGAL_VELOCITY,
   MPIPathMessageILLEGAL_ACCELERATION,
   MPIPathMessageILLEGAL_TIMESLICE,
   MPIPathMessageINVALID_BLENDING,
} MPIPathMessage;
```

Description

MPIPathMessage is an enumeration of the Path error messages that can be returned by the MPI library.

MPIPathMessagePATH_INVALID

The path object is not valid. This message code is returned by a path method if the path object handle is not valid. This problem can be caused by a failed mpiPathCreate(...). To prevent this problem, check your path objects after creation by using mpiPathValidate(...).

MPIPathMessageILLEGAL_DIMENSION

The path dimensions are not valid. This message code is returned by mpiPathParamsSet(...) or mpiPathMotionParamsGenerate(...) if the path dimension is less than one or greater than or equal to MPIPathPointDIMENSION_MAX. Also, this message code is returned if specific path element types have dimension restrictions. For example, the ARC type is limited to 2 dimensions and the ARC_END_POINT type is limited to 3 dimensions. To correct this problem, select an appropriate dimension for the path element type.

MPIPathMessageILLEGAL_ELEMENT

The path element type is not valid. This message code is returned by mpiPathAppend(...) if the specified path element type is not a member of the MPIPathElementType enumeration.

MPIPathMessageARC_ILLEGAL_DIMENSION

The path element arc dimension is not valid. This message code is returned by mpiPathAppend(...) if the ARC or ARC_CENTER element is not 2 dimensions. To correct this problem, set the path dimension to 2.

MPIPathMessageHELIX_ILLEGAL_DIMENSION

Not supported.

MPIPathMessageILLEGAL_RADIUS

The path element arc radius is not valid. This message code is returned by mpiPathAppend(...) if the ARC element radius is less than or equal to zero. To correct this problem, set the arc radius to a value greater than zero.

MPIPathMessagePATH_TOO_LONG

The path length is not valid. This message code is returned by mpiPathMotionParamsGenerate(...) if the path length is greater than MAX_PATH_POINTS. To correct the problem, specify a path with fewer points than MAX_PATH_POINTS.

MPIPathMessageILLEGAL_VELOCITY

The path element velocity is not valid. This message code is returned by mpiPathAppend(...) if the specified velocity is less than or equal to zero. To correct this problem, set the element velocity to a value greater than zero.

MPIPathMessageILLEGAL_ACCELERATION

The path element velocity is not valid. This message code is returned by mpiPathAppend(...) if the specified velocity is less than or equal to zero. To correct this problem, set the element velocity to a value greater than zero.

MPIPathMessageILLEGAL_TIMESLICE

The path element time slice is not valid. This message code is returned by mpiPathAppend(...) if the specified time slice is less than or equal to zero. To correct this problem, set the element time slice to a value greater than zero.

MPIPathMessageINVALID_BLENDING

The path element blending is not valid. This message code is returned by mpiPathMotionParamsGenerate(...) if the element blending is set to TRUE and the motion type does not support blending. To correct this problem, either set the element blending to FALSE or select a different motion type.

See Also

MPIPathParams

Definition

```
typedef struct MPIPathParams {
    long              dimension;
    MPIPathPoint      start;
    double            velocity;
    double            acceleration;
    double            deceleration;
    MPIMotionType     interpolation;
    double            timeSlice;
    double            conversion
                      [MPIPathPointDIMENSION_MAX][MPIPathPointDIMENSION_MAX];
} MPIPathParams;
```

Description

dimension	This value defines the number of axes to coordinate. Please see MPIPathPoint data type documentation for more information.
start	This structure defines the initial point for the path.
velocity	This value defines the speed along the path. The units are in counts per second.
acceleration	This value defines the rate of change of speed to reach the velocity along the path. The units are in counts per second * second.
deceleration	This value defines the rate of change of speed to reach zero velocity along the path. The units are in counts per second * second.
interpolation	This value specifies the motion algorithm to generate the path. Please see MPIMotionType data type documentation for more information.
timeSlice	This value specifies the amount of time between points. The units are in seconds. Smaller timeSlice values will improve the path accuracy, but increase the number of points to calculate and buffer.
conversion	This value is an *N* x *N* matrix (where *N* is the number of dimensions in the path motion) that scales and rotates the axes used in the path motion. This is useful when using two axes with different resolutions for each axis. **For two axes with different resolution:** Set conversion[0][0] to (desired x resolution / actual x resolution) Set conversion [1][1] to (desired y resolution / actual y resolution) Set conversion[0][1] and conversion[1][0] = 0 **For a coordinate rotation, where alpha is the rotation of the coordinate system:** Set conversion[0][0] and conversion [1][1] = cos(alpha) Set conversion[0][1] = sin(alpha)

	Set conversion[1][0] = -sin(alpha)

See Also

mpiPathParamsGet | mpiPathParmsSet | mpiPathMotionParamsGenerate | MPIPathPointDIMENSION_MAX

MPIPathPoint

Definition

```
typedef struct MPIPathPoint {
    double   position[MPIPathPointDIMENSION_MAX];
} MPIPathPoint;
```

Description

position	This array defines the axis command positions for a path point. There must be one position value for each dimension.

See Also

MPIPathParams | mpiPathParamsGet | mpiPathParmsSet | mpiPathPointDIMENSION_MAX

mpiPathElementTYPE

Declaration

```
#define mpiPathElementTYPE(type) ((type) & MPIPathElementTypeMASK)
```

Required Header: stdmpi.h

Description

mpiPathElementTYPE is a macro that masks off all other bits in type, leaving the path element type.

See Also

MPIPathElementType

mpiPathElementAttrMaskBIT

Declaration

```
#define mpiPathElementAttrMaskBIT(attr) (0x1 << (attr))
```

Required Header: stdmpi.h

Description

mpiPathElementAttrMaskBIT is a macro that converts the path element attribute into the path element attribute mask.

See Also

MPIPathElementAttrs | MPIPathElementAttrMask

mpiPathElementATTR

Declaration

```
#define mpiPathElementATTR(type,attr)
                           ((type) |= mpiPathElementAttrMaskBIT(attr))
```

Required Header: stdmpi.h

Description

mpiPathElementATTR is a macro that turns on the specified path element attribute mask bits in the path element type.

See Also

MPIPathAttr | MPIPathElementAttrMask

MPIPathPointDIMENSION_MAX

Definition

```
#define MPIPathPointDIMENSION_MAX       (16)
```

Description

MPIPathPointDIMENSION_MAX defines the maximum dimensions for path objects.

See Also

MPIPathParams | mpiPathParamsGet | mpiPathParmsSet | mpiPathPointDIMENSION_MAX

MPIPathPointPOINTS_MAX

Definition

```
#define MPIPathPointPOINTS_MAX  (5000)
```

Description

MPIPathPointPOINTS_MAX defines the maximum number of points for a path.

See Also

MPIPathParams | mpiPathMotionParamsGenerate

Platform Objects

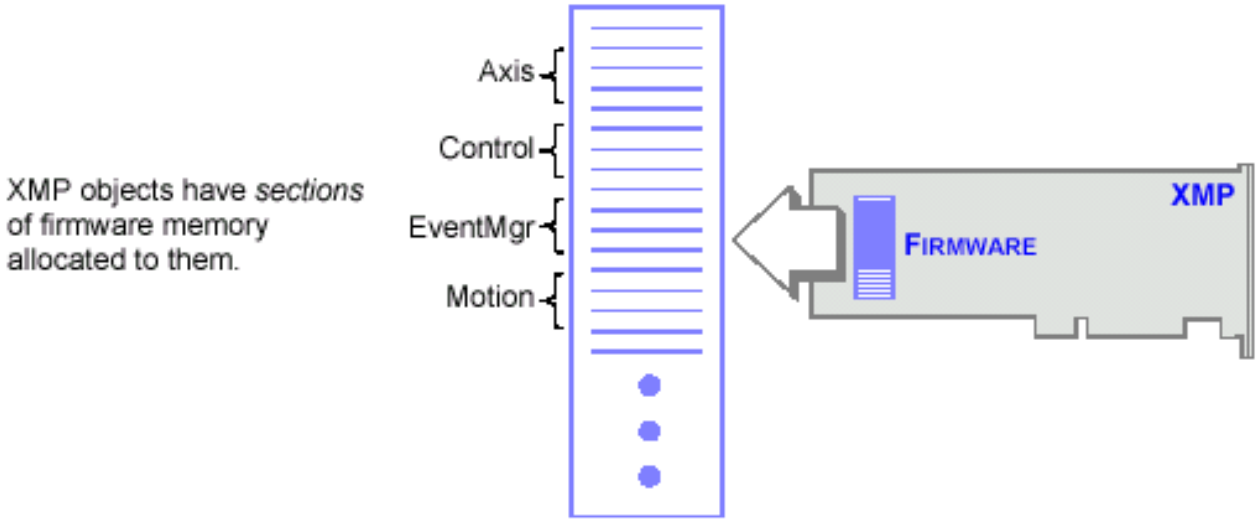
Introduction

The **Platform** module provides a common interface to platform-specific functionality, such as memory allocation, resource locking, interrupts, signalling, and others.

The Platform object provides low-level platform-specific functionality and depends upon the combination of the operating system and the C compiler used for development. The Platform module was written to provide platform-independent access functions for use by the MPI. Unless your application needs to be written for compatibility with different platforms, MEI encourages the use of OS-specific functions. If an MEIPlatform object handle is required, one should obtain this handle from the MPIControl method meiControlPlatform.

WARNING!
Do NOT attempt to use the (intentionally undocumented) method, meiPlatformCreate(). Using this method will interfere with the inner workings of the MPI.

The mei*Object*Give/Take(...) methods all use the meiPlatformLockGive**/**Take(...) methods. When you take a lock, you take exclusive access to the resource (i.e., the section of XMP firmware memory associated with that Object). When you give a lock, you release (give up) that exclusive access. Think of it as TakeAccessOf and GiveUpAccess.



Methods

meiPlatform**Alloc**	Allocate system memory.
meiPlatform**AssertSet**	Set an assertion handling function to be used by the MPI library.

meiPlatform**Atof**	Convert a numeric string to a double.
meiPlatform**Atol**	Convert a numeric string to a long.
meiPlatform**ExmpTempGet**	
meiPlatform**ExmpTempInit**	
meiPlatform**FileClose**	Close a file handle.
meiPlatform**FileOpen**	Open a file handle.
meiPlatform**FileRead**	Read data from a file handle created by meiPlatformFileOpen.
meiPlatform**FileWrite**	Writes data to a file whose handle was created by meiPlatformFileOpen.
meiPlatform**FirmwareAddress16To32**	
meiPlatform**FirmwareAddress32To16**	
meiPlatform**FirmwareAddress32To64**	
meiPlatform**FirmwareAddress64To32**	
meiPlatform**Free**	Free system memory.
meiPlatform**HostAddress32To64**	
meiPlatform**HostAddress64To32**	
meiPlatform**Key**	Return an input character if an input character is available.
meiPlatform**MemoryGet64**	
meiPlatform**MemorySet64**	
meiPlatform**MemoryToFirmware**	Convert a host memory address to a controller memory address.
meiPlatform**MemoryToHost**	Covert a controller memory address to a host memory address.
meiPlatform**Sleep**	Put the current thread to sleep for the number of milliseconds specified.
meiPlatform**ProcessId**	Return the process identification number of the current process.
meiPlatform**TimerCount**	Write to ticks the current timer count.
meiPlatform**TimerFrequency**	Write to frequency the timer frequency of the current platform.
meiPlatform**Trace**	Display printf(...)-style trace information
meiPlatform**TraceEol**	Set the end-of-line (eol) to be used by meiPlatformTrace(...).
meiPlatform**TraceFile**	Redirect trace output.
meiPlatform**TraceFunction**	Display the trace output.
meiPlatform**Word64Orient**	

Data Types

MEIPlatform**BoardType**
MEIPlatform**FileMode**
MEIPlatform**Info**
MEIPlatform**Message**

Constants

MEIPlatform**ControlCountMax**
MEIPlatform**InfoCHAR_MAX**

meiPlatformAlloc

Declaration

```
void meiPlatformAlloc(long   size)
```

Required Header: stdmei.h

Description

meiPlatformAlloc allocates system memory. meiPlatformAlloc will return NULL upon failure to allocate memory.

size	the number of bytes to allocate.

See Also

meiPlatformFree

meiPlatformAssertSet

Declaration

```
void meiPlatformAssertSet(void (MPI_DEF2 *func)
                          (const char *file, long line));
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00

Description

meiPlatformAssertSet sets an assertion handling function to be used by the MPI library.

When an assertion occurs, the filename and line number where that assertion happened will be passed to the assertion handling function. If no assertion handling function is set, information about the assertion will be reported to stdout and the C function, exit() will be called with an argument of 1.

An MPI library assertion should never occur. If it does, please contact MEI's technical support.

NOTE: meiPlatformAssertSet is only fully implemented for Windows operating systems.

platform	the handle to the controller's platform object. This should be obtained from meiControlPlatform(...).
firmware	the controller memory address to be converted.
\host	the location where the host memory address will be written.

See Also

meiControlPlatform

meiPlatformAtof

Declaration

```
double meiPlatformAtof(const char *ascii)
```

Required Header: stdmei.h

Description

meiPlatformAtof converts a numeric string to a double. This function returns the converted value as a double.

***ascii**	string to be converted

Returns
Converted the numeric text string ascii to a ***long*** and returned it.

See Also

meiPlatformAtol

meiPlatformAtol

Declaration

```
long meiPlatformAtol(const char *ascii)
```

Required Header: stdmei.h

Description

meiPlatformAtol converts a numeric string to a long. This function returns the converted value as a long.

***ascii**	string to be converted

Returns
Converted the numeric text string ascii to a ***long*** and returned it

See Also

meiPlatformAtof

meiPlatformExmpTempGet

Declaration

```
long meiPlatformExmpTempGet(MEIPlatform    platform,
                            long*          temp)
```

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

Description

meiPlatformExmpTempGet retrieves the internal temperature of the eXMP-SynqNet controller. This function should be used to poll the temperature to be sure it is within the allowable range. Hardware problems such as a broken fan, could cause the temperature of the eXMP to rise to a potentially unsafe level. When an unsafe temperature level has been detected, a shutdown routine should be executed by the motion application.

meiPlatformExmpTempInit(...) must be called once before polling with an meiPlatformExmpTempGet (...).

NOTE: This function is only supported on systems using an eXMP-SynqNet.

platform	the handle to the Platform object.
temp	the temperature (in Celsius) of the eXMP-SynqNet is returned in this variable.

See Also

meiPlatformExmpTempInit

eXMP-SynqNet Hardware

meiPlatformExmpTempInit

Declaration

```
long meiPlatformExmpTempInit(MEIPlatform    platform)
```

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

Description

meiPlatformExmpTempInit initializes the temperature reading routine on the eXMP-SynqNet controller. meiPlatformExmpTempInit must be called once before polling with an meiPlatformExmpTempGet(...).

NOTE: This function is only supported on systems using an eXMP-SynqNet.

platform	the handle to the Platform object.

See Also

meiPlatformExmpTempGet

eXMP-SynqNet Hardware

meiPlatformFileClose

Declaration

```
long meiPlatformFileClose(long   file)
```

Required Header: stdmei.h

Description

meiPlatformFileClose closes a file handle. meiPlatformFileClose is a platform independent replacement for the C function fclose().

file	the file handle to be closed.

See Also

meiPlatformFileOpen

meiPlatformFileOpen

Declaration

```
long meiPlatformFileOpen(const  char            *fileName,
                         MEIPlatformFileMode    mode)
```

Required Header: stdmei.h

Description

meiPlatformFileOpen opens a file handle. meiPlatformFileOpen is a platform independent replacement for the C function fopen().

fileName	the name of the file to open.
mode	the access mode used to open the file. Different MEIPlatformFileMode values may be or'ed together to produce the desired mode. For example: `MEIPlatformFileModeREAD | MEIPlatformFileModeBINARY)`

Returns
meiPlatformFileOpen returns the newly created file handle.

See Also

meiPlatformFileClose | meiPlatformFileRead | meiPlatformFileWrite | MEIPlatformFileMode

meiPlatformFileRead

Declaration

```
long meiPlatformFileRead(long   file,
                         char   *buffer,
                         long   byteCount)
```

Required Header: stdmei.h

Description

meiPlatformFileRead reads data from a file handle created by meiPlatformFileOpen. meiPlatformFileRead is a platform independent replacement for the C function fread().

file	the handle of the file from which data will be read.
buffer	the storage location where data from the file will be written to.
byteCount	the number of bytes to read from the file.

See Also

meiPlatformFileOpen | meiPlatformFileWrite

meiPlatformFileWrite

Declaration

```
long meiPlatformFileWrite(long          file,
                          const  char   *buffer,
                          long          byteCount)
```

Required Header: stdmei.h

Description

meiPlatformFileRead writes data to a file whose handle was created by meiPlatformFileOpen. meiPlatformFileWrite is a platform independent replacement for hte C function fwrite().

file	the handle of the file to which data will be written.
buffer	the location of the data to be written to the file.
byteCount	the number of bytes to be written to the file.

See Also

meiPlatformFileOpen | meiPlatformFileRead

meiPlatformFirmwareAddress16To32

Declaration

```
long meiPlatformFirmwareAddress16To32(MEIPlatform  platform,
                                      void         **firmwareAddress)
```

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

Description

meiPlatformFirmwareAddress16To32 takes a firmware address that is in the 16-bit space of the XMP and converts it to the 32-bit space. Conversion for an XMP is done by dividing the ***firmwareAddress*** by 2. No conversion is done for a ZMP.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress32To16

meiPlatformFirmwareAddress32To16

Declaration

```
long meiPlatformFirmwareAddress32To16(MEIPlatform  platform,
                                      void         **firmwareAddress,
                                      long         lowWord)
```

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

Description

meiPlatformFirmwareAddress32To16 takes a firmware address that is in the 32-bit space of the XMP and converts it to the 16-bit space.
Set ***lowWord*** to 1 to tell the function to return the 16-bit address of the lower 16-bit word.
Set ***lowWord*** to 0 to get the 16-bit address of the upper 16-bit word.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.
lowWord	**lowWord set to 1** tells the function to return the 16-bit address of the lower 16-bit word. **lowWord set to 0** gets the 16-bit address of the upper 16 bit word.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress16To32

meiPlatformFirmwareAddress32To64

Declaration

```
/* calculates base address of 64 bit entity given a pointer to
one of the 32 bit words within the 64 bit entity */
long meiPlatformFirmwareAddress32To64(MEIPlatform  platform,
                                      void         **firmwareAddress,
                                                    /* address of 32 bit entity */
                                      long         lowWord)
                                                   /* set to 1 if given address
                                                   is the low word */
```

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

Description

meiPlatformFirmwareAddress32To64 takes a pointer to a 32-bit word within a 64-bit entity and returns a pointer to the base address of the 64-bit entitiy. Set ***lowWord*** to 1 if the given address points to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.
lowWord	Set lowWord to 1 if the given address points to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress64To32

meiPlatformFirmwareAddress64To32

Declaration

```
long meiPlatformFirmwareAddress64To32(MEIPlatform  platform,
                                      void         **firmwareAddress,
                                      long         lowWord)
```

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

Description

meiPlatformFirmwareAddress64To32 takes a pointer to a 64-bit entity and returns a pointer to one of the 32-bit words within the 64-bit entity. Set ***lowWord*** to 1 to get a pointer to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.
lowWord	Set lowWord to 1 to get a pointer to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress32To64

meiPlatformFree

Declaration

```
long meiPlatformFree(void   *alloc,
                     long   size)
```

Required Header: stdmei.h

Description

meiPlatformFree frees system memory.

alloc	the address of the memory to free.
size	the number of bytes to free. This must match the size specified for meiPlatformAlloc when alloc was allocated.

See Also

meiPlatformAlloc

meiPlatformHostAddress32To64

Declaration

```
/* calculates base address of 64 bit entity given a pointer to
one of the 32 bit words within the 64 bit entity */
long meiPlatformHostAddress32To64(MEIPlatform  platform,
                                  void         **firmwareAddress,
                                               /* address of 32 bit entity */
                                  long         lowWord)
                                               /* set to 1 if given address
                                               is the low word */
```

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

Description

meiPlatformHostAddress32To64 takes a pointer to a 32-bit word within a 64-bit entity and returns a pointer to the base address of the 64-bit entitiy. Set ***lowWord*** to 1 if the given address points to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	a pointer to a valid firmware address from the host's perspective.
lowWord	set lowWord to 1 if the given address points to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformHostAddress64To32

meiPlatformHostAddress64To32

Declaration

```
long meiPlatformHostAddress64To32(MEIPlatform  platform,
                                  void         **firmwareAddress,
                                               /* address of
                                               generic 64entity */
                                  long         lowWord)
```

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

Description

meiPlatformHostAddress64To32 takes a pointer to a 64-bit entity and returns a pointer to one of the 32-bit words within the 64-bit entity. Set ***lowWord*** to 1 to get a pointer to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	a pointer to a valid firmware address from the host's perspective.
lowWord	set lowWord to 1 to get a pointer to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformHostAddress32To64

meiPlatformKey

Declaration

```
long meiPlatformKey(MPIWait wait)
```

Required Header: stdmei.h

Description

meiPlatformKey returns an input character (typically a keystroke) if an input character is available.

If an input charater is not available, *PlatformKey* waits ***wait*** milliseconds for an input character to become available.

NOTE:
meiPlatformKey is not fully implemented for all operating sytems. For example, in VentureCom's RTX Windows Extensions, a keystroke will be simulated after 10 seconds.

If "wait" is	*Then*
MPIWaitFOREVER (-1)	*PlatformKey* will wait for an input character forever
MPIWaitPOLL (0)	*PlatformKey* will return immediately
a value (not -1 or 0)	*PlatformKey* will wait for an input character for ***wait*** milliseconds

Return Values	
-1	if no input character was available
0	*PlatformKey* has read a non-zero character (typically a function key or other non-ASCII value), and **meiPlatformKey(...)** should be called AGAIN immediately to receive that non-zero character
a value (not -1 or 0) (an ASCII character)	(typically a keystroke) if an input character is available

See Also

meiSqNodeCreate | meiSqNodeValidate

meiPlatformMemoryGet64

Declaration

```
long meiPlatformMemoryGet64(MEIPlatform  platform,
                            void         *dst,
                            const void   *src)
```

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

Description

meiPlatformMemoryGet64 gets (reads) 8 bytes of platform memory (starting at address ***src***) to application memory (starting at address ***dst***).

This function should be used to get/read any 64-bit data from the controller. This function can take up to one foreground cycle to complete. This function will take a platform lock blocking all other tasks from accessing the controller.

platform	the handle to the controller's platform object.
***dst**	address of host data storage area. Storage MUST be two words (64 bits).
***src**	address of data (in host space) on the controller to be read.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	The board type is not an XMP or ZMP controller. 64-bit data is only supported on XMP and ZMP controllers.
MEIPlatformMessageCOPY64_FAILURE	The 64-bit value could not be read successfully.
MPIMessageFATAL_ERROR	The platform type does not exist.

Sample Code

```
...
/* axisPosition.actual is defined as MEIInt64 which is 64 bits */

if (returnValue == MPIMessageOK) {
  returnValue = meiPlatformMemoryGet64(axis->platform,
                                       &axis->axisPosition.actual,
                                       &axis->Axis->ActualPosition);
}

if (returnValue == MPIMessageOK) {
  *actual = (double)axis->axisPosition.actual;
}
...
```

See Also

meiPlatformMemorySet64

meiPlatformMemorySet64

Declaration

```
long meiPlatformMemorySet64(MEIPlatform  platform,
                            void         *dst,
                            const void   *src)
```

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

Description

meiPlatformMemorySet64 sets (writes) 8 bytes of application memory (starting at address ***src***) to platform memory (starting at address ***dst***).

This function should be used to set/write any 64-bit data to the controller. This function can take up to one foreground cycle to complete. This function will take a platform lock blocking all other tasks from accessing the controller.

platform	the handle to the controller's platform object.
dst	address of data (in host space) on the controller to be written.
src	address of host data storage area. Storage MUST be two words (64 bits).

Returns	
MPIMessageOK	
MPIMessageFATAL_ERROR	The platform type does not exist.

Sample Code

```
fMEIInt64 newTargetPos;


returnValue = meiPlatformMemoryGet64(axis->platform,
                                     &newTargetPos,
                                     &axis->Axis->TC.CommandPosition);

if (returnValue == MPIMessageOK) {
    returnValue = meiPlatformMemorySet64(axis->platform,
                                         &axis->Axis->TC.TargetPosition,
                                         &newTargetPos);
}
```

See Also

meiPlatformMemoryGet64

meiPlatformMemoryToFirmware

Declaration

```
long meiPlatformMemoryToFirmware(MEIPlatform   platform,
                                 const  void   *host,
                                 void          **firmware)
```

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

Description

meiPlatformMemoryToFirmware converts a host memory address to a controller memory address.

platform	the handle to the controller's platform object. This should be obtained from meiControlPlatform().
host	the host memory address to be converted.
firmware	the location where the controller's memory address will be written.

See Also

meiPlatformMemoryToHost | meiControlPlatform

meiPlatformMemoryToHost

Declaration

```
long meiPlatformMemoryToHost(MEIPlatform   platform,
                             const  void   *firmware,
                             void          **host)
```

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

Description

meiPlatformMemoryToHost coverts a controller memory address to a host memory address.

platform	the handle to the controller's platform object. This should be obtained from meiControlPlatform(...).
firmware	the controller memory address to be converted.
host	the location where the host memory address will be written.

See Also

meiPlatformMemoryToFirmware | meiControlPlatform

meiPlatformSleep

Declaration

```
void meiPlatformSleep(long   milliseconds)
```

Required Header: stdmei.h

Description

meiPlatformSleep puts the current thread to sleep for the number of ***milliseconds*** specified.

NOTE:
Different platforms have different time slice "quanta" (minimum sleep resolution) for threads. For example, Windows NT, 2000, and XP have a quanta of 10ms. For example, even if meiPlatformSleep (2) is specified, the actual sleep period will essentially be equivalent to meiPlatformSleep(10).

milliseconds	the number of milliseconds for which to put the current thread to sleep

Returns
Converted the numeric text string ascii to a ***long*** and returned it.

See Also

meiPlatformProcessId

Declaration

```
long meiPlatformProcessId(void)
```

Required Header: stdmei.h

Description

meiPlatformProcessId returns the process identification number of the current process.

See Also

meiPlatformTimerCount

Declaration

```
long meiPlatformTimerCount(long  *ticks)
```

Required Header: stdmei.h

Description

meiPlatformTimerCount reads the host CPU's timer value and writes it into the contents of a long pointed to by ***ticks***. The resolution of the platform timer can be determined with meiPlatformTimerFrequency(...).

***ticks**	a pointer to the timer value for the host CPU.

See Also

meiPlatformTimerFrequency

meiPlatformTimerFrequency

Declaration

```
long meiPlatformTimerFrequency(long  *frequency)
```

Required Header: stdmei.h

Description

meiPlatformTimerFrequency reads the host CPU's timer frequency and writes it into the contents of a long pointed to by ***frequency***. The platform timer value can be read with meiPlatformTimerCount(...).

***frequency**	a pointer to the timer frequency for the host CPU.

See Also

meiPlatformTimerCount

meiPlatformTrace

Declaration

```
long meiPlatformTrace(const char *format, ...)
```

Required Header: stdmei.h

Description

meiPlatformTrace displays **printf(...)**-style trace information. An *end-of-line* character will be appended to the output, newline by default.

Library modules call **meiTrace#(...)**, a macro which can be conditionally compiled to call **meiPlatformTrace(...)** (by defining the symbol MEI_TRACE when building the library).

Otherwise, calls to **meiTrace#(...)** are removed by the C preprocessor.

node	a handle of the SqNode object to delete in the reverse order to avoid memory leaks.

See Also

meiPlatformTraceEol

Declaration

```
char meiPlatformTraceEol(char eol)
```

Required Header: stdmei.h

Description

The **meiPlatformTraceEol** function sets the *end-of-line* (***eol***) character that will be used by meiPlatformTrace(...).

milliseconds	the number of milliseconds for which to put the current thread to sleep

Returns
The previous end-of-line character used by meiPlatformTrace(...)

See Also

meiPlatformTrace

meiPlatformTraceFile

Declaration

```
long meiPlatformTraceFile(const char *fileName)
```

Required Header: stdmei.h

Description

meiPlatformTraceFile redirects trace output to ***fileName***, after first closing any previously opened trace file. If no trace file has been explicitly opened, trace output will go to standard output.

Return Values	
MPIMessageOK	

See Also

meiPlatformTrace

meiPlatformTraceFunction

Declaration

```
MEITraceFunction meiPlatformTraceFunction(MEITraceFunction traceFunction)
```

Required Header: stdmei.h

Description

meiPlatformTraceFunction displays the trace output using ***traceFunction***, and replaces the internal function that was called by meiPlatformTrace(...) to display the trace output. Use *PlatformTraceFunction* to enable your application to take control of the display of trace output.

Return Values	
the previous *traceFunction*	if there is a previous function
NULL	if no ***traceFunction*** has been specified (the default trace function is used)

See Also

meiPlatformTrace

meiPlatformWord64Orient

Declaration

```
long meiPlatformWord64Orient(MEIPlatform  platform,
                             MEIInt64     *dst,
                             MEIInt64     *src)
```

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

This function is used to orient the two 32-bit words within a 64-bit entity. This function is needed when reading/writing 64-bit entites from/to the controller where the Get/Set64 functions cannot be used. The function is needed because the XMP is little endian and the ZMP is big endian. The low level MPI read/write functions already deal with byte swapping the bytes within a 32-bit word when reading/writing from/to the controller. The problem is that the generalized low level functions do not know which data is 64-bit data and cannot automatically swap the words. The Get/Set64 function do know that the data being written/read is 64 bits and automatically swaps the 32-bit words when needed.

This function will do nothing to the data for an XMP, but will swap the two 32-bit words for a ZMP. This function should be used when writing general code that could run on either an XMP or a ZMP controller.

platform	
dst	a pointer to 64-bit entity where oriented data will be written. Data should be on the host, not the controller.
src	a pointer to 64-bit entity to be oriented. Data should be on the host, not the controller.

Returns
MPIMessageOK

See Also

MEIPlatformBoardType

Definition

```
typedef enum {
    MEIPlatformBoardTypeUNKNOWN,
    MEIPlatformBoardTypeXMP,
    MEIPlatformBoardTypeZMP,
} MEIPlatformBoardType;
```

Description

MEIPlatformBoardType is the type of motion controller card that is being used.

MEIPlatformBoardTypeUNKNOWN	Board is not being recognized.
MEIPlatformBoardTypeXMP	An XMP Motion Controller board has been recognized.
MEIPlatformBoardTypeZMP	A ZMP Motion Controller board has been recognized.

See Also

MEIPlatformFileMode

Definition

```
typedef enum {
    MEIPlatformFileModeREAD,     /* default */
    MEIPlatformFileModeWRITE,
    MEIPlatformFileModeTEXT,     /* default */
    MEIPlatformFileModeBINARY,
    MEIPlatformFileModeTRUNC,
    MEIPlatformFileModeAPPEND,
} MEIPlatformFileMode;
```

Description

MEIPlatformFileMode is an enumeration that is used as an argument for methods that open files.

MEIPlatformFileModeREAD	Open a file for read access
MEIPlatformFileModeWRITE	Open a file for write access
MEIPlatformFileModeTEXT	Open a file as text format
MEIPlatformFileModeBINARY	Open a file as binary format
MEIPlatformFileModeTRUNC	Truncate existing file or create for reading and writing
MEIPlatformFileModeAPPEND	Open existing file for appending all writes

See Also

meiPlatformFileOpen

MEIPlatformInfo

Definition

```
typedef struct MEIPlatformInfo {
   char   OSInfo[MEIPlatformInfoCHAR_MAX];
   char   CPUInfo[MEIPlatformInfoCHAR_MAX];
   long   CPUMHz;
} MEIPlatformInfo;
```

Change History: Added in the 03.02.00

Description

MEIPlatformInfo is a structure that contains read only data about the host system characteristics. It contains Operating System, CPU type, and CPU clock speed.

OSInfo	a string containing the Operating System information.
CPUInfo	a string containing the CPU information.
CPUMHz	the CPU clock speed in MHz

See Also

MEIPlatformInfoCHAR_MAX

MEIPlatformMessage

Definition

```
typedef enum {

    MEIPlatformMessagePLATFORM_INVALID,
    MEIPlatformMessageDEVICE_INVALID,
    MEIPlatformMessageDEVICE_ERROR,
    MEIPlatformMessageDEVICE_MAP_ERROR,
    MEIPlatformMessageCOPY64_FAILURE,
} MEIPlatformMessage;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00.

Description

MEIPlatformMessage is an enumeration of SynqNet error messages that can be returned by the MPI library.

MEIPlatformMessagePLATFORM_INVALID

The platform object is not valid. This message code is returned by a platform method if the platform object handle is not valid. Most applications do not use the platform module. The MPI library uses the platform module internally. If an application needs a platform handle, use meiControlPlatform(...). Do NOT create your own platform object with meiPlatformCreate(...).

MEIPlatformMessageDEVICE_INVALID

The platform device driver is not valid. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the platform device handle is not valid. This message code comes from the lower level routines, meiPlatformInit(...) or meiPlatformDeviceClose(...). To correct the problem, make sure the device driver is installed and operational.

MEIPlatformMessageDEVICE_ERROR

The platform device failed. This message code is returned by the platform methods that fail to access a controller via a device driver. It occurs if the specified board type is not a member of the MEIPlatformBoardType enumeration. It also occurs if the device driver fails to read/write controller memory or there is an interrupt handling failure. To correct the problem, verify the platform has support for your controller and the device drive is installed and operational. Check for any resource conflicts (memory range, I/O port range, and interrupts) with other devices.

MEIPlatformMessageDEVICE_MAP_ERROR

The platform device memory mapping failed. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the controller memory could not be mapped to the operating system's memory space. To correct this problem, verify there are no memory resource conflicts. Also, make sure the host computer and operating system have enough free memory for the controller (XMP-Series requires 8 Mbytes).

MEIPlatformMessageCOPY64_FAILURE

The 64-bit read failed. This message is returned by meiPlatformMemoryGet64(…), mpiAxisCommandPositionGet(…), or mpiAxisActualPositionGet(…), if the 64-bit position data cannot be read atomically. Internally, the MPI uses an algorithm to construct the 64-bit position data via multiple 32-bit reads. If the 64-bit position value is not valid after multiple attempts, this error will be returned. If your application experiences this error message, if possible, use the equivalent 32-bit methods, mpiAxisCommandPositionGet32(…), mpiAxisActualPositionGet32(…), or contact MEI.

See Also

MEIPlatformControlCountMax

Definition

```
#define   MEIPlatformControlCountMax   (8) /* maximum number of controllers
the
                                           driver will currently support */
```

Change History: Added in the 03.04.00

Description

MEIPlatformControlCountMax defines the maximum number of controllers supported on a single system.

See Also

MEIPlatformInfoCHAR_MAX

Definition

```
#define MEIPlatformInfoCHAR_MAX (128)
```

Change History: Added in the 03.02.00

Description

MEIPlatformInfoCHAR_MAX defines the maximum number of characters for the MEIPlatformInfo strings.

See Also

MEIPlatformInfo | MEIPlatformBoardType

Probe Objects

Introduction

A Probe object manages a hardware logic block. The Probe hardware can latch and store up to 16 motor positions or node clock values within one controller sample period. The latches can be triggered by one of the configurable input sources (home, index, +/- limits, or one of the first 16 general purpose motor I/O).

The Probe is used in applications that need multiple data captures within one controller sample period. For applications that do not need multiple latches per sample, the Capture object is a much better solution. For example, a homing routine should use Capture (not Probe). Homing usually requires one or two precise position latches triggered by a home sensor and/or encoder index input to calibrate the position feedback to a physical location. The Probe is useful in high speed scanning applications, where the input can trigger at rates higher than the controller's sample rate. In this case, the Probe can be used with the Data Recorder to collect bursts of scan data. Later, the application can process the data.

Probe data is in raw position feedback counts or SynqNet node clock ticks. Typically, the high-speed Probe data and the sampled position data are collected with the Data Recorder. Later, the data is processed and the probed positions are calculated or interpolated from the time values.

There are two Probe data types: Position and Time. For the position data type, the lower 16 bits of the position counters are latched in the FPGA. This methodology works well for incremental quadrature encoders. For the time data type, the FPGA latches the clock value. The application must read the position from the controller's previous sample and the present sample, to interpolate the probe position. This methodology works well for cyclic feedback data that is digitally transmitted from the drive to the FPGA. Many drives have proprietary serial encoders that decode the encoder position and send the position information to the FPGA once per sample. In these cases, time-based probing is more accurate than position based probing.

For the position data type, the trigger source input and the feedback must be on the same motor.

For the time data type, the trigger source input and the clock must be on the same node. But, since SynqNet nodes are synchronized to the controller's clock, the position values can be interpolated across multiple nodes. Thus, a single probe input used with the Data Recorder can be used to collect/interpolate positions for several axes, across several nodes.

| Error Messages |

Methods

meiProbe**Info**
mpiProbe**Params**
mpiProbe**Status**
mpiProbe**Validate**

Data Types

MEIProbe**Address**
MPIProbe**Config**
MPIProbe**Data**
MPIProbe**Info**
MPIProbe**Message** / MEIProbe**Message**
MPIProbe**Number**
MPIProbe**Params**
MPIProbe**Source**
MPIProbe**Status**
MEIProbe**Trace**
MPIProbe**Type**

Constants

MPIProbe**MaxProbeRegisters**

mpiProbeConfigGet

Declaration

```
long mpiProbeConfigGet(MPIProbe         probe,
                       MPIProbeConfig   *config,
                       void             *external);
```

Required Header: stdmpi.h

Description

mpiProbeConfigGet reads a Probe's configuration and writes it into the structure pointed to by ***config***, and also into the implementation specific structure pointed to by ***external*** (if external is not NULL).

Remarks

external is reserved for future functionality and should be set to NULL.

probe	a handle to a Probe object.
***config**	a pointer to a Probe configuration structure.
***external**	a pointer to an implementation specific structure.

Return Values	
MPIMessageOK	
MPIProbeMessagePROBE_INVALID	

See Also

mpiProbeConfigSet

mpiProbeConfigSet

Declaration

```
long mpiProbeConfigSet(MPIProbe        probe,
                       MPIProbeConfig  *config,
                       void            *external);
```

Required Header: stdmpi.h

Description

mpiProbeConfigSet sets a Probe's configuration using data from the structure pointed to by ***config***, and also using data from the implementation specific structure pointed to by ***external*** (if external is not NULL).

Remarks

external is reserved for future functionality and should be set to NULL.

probe	a handle to a Probe object.
***config**	a pointer to a Probe configuration structure.
***external**	a pointer to an implementation specific structure. ***external*** is reserved for future functionality and should be set to NULL.

Return Values	
MPIMessageOK	
MPIProbeMessagePROBE_TYPE_INVALID	
MPIProbeMessagePROBE_INVALID	

See Also

mpiProbeConfigGet

mpiProbeControl

Declaration

```
MPIControl mpiProbeControl(MPIProbe     probe);
```

Required Header: stdmpi.h

Description

mpiProbeControl returns a handle to the Control object with which the Probe object is associated.

probe	a handle to a Probe object.

Return Values	
MPIControl	handle to a Control object.
MPIHandleVOID	if Probe is not valid

See Also

mpiProbeCreate | mpiControlCreate

mpiProbeCreate

Declaration

```
MPIProbe mpiProbeCreate(MPIProbe          control,
                        MPIProbeParams    *params)
```

Required Header: stdmpi.h

Description

mpiProbeCreate creates a Probe object identified by the params, which is associated with a control object. ProbeCreate is the equivalent of a C++ constructor.

The probe params structure specifies the probe type, a probe object identification number, and an index to a hardware probe engine. See MPIProbeParams for details.

control	a handle to a Probe object.
***params**	a pointer to a probe parameter's structure.

Return Values	
handle	handle to a Probe object.
MPIHandleVOID	if the object could not be created.
MPIProbeMessagePROBE_INVALID	The probe number specified was invalid. The likely cause of this is that the node you are referring to does not support the probe feature. Please refer to the Node FPGA Images: Features Table to see if your node supports probe.

See Also

mpiProbeDelete | mpiProbeValidate | MPIProbeParams

mpiProbeDelete

Declaration

```
long mpiProbeDelete(MPIProbe     probe);
```

Required Header: stdmpi.h

Description

mpiProbeDelete deletes a Probe object and invalidates its handle. ProbeDelete is the equivalent of a C++ constructor.

Remarks

All objects that are created must be deleted in the reverse order to avoid memory leaks.

probe	a handle to a Probe object.

Return Values	
MPIMessageOK	

See Also

mpiProbeCreate | mpiProbeValidate

meiProbeInfo

Declaration

```
long meiProbeInfo(MPIProbe      probe,
                  MEIProbeInfo  *info);
```

Required Header: stdmei.h

Description

meiProbeInfo reads a Probe's information and writes it into a structure pointed to by ***info***. The ***info*** structure contains read only data about the probe engine.

probe	a handle to a Probe object.
***info**	contains read only data about the probe engine.

Return Values	
MPIMessageOK	

See Also

mpiProbeParams

Declaration

```
long mpiProbeParams(MPIProbe        probe,
                    MPIProbeParams  *params);
```

Required Header: stdmpi.h

Description

mpiProbeParams reads the parameters of a Probe object and writes it to the structure pointed to by ***params***. The Probe parameters are set with mpiProbeCreate(...).

probe	a handle to a Probe object.
***params**	a pointer to a Probe parameter's structure.

Return Values	
MPIMessageOK	

See Also

mpiProbeStatus | mpiProbeCreate

mpiProbeStatus

Declaration

```
long mpiProbeStatus(MPIProbe          probe,
                    MPIProbeStatus    *status,
                    void              *external);
```

Required Header: stdmpi.h

Description

mpiProbeStatus reads status from the Probe and writes it into the structure pointed to by ***status*** and also writes it into the implementation-specific structure pointed to be ***external*** (if external is not NULL). The Probe status structure contains data from the probe engine, including digital I/O states and the latched probe values.

external is reserved for future functionality and should be set to NULL.

probe	a handle to a Probe object.
***status**	a pointer to a Probe status structure.
***external**	a pointer to an implementation specific structure. ***external*** is reserved for future functionality and should be set to NULL.

Return Values	
MPIMessageOK	

See Also

meiProbeInfo | mpiProbeParams

mpiProbeValidate

Declaration

```
long mpiProbeValidate(MPIProbe      probe);
```

Required Header: stdmpi.h

Description

mpiProbeValidate validates the Probe object and its handle. ProbeValidate should be called immediately after an object is created.

probe	a handle to a Probe object.

Return Values	
MPIMessageOK	

See Also

mpiProbeCreate | mpiProbeDelete

MEIProbeAddress

Definition

```
typedef struct MEIProbeAddress {
    long    *primaryPosition;
    long    *secondaryPosition;
    long    *time;
    long    *status;
    short   *data;
} MEIProbeAddress;
```

Description

MEIProbeAddress contains the controller addresses for the Probe data fields. The addresses in this structure are useful for configuring the Recorder to collect data during probing.

***primaryAddress**	a pointer to the address for the Motor's primary position feedback.
***secondaryPosition**	a pointer to the address for the Motor's secondary position feedback.
***time**	a pointer to the address for the feedback time value.
***status**	a pointer to the address for the Probe status
***data**	a pointer to the address for the Probe data.

See Also

MEIProbeInfo

MPIProbeConfig

Definition

```
typedef struct MPIProbeConfig {
    MPI_BOOL        enable;         /* TRUE/FALSE */
    MPIProbeSource  source;
    MPIProbeData    data;
    MPI_BOOL        inputFilter;    /* TRUE/FALSE */
} MPIProbeConfig;
```

Change History: Modified in the 03.03.00

Description

MPIProbeConfig specifies the Probe's configuration.

enable	Enables or disables the Probe triggering. A value of TRUE enables the Probe triggering, FALSE disables Probe triggering.
source	An enumerated Probe trigger source input. A Probe can be configured to trigger from one input source.
data	An enumerated Probe data type. A probe can be configured to collect position or time data from a motor's feedback.
inputFilter	Enables or disables the source input filtering. The hardware Probe engine has a 4 state digital filter to eliminate noise on the input signal. When enabled, the Probe will not trigger until the input signal is stable for 4 clock periods. For example, the MEI-RMB clock is 25Mhz. When the inputFilter is enabled, the input signal must transition to and remain at either a high or low state for 160 nanoseconds to trigger the Probe.

See Also

meiProbeConfigGet | meiProbeConfigSet

MPIProbeData

Definition

```
typedef enum  MPIProbeData {
    MPIProbeDataPOSITION_PRIMARY,
    MPIProbeDataPOSITION_SECONDARY,
    MPIProbeDataTIME,
} MPIProbeData;
```

Description

MPIProbeData is an enumeration of Probe data types. This specifies to the probe engine what data to collect when triggered.

MPIProbeDataPOSITION_PRIMARY	Position from the motor's primary feedback
MPIProbeDataPOSITION_SECONDARY	Position from the motor's secondary feedback
MPIProbeDataTIME	Clock value from the node. The clock value can be used to derive the position by interpolating between the positions from the previous sample and the present sample period and dividing by the difference between the probe clock value and the initial clock value.

See Also

meiProbeConfigGet | meiProbeConfigSet | MPIProbeConfig

MEIProbeInfo

Definition

```
typedef struct MEIProbeInfo {
    MEIProbeAddress    address;
} MEIProbeInfo;
```

Description

MEIProbeInfo contains read only information about the Probe.

address	a structure containing controller addresses for useful data during probing.

See Also

meiProbeInfo | meiProbeStatus

MPIProbeMessage / MEIProbeMessage

Definition: MPIProbeMessage

```
typedef enum {
    MPIProbeMessageNODE_INVALID,
    MPIProbeMessagePROBE_TYPE_INVALID,
    MPIProbeMessagePROBE_INVALID,
} MPIProbeMessage;
```

Change History: Modified in the 03.03.00

Description

MPIProbeMessage is an enumeration of SynqNet node error messages that can be returned by the MPI library.

MPIProbeMessageNODE_INVALID

The SynqNet node number is not available on the network. This message code is returned by MPI methods that fail a service command transaction due to the specified node number being greater than or equal to the total number of nodes discovered during network initialization. To correct this problem, check the discovered node count with meiSynqNetInfo(...). If the node count is not what you expected, check your network wiring, node condition, and re-initialize the network with mpiControlReset(...).

MPIProbeMessagePROBE_TYPE_INVALID

The Probe data type is not valid. This message is returned by mpiProbeConfigSet(...) if the specified Probe data type is not one of the enumerated values.

MPIProbeMessagePROBE_INVALID

The Probe is not valid. This message is returned by mpiProbeConfigGet(...) or mpiProbeConfigSet(...) if the specified Probe params or configurations are out of range. To correct this problem, check your Probe params and config structures.

Definition: MEIProbeMessage

```
typedef enum {
    MEIProbeMessageLAST = MPIProbeMessageLAST
} MEIProbeMessage;
```

Change History: Modified in the 03.03.00

Description

MEIProbeMessage is mainly a placeholder.

See Also

MPIProbeNumber

Definition

```
typedef union MPIProbeNumber {
    long     motor;
    long     node;
} MPIProbeNumber;
```

Description

MPIProbeNumber specifies the identification number for the Probe. The Probe type determines the identification number definition.

motor	An index to identify the motor that is associated with the Probe. Must be specified for MPIProbeTypeMOTOR Probe types.
node	An index to identify the node that is associated with the Probe. Must be specified for MPIProbeTypeIO Probe types.

See Also

MPIProbeType | MPIProbeParams

MPIProbeParams

Definition

```
typedef struct MPIProbeParams {
    MPIProbeType     type;
    MPIProbeNumber   number;
    long             probeIndex;
} MPIProbeParams;
```

Description

MPIProbeParams specifies the parameters for a Probe engine.

type	An enumerated Probe type. Determines the Probe number definition.
number	An union to identify the object number that is associated with the Probe. The Probe type defines the number.
probeIndex	An index to specify the Probe engine. The hardware may support one or more Probe engines per Motor or Node. The number of Probe engines supported by the hardware can be read with meiMotorInfo(...) and meiSqNodeInfo(...).

See Also

mpiProbeParams | mpiProbeCreate | MEIMotorInfo | MEISqNodeInfo

MPIProbeSource

Definition

```
typedef enum MPIProbeSource {
    MPIProbeSourceHOME,
    MPIProbeSourceINDEX,
    MPIProbeSourceLIMIT_HW_NEG,
    MPIProbeSourceLIMIT_HW_POS,
    MPIProbeSourceINDEX_SECONDARY,
    MPIProbeSourceMOTOR_IO_0,
    MPIProbeSourceMOTOR_IO_1,
    MPIProbeSourceMOTOR_IO_2,
    MPIProbeSourceMOTOR_IO_3,
    MPIProbeSourceMOTOR_IO_4,
    MPIProbeSourceMOTOR_IO_5,
    MPIProbeSourceMOTOR_IO_6,
    MPIProbeSourceMOTOR_IO_7,
    MPIProbeSourceMOTOR_IO_8,
    MPIProbeSourceMOTOR_IO_9,
    MPIProbeSourceMOTOR_IO_10,
    MPIProbeSourceMOTOR_IO_11,
    MPIProbeSourceMOTOR_IO_12,
    MPIProbeSourceMOTOR_IO_13,
    MPIProbeSourceMOTOR_IO_14,
    MPIProbeSourceMOTOR_IO_15,
} MPIProbeSource;
```

Description

MPIProbeSource is an enumeration of input trigger sources for a Probe.

MPIProbeSourceHOME	Home input in the dedicated I/O
MPIProbeSourceINDEX	Index input from the primary encoder in the dedicated I/O
MPIProbeSourceLIMIT_HW_NEG	Hardware Negative Limit input in the dedicated I/O
MPIProbeSourceLIMIT_HW_POS	Hardware Positive Limit input in the dedicated I/O
MPIProbeSourceINDEX_SECONDARY	Index input from the secondary encoder in the dedicated I/O

MPIProbeSourceMOTOR_IO_0	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_1	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_2	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_3	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_4	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_5	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_6	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_7	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_8	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_9	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_10	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_11	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_12	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_13	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_14	Bit number 0 in the Motor's configurable I/O
MPIProbeSourceMOTOR_IO_15	Bit number 0 in the Motor's configurable I/O

See Also

MPIProbeConfig

MPIProbeStatus

Definition

```
typedef struct MPIProbeStatus {
    long    maxRegisters;
    long    index;
    long    io;
    long    regs[MPIMaxProbeRegisters];
} MPIProbeStatus;
```

Description

MPIProbeStatus specifies the present condition of the Probe engine.

maxRegisters	The maximum number of Probe data registers that are available. This value is determined by the hardware Probe engine capability and the number of data registers that are initialized in the SynqNet packets.
index	The Probe engine index. Depending on the Probe type, the Probe is either associated with a motor or node object. Each Probe can have 1 or more hardware Probe engines. The probe engine is identified by its index.
io	The input state values for the Probe data. Each bit represents a triggered Probe data value in the regs array. If a bit is active, then a corresponding Probe data value was stored in the regs array. For example, if io = 0x3, then regs[0] and regs[1] have valid Probe data.
regs	An array of captured Probe data. The maximum number of data fields is specified by maxRegisters. Each element of the array that contains valid Probe data is flagged by a bit in the I/Ostatus.

See Also

mpiProbeStatus | mpiProbeParams | MPIProbeParams

MEIProbeTrace

Definition

```
typedef enum {
     MEIProbeTraceFIRST = MEITraceLAST << 1,
     MEIProbeTraceLAST  = MEIProbeTraceFIRST << 15
} MEIProbeTrace;
```

Description

MEIProbeTrace is an enumeration of probe object trace bits to enable debug tracing.

See Also

MPIProbeType

Definition

```
typedef enum MPIProbeType {
    MPIProbeTypeMOTOR,
    MPIProbeTypeIO,
} MPIProbeType;
```

Description

MPIProbeType is an enumeration of Probe types. This specifies whether the Probe engine is associated with a Motor or Node I/O. The available Probe type is dependent on the hardware Probe engine implementation.

MPIProbeTypeMOTOR	The Probe hardware engine is a component of the Motor object. For the Motor type, the Probe is identified by the Motor number.
MPIProbeTypeIO	The Probe hardware engine is a component of the Node I/O. For the I/O type, the Probe is identified by the Node number.

See Also

MPIProbeNumber | MPIProbeParams

MPIMaxProbeRegisters

Definition

```
#define MPIMaxProbeRegisters   (16)
```

Description

MPIMaxProbeRegisters defines the maximum limit for the probe registers for the Probe hardware.

See Also

Recorder Objects

Introduction

A **Recorder** object provides a mechanism to collect and buffer any data in the controller's memory. After a recorder is configured and started, the controller copies the data from the specified addresses to a local buffer every "N" samples. Later, the host can collect the data by polling or via interrupt-based events.

The controller supports up to 32 data recorders, which can collect data from up to a total of 32 addresses. The buffers can be dynamically allocated. A larger data recorder buffer may be required for higher sample rates, slow host computers, when running via client/server, or when a large number of data fields are being recorded.

A recorder can be started or stopped from the host application or from the controller by configuring a data recorder trigger. When the trigger conditions are met, the controller will automatically start or stop a data recorder. This is very useful for logging relevant variables during the period preceding a fault or error. Normally, the recorder stops collecting data when the buffer is full. It can also be configured to continuously collect data, overwriting the previous data until it is commanded to stop. This is useful for trapping a recent history of controller data.

When using data recorders, make sure to enable enough recorder objects and buffer memory with mpiControlConfigSet. Then, configure the recorders with mpiRecorderRecordConfig or mpiRecorderConfigSet, and start recording with mpiRecorderStart. Data can then be collected with mpiRecorderRecordGet.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

See Also:
Buffer Size

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiRecorder**Create**	Create Recorder object
mpiRecorder**Delete**	Delete Recorder object
mpiRecorder**Validate**	Validate Recorder object

Configuration and Information Methods

mpiRecorder**ConfigGet**	Get Recorder's configuration

mpiRecorder**ConfigSet**	Set Recorder's configuration
mpiRecorder**RecordConfig**	Configure type of data record that Recorder will capture
mpiRecorder**Status**	Get status of Recorder

Event Methods

mpiRecorder**EventNotifyGet**	Get event mask of events for which host notification has been requested
mpiRecorder**EventNotifySet**	Set event mask of events for which host notification will be requested
mpiRecorder**EventReset**	Reset the events specified in event mask that are generated by Recorder

Action Methods

mpiRecorder**RecordGet**	Get data records from Recorder
mpiRecorder**Start**	Start recording data records using Recorder
mpiRecorder**Stop**	Stop recording data records using Recorder

Memory Methods

mpiRecorder**Memory**	Get address to Recorder's memory
mpiRecorder**MemoryGet**	Copy data from Recorder memory to application memory
mpiRecorder**MemorySet**	Copy data from application memory to Recorder memory

Relational Methods

mpiRecorder**Control**	Return handle of Control object associated with Recorder
mpiRecorder**Number**	

Data Types

- MPIRecorder**Config** / MEIRecorder**Config**
- MPIRecorder**Message**
- MPIRecorder**Record** / MEIRecorder**Record**
- MEIRecorder**RecordAxis**
- MEIRecorder**RecordFilter**
- MPIRecorder**RecordPoint**
- MPIRecorder**RecordType** / MEIRecorder**RecordType**
- MPIRecorder**Status**
- MEIRecorder**Trace**
- MEIRecorder**Trigger**
- MEIRecorder**TriggerCondition**

MEIRecorder**TriggerIndex**

MEIRecorder**TriggerType**

MEIRecorder**TriggerUser**

Constants

MPIRecorder**ADDRESS_COUNT_MAX**

MEIRecorder**MAX_AXIS_RECORDS**

MEIRecorder**MAX_FILTER_RECORDS**

mpiRecorderRecordConfig

Declaration

```
long mpiRecorderRecordConfig(MPIRecorder           recorder,
                             MPIRecorderRecordType type,
                             long                  count,
                             void                  *handle)
```

Required Header: stdmpi.h

Description

mpiRecorderRecordConfig configures the type (***type***) of record that a Recorder (***recorder***) will capture.

If "type" is	Then
MPIRecorderRecordTypePOINT	***count*** data points will be recorded, and ***handle*** points to an array of ***count*** controller addresses
MEIRecorderRecordTypeAXIS	***count*** records of type MPIRecorderRecordAxis{} will be recorded, and ***handle*** points to an array of ***count*** Axis handles
MEIRecorderRecordTypeFILTER	***count*** records of type MPIRecorderRecordFilter{} will be recorded, and ***handle*** points to an array of ***count*** Filter handles

Return Values	
MPIMessageOK	

See Also

MPIRecorderRecordAxis | MPIRecorderRecordFilter

mpiRecorderConfigSet

Declaration

```
long mpiRecorderConfigSet(MPIRecorder       recorder,
                          MPIRecorderConfig *config,
                          void              *external)
```

Required Header: stdmpi.h

Description

mpiRecorderConfigSet sets a Recorder's (***recorder***) configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Recorder's configuration information in ***external*** is in addition to the Recorder's 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 **MEIRecorderConfig{}** or is NULL.

Return Values	
MPIMessageOK	

See Also

MEIRecorderConfig | MEIRecorderConfig | mpiRecorderConfigGet

mpiRecorderStart

Declaration

```
long mpiRecorderStart(MPIRecorder  recorder,
                      long         count); /* -1 => continuous,
                                              >0 => # of records */
```

Required Header: stdmpi.h

Description

mpiRecorderStart commands the controller to begin recording data records. Before starting a recorder, it must be configured with mpiRecorderRecordConfig(...) or mpiRecordConfigGet/Set(...).

recorder	a handle to a Recorder object
count	The number of data records to record. If (-1) is specified, the data recorder will continuously record until the buffer is full. If the host is retrieving data from the buffer faster than the controller can fill the buffer, the controller will continuously copy data to the buffer. The valid range is from -1 (continuous recording) to the maximum number of records available in the data recorder buffer.

Return Values	
MPIMessageOK	
MPIRecorderMessageSTARTED	

See Also

mpiRecorderRecordConfig | mpiRecorderStop | mpiRecorderConfigGet | mpiRecorderConfigSet | mpiControlConfigGet | mpiControlConfigSet

mpiRecorderRecordGet

Declaration

```
long mpiRecorderRecordGet(MPIRecorder       recorder,
                          long              countMax,
                          MPIRecorderRecord *record,
                          long              *count)
```

Required Header: stdmpi.h

Description

mpiRecorderRecordGet obtains a Recorder's (***recorder***) data records. The record type must have been configured previously, by a prior call to mpiRecorderRecordConfig(...).

RecorderRecordGet gets a maximum of ***countMax*** records and writes them into the location pointed to by ***record*** (the location must be large enough to hold them). RecorderRecordGet also writes the actual number of records that were obtained to the location pointed to by ***count***.

If the recorder data buffer is full and recording is enabled, recording will be temporarily disabled while either all or ***countMax*** records are obtained, whichever is less. Any records not obtained will be lost.

Return Values	
MPIMessageOK	

See Also

mpiRecorderRecordConfig

mpiRecorderCreate

Declaration

```
MPIRecorder mpiRecorderCreate(MPIControl control,
                              long       number);
```

Required Header: stdmpi.h

Description

mpiRecorderCreate creates a Recorder object identified by ***number***, which is associated with a control object. *RecorderCreate* is the equivalent of a C++ constructor.

The recorder number specifies which recorder to create. The valid range for the number parameter is -1 to the controller's ***recordCount*** (MPIControlConfig.recorderCount). Use a recorder number of -1 to specify the recorder number as the next available recorder.

See MPIControlConfig{.} for details. If the recorder is not enabled or is already in use (another process has called mpiRecorderCreate(...) with the same number parameter), mpiRecorderCreate(...) will return an invalid handle causing subsequent mpiRecorderValidate(...) calls to fail.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

control	a handle to a Control object.
number	An index to the controller's data recorder. If (-1) is specified, the next available recorder object handle will be returned. The valid range is from -1 (next available recorder) to the controller's recordCount - 1. When using (-1), make sure to delete the recorder object to free it for other applications. If the recorder object is not freed, it will not be accessible to another application until the controller is reset.

Return Values	
handle	to a Recorder object
MPIHandleVOID	if the Recorder object could not be created

See Also

mpiRecorderDelete | mpiRecorderValidate | MPIControlConfig | mpiControlConfigGet | mpiControlConfigSet

mpiRecorderDelete

Declaration

```
long mpiRecorderDelete(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderDelete deletes a Recorder object and invalidates its handle (***recorder***). *RecorderDelete* is the equivalent of a C++ destructor.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

control	a handle to a Control object.
number	An index to the controller's data recorder. If (-1) is specified, the next available recorder object handle will be returned. The valid range is from -1 (next available recorder) to the controller's recordCount - 1. When using (-1), make sure to delete the recorder object to free it for other applications. If the recorder object is not freed, it will not be accessible to another application until the controller is reset.

Return Values	
MPIMessageOK	

See Also

mpiRecorderCreate | mpiRecorderValidate

mpiRecorderValidate

Declaration

```
long mpiRecorderValidate(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderValidate validates the Recorder object and its handle. RecorderValidate should be called immediately after an object is created.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

recorder	a handle to a Recorder object

Return Values	
MPIMessageOK	
MPIRecorderMessageNOT_ENABLED	
MPIRecorderMessageNO_RECORDERS_AVAIL	

See Also

mpiRecorderCreate | mpiRecorderDelete

mpiRecorderConfigGet

Declaration

```
long mpiRecorderConfigGet(MPIRecorder       recorder,
                          MPIRecorderConfig *config,
                          void              *external)
```

Required Header: stdmpi.h

Description

mpiRecorderConfigGet gets a Recorder's (***recorder***) configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Recorder's configuration information in ***external*** is in addition to the Recorder's 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 **MEIRecorderConfig{}** or is NULL.

Return Values	
MPIMessageOK	

See Also

MPIRecorderConfig | MEIRecorderConfig | mpiRecorderConfigSet

mpiRecorderStatus

Declaration

```
long mpiRecorderStatus(MPIRecorder       recorder,
                       MPIRecorderStatus *status,
                       void              *external)
```

Required Header: stdmpi.h

Description

mpiRecorderStatus gets the status of the Recorder (***recorder***) and writes it into the structure pointed to by ***status***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

Remarks

external should always be set to NULL.

recorder	a handle to a Recorder object
***status**	a pointer to Recorder's status structure
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MPIRecorderStatus

mpiRecorderEventNotifyGet

Declaration

```
long mpiRecorderEventNotifyGet(MPIRecorder   recorder,
                               MPIEventMask  *eventMask,
                               void          *external)
```

Required Header: stdmpi.h

Description

mpiRecorderEventNotifyGet writes the event mask into the structure pointed to by **eventMask**, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL). (The event mask specifies the event type(s) generated by a Recorder (***recorder***), for which host notification has been requested.)

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

MEIEventNotifyData | MEIEventStatusInfo | mpiRecorderEventNotifySet

mpiRecorderEventNotifySet

Declaration

```
long mpiRecorderEventNotifySet(MPIRecorder   recorder,
                               MPIEventMask  eventMask,
                               void          *external)
```

Required Header: stdmpi.h

Description

mpiRecorderEventNotifySet requests host notification of the event(s) specified by ***eventMask*** and generated by a Recorder (***recorder***), and also generated by the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The events in ***external*** are in addition to the events in ***recorder***, i.e, the events in ***recorder*** and in ***external*** are not necessarily the same events. Note that ***recorder*** or ***external*** can be NULL (but not both NULL).

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

The mask of event types (generated by a Recorder object) consists of MEIEventMaskRECORDER_FULL and MEIEventMaskRECORDER_DONE.

To	Use "eventMask"
Enable host notification of all Recorder events	MPIEventMaskALL
Disable host notification of all Recorder events	MPIEventTypeNONE

Remarks

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

Return Values	
MPIMessageOK	

See Also

MEIEventMaskRECORDER | MEIEventNotifyData | MEIEventStatusInfo
mpiRecorderEventNotifyGet

mpiRecorderEventReset

Declaration

```
long mpiRecorderEventReset(MPIRecorder  recorder,
                           MPIEventMask eventMask)
```

Required Header: stdmpi.h

Description

mpiRecorderEventReset resets the event(s) specified in ***eventMask*** and generated by a Recorder (***recorder***). Your application should call *RecorderEventReset* only after one or more latchable events have occurred.

Return Values	
MPIMessageOK	

See Also

mpiControlEventReset | mpiMotionEventReset | mpiMotorEventReset | mpiSequenceEventReset | meiSynqNetEventReset | meiSqNodeEventReset | mpiAxisEventReset

Event Notification Methods

mpiRecorderStop

Declaration

```
long mpiRecorderStop(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderStop instructs a Recorder (***recorder***) to stop recording data records.

recorder	a handle to a Recorder object

Return Values	
MPIMessageOK	
MPIRecorderMessageSTOPPED	

Sample Code

```
/*
    Look for the warning code when the recorder is already stopped.
    This is usually not considered a bad thing (error).
*/
returnValue = mpiRecorderStop(recorder);
if(returnValue == MPIRecorderMessageSTOPPED)
{
returnValue = MPIMessageOK;
}
msgCHECK(returnValue);
```

See Also

mpiRecorderStart

mpiRecorderMemory

Declaration

```
long mpiRecorderMemory(MPIRecorder recorder,
                       void        **memory)
```

Required Header: stdmpi.h

Description

mpiRecorderMemory writes an address to the contents of ***memory***. An address can be used to access a Recorder's (***recorder***) memory. An address calculated from it can be passed as the ***src*** argument to mpiRecorderMemoryGet(...) and as the ***dst*** argument to mpiRecorderMemorySet(...).

Return Values	
MPIMessageOK	

See Also

mpiRecorderMemoryGet | mpiRecorderMemorySet

mpiRecorderMemoryGet

Declaration

```
long mpiRecorderMemoryGet(MPIRecorder   recorder,
                          void          *dst,
                          const void    *src,
                          long          count)
```

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

Description

mpiRecorderMemoryGet copies ***count*** bytes of a Recorder's (***recorder***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiRecorderMemory | mpiRecorderMemorySet

mpiRecorderMemorySet

Declaration

```
long mpiRecorderMemorySet(MPIRecorder   recorder,
                          void          *dst,
                          const void    *src,
                          long          count)
```

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

Description

mpiRecorderMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Recorder's (***recorder***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiRecorderMemory | mpiRecorderMemoryGet

mpiRecorderControl

Declaration

```
MPIControl mpiRecorderControl(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderControl returns a handle to the motion controller (Control object) that a Recorder (***recorder***) is associated with.

Return Values	
handle	to a Control object that a Recorder is associated with
MPIHandleVOID	if the Recorder object is invalid

See Also

mpiRecorderNumber

Declaration

```
long mpiRecorderNumber(MPIRecorder        recorder,
                       long               *number);
```

Required Header: stdmpi.h

Description

mpiRecorderNumber reads the index of a Recorder object and writes it into the contents of a long pointed to by ***number***. Each data recorder associated with a controller is indexed by a number (0, 1, 2, etc.).

recorder	a handle to a Recorder object.
***number**	a pointer to the index of a Recorder object.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessageHANDLE_INVALID	

See Also

mpiRecorderCreate

MPIRecorderConfig / MEIRecorderConfig

Definition: MPIRecorderConfig

```
typedef struct MPIRecorderConfig {
    long     period;     /* collect 1 record every `period` milliseconds */
    long     highCount;  /* >0 => record count to trigger high buffer */
    MPI_BOOL bufferWrap; /* TRUE/FALSE */

    long  addressCount;  /* number of data point addresses in address[] */
    void  *address[MPIRecorderADDRESS_COUNT_MAX];
} MPIRecorderConfig;
```

Change History: Modified in the 03.03.00

Description

MPIRecorderConfig structure specifies the configurations for a data recorder. It configures the sampling period, the buffer high event level, whether the buffering should wrap around, and a list of controller addresses to record.

period	The number of controller samples between successive data recorder acquisitions. A value of zero or one means the data recorder will acquire data every sample. A value of 2 means every other sample, 3 means every 3rd sample, etc. The valid range is 0 to 32767.
highCount	The number of buffered records until a MPIEventTypeRECORDER_HIGH status/event is generated. The valid range is 1 to the recorder buffer size configured by mpiControlConfigSet(...).
bufferWrap	Data recorder buffer rollover. A value of TRUE enables the buffer rollover, FALSE (default) disables the buffer rollover. When the bufferWrap is disabled, the controller will stop collecting data when the buffer is full. When bufferWrap is enabled, the controller will continuously collect data after the buffer is full, overwriting any previously collected data. The bufferWrap should be enabled if your application only wants to retrieve the last buffer of data after the data recorder is stopped. Most applications should set the bufferWrap to FALSE.
addressCount	The number of controller addresses in the address array.
***address**	An array of controller memory addresses to be recorded.

Definition: MEIRecorderConfig

```
typedef struct  MEIRecorderConfig {
    MEIRecorderTrigger  trigger[MEIRecorderTriggerIndexLAST];
} MEIRecorderConfig;
```

Description

MEIRecorderConfig specifies the configurations for the controller's data recorder triggers.

A data recorder can be started or stopped from the host application with mpiRecorderStart/Stop(...) or from the controller by configuring a data recorder trigger. When the trigger conditions are met, the controller will automatically start or stop a data recorder.

trigger	An array of data recorder trigger configuration structures.

See Also

mpiRecorderConfigGet | mpiRecorderConfigSet | mpiRecorderStart | mpiRecorderStop

MPIRecorderMessage

Definition

```
typedef enum {

    MPIRecorderMessageRECORDER_INVALID,
    MPIRecorderMessageSTARTED,
    MPIRecorderMessageSTOPPED,
    MPIRecorderMessageNOT_CONFIGURED,
    MPIRecorderMessageNO_RECORDERS_AVAIL,
    MPIRecorderMessageNOT_ENABLED,
    MPIRecorderMessageRUNNING,
    MPIRecorderMessageRECORD_COUNT_INVALID,
} MPIRecorderMessage;
```

Change History: Modified in the 03.04.00

Description

MPIRecorderMessage lists the error messages returned by the Recorder module.

MPIRecorderMessageRECORDER_INVALID

The recorder object is not valid. This message code is returned by a recorder method if the recorder object handle is not valid. This problem can be caused by a failed mpiRecorderCreate(...). To prevent this problem, check your recorder objects after creation by using mpiRecorderValidate(...).

MPIRecorderMessageSTARTED

The data recorder is already running. This message code is returned by mpiRecorderStart(...) if the data recorder has already been started. If this is a problem, call mpiRecorderStop(...) to stop the data recorder or wait for the recorder to collect the number of specified records and stop.

MPIRecorderMessageSTOPPED

The data recorder is not running. This message code is returned by mpiRecorderStop(...) if the data recorder has already been stopped. If this is a problem, call mpiRecorderStart(...) to start the data recorder.

MPIRecorderMessageNOT_CONFIGURED

The data recorder has not been configured. This message code is returned by mpiRecorderRecordGet(...) if the data address count has not been configured. To correct this problem, configure the data recorder with mpiRecorderConfigSet(...).

MPIRecorderMessageNO_RECORDERS_AVAIL

This message code is returned when a recorder number of -1 is specified and all enabled recorders have been previously reserved by mpiRecorderCreate(...) method calls. Reserved recorders are released by calling mpiRecorderDelete(...), however, it is possible for a fatal error to occur in your application in which case mpiRecorderDelete(...) may not be called. To override a reserved recorder number, explicitly specify the recorder number (i.e. a number other than -1) when calling mpiRecorderCreate(...).

MPIRecorderMessageNOT_ENABLED

An attempt was made to create a recorder that is not enabled on the controller. Recorder objects can be enabled on the controller by calling mpiControlConfigSet(...).

MPIRecorderMessageRUNNING

An attempt was made to call mpiRecorderConfigSet(...) while the recorder was running.

MPIRecorderMessageRECORD_COUNT_INVALID

This message code is returned by mpiRecorderStart(...) and mpiRecorderRecordConfig when the specified recorder object has no records allocated to it. The recorder record count is configured by calling mpiControlConfigSet(...) with MPIControlConfig.recordCount[recorderNumber] set to a non-zero value. The recorder itself is enabled by setting MPIControlConfig.recorderCount. See MPIControlConfig.

See Also

mpiRecorderCreate | mpiRecorderValidate

MPIRecorderRecord / MEIRecorderRecord

Definition: MPIRecorderRecord

```
typedef union {
   MPIRecorderRecordPoint  point[MPIRecorderADDRESS_COUNT_MAX];
} MPIRecorderRecord;
```

Description

point	An array of recorded values corresponding to the XMP addresses stored in MPIRecorderConfig.address[].

Definition: MEIRecorderRecord

```
typedef union {
    MEIRecorderRecordAxis     axis[MEIXmpMAX_Axes];
    MEIRecorderRecordFilter   filter[MEIXmpMAX_Filters];
    MPIRecorderRecord         dummy;  /* ensure proper sizing */
} MEIRecorderRecord;
```

Description

axis	An array of MEIRecorderRecordAxis records.
filter	An array of MEIRecorderRecordFilter records.
dummy	A dummy structure that ensures that MEIRecorderRecord has the proper size.

See Also

MPIRecorderConfig

MEIRecorderRecordAxis

Definition

```
typedef struct MEIRecorderRecordAxis {
    long      sample;     /* sample number */
    float     dac;        /* voltage */
    MEIInt64  actual;     /* actual position */
    MEIInt64  command;    /* command position */
} MEIRecorderRecordAxis;
```

Change History: Modified in the 03.04.00

Description

sample	The XMP sample number in which the following values were recorded.
dac	The output of the primary DAC of the motor associated with the axis.
actual	The actual position of the axis.
command	The command position of the axis.

See Also

MEIRecorderRecordFilter

Definition

```
typedef struct MEIRecorderRecordFilter {
    long   sample;          /* sample number */
    long   positionError;   /* position error */
    float  dac;             /* voltage */
} MEIRecorderRecordFilter;
```

Change History: Modified in the 03.04.00

Description

sample	The XMP sample number in which the following values were recorded
positionError	The position error value fed to the filter. Data is represented as a float.
dac	The output of the filter that gets sent to a motor's primary DAC.

See Also

MPIRecorderRecordPoint

Definition

```
typedef long MPIRecorderRecordPoint;
```

Description

MPIRecorderRecordPoint	represents one recorder record. This will correspond to the value of one XMP address.

See Also

MPIRecorderType / MEIRecorderType

Definition: MPIRecorderType

```
typedef enum {
    MPIRecorderRecordTypeINVALID,
    MPIRecorderRecordTypePOINT,
} MPIRecorderRecordType;
```

Description

MPIRecorderRecordTypeINVALID	an invalid record type.
MPIRecorderRecordTypePOINT	specifies to the data recorder that MPIRecorderRecordPoint records (copies of controller memory locations) are being recorded.

Definition: MEIRecorderType

```
typedef enum {
    MEIRecorderRecordTypeAXIS,
    MEIRecorderRecordTypeFILTER,
} MEIRecorderRecordType;
```

Description

Predefined types for setting up the type of data an MPIRecorder object will record. This is used by the mpiRecorderRecordConfig() method.

MEIRecorderRecordTypeAXIS	specifies to the data recorder that MEIRecorderRecordAxis records are being recorded.
MEIRecorderRecordTypeFILTER	specifies to the data recorder that MEIRecorderRecordFilter records are being recorded.

See Also

MPIRecorder | MEIRecorderRecordAxis | MEIRecorderRecordFilter | mpiRecorderRecordConfig

MPIRecorderStatus

Definition

```
typedef struct MPIRecorderStatus {
    MPI_BOOL  enabled;
    MPI_BOOL  full;
    long      recordCount;
    long      recordCountMax;
    MPI_BOOL  reserved;
} MPIRecorderStatus;
```

Change History: Modified in the 03.03.00

Description

enabled	If the recorder is enabled (recording) then enabled will equal a non-zero value (-1), otherwise enabled will equal 0.
full	If the recorder is full (the number of stored records >= MPIRecorderConfig. fullCount) then full will equal TRUE, otherwise full will equal FALSE.
recordCount	The number of stored records in the recorder.
recordCountMax	The maximum number of records the recorder can store.
reserved	TRUE if the recorder object has been previously created by the MPI and not yet deleted. A reserved recorder number cannot be reused until the recorder's reservation is canceled using mpiRecorderDelete(…) or the reservation is explicitly overwritten by specifying the recorder number (i.e. a number other than -1) when calling mpiRecorderCreate(...). If no recorder handle is available to call the mpiRecorderDelete(…) method, then calling meiControlRecorderCancel(…) may be used. In this case, use meiControlRecorderStatus(…) to verify that the recorder is not in use before canceling the recorder's reservation.

See Also

mpiRecorderStatus | mpiRecorderCreate | mpiRecorderDelete | meiControlRecorderCancel | meiControlRecorderStatus

recorderinuse.c

MEIRecorderTrace

Definition

```
typedef enum {

    MEIRecorderTraceRECORD_GET,
    MEIRecorderTraceSTATUS,
    MEIRecorderTraceOVERFLOW,
} MEIRecorderTrace;
```

Description

MEIRecorderTraceRECORD_GET	will display trace information when the data recorder retrieves records.
MEIRecorderTraceSTATUS	will display trace information when the MPI retrieves the data recorder status.
MEIRecorderTraceOVERFLOW	will display trace information when the data recorder overflows.

See Also

MEIRecorderTrigger

Definition

```
typedef struct MEIRecorderTrigger {
    MEIRecorderTriggerType     type;
    union {
        MEIRecorderTriggerUser user;
        } attributes;
} MEIRecorderTrigger;
```

Description

MEIRecorderTrigger specifies the configurations for a data recorder trigger.

type	The data recorder trigger type. See the MEIRecorderTriggerType enumeration.
user	The configurations for a user specified trigger type. See MEIRecorderTriggerUser.

See Also

MEIRecorderTrigger | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerCondtion

Definition

```
typedef enum MEIRecorderTriggerCondition {
    MEIRecorderTriggerConditionNONE,
    MEIRecorderTriggerConditionEQ,
    MEIRecorderTriggerConditionGREATER_THAN_OR_EQ,
    MEIRecorderTriggerConditionLESS_THAN_OR_EQ,
    MEIRecorderTriggerConditionNOT_EQ,
    MEIRecorderTriggerConditionCHANGE,
    MEIRecorderTriggerConditionMATCH = MEIRecorderTriggerConditionEQ,
    MEIRecorderTriggerConditionREPEAT = 0x80000000
} MEIRecorderTriggerCondition;
```

Change History: Modified in the 03.03.00

Description

MEIRecorderTriggerCondtion is an enumeration of a data recorder's trigger conditions. The mask and pattern fields referred to are from the MEIRecorderTriggerUser structure.

All trigger conditions (except MEIRecorderTriggerConditionCHANGE) are "single shot." This means that they will only trigger one time and will not continue to trigger even if the conditions are met. RecorderTriggers can be made to repeat by ORing in MEIRecorderTriggerConditionREPEAT with any of the other RecorderTriggerConditions.

MEIRecorderTriggerConditionCHANGE is not a single shot. When it triggers, it will set the pattern value equal to the value at the specified address and rearm.

NOTE: There are 2 Recorder triggers. Trigger 0 will start the recorder, and Trigger 1 will stop the recorder. Only one of the triggers will be evaluated at a time. The trigger that is evaluated depends on the state of the Recorder. If the Recorder is not active, then Trigger 0 (Start) will be evaluated. If the Recorder is active, then Trigger 1 (Stop) will be evaluated.

MEIRecorderTriggerTriggerConditionNONE	Disables the trigger.
MEIRecorderTriggerTriggerConditionEQ	Triggers when the value at the specified address ANDed with the mask is equal to the pattern.
MEIRecorderTriggerTriggerConditionGREATER_THAN_OR_EQ	Triggers when the value at the specified address ANDed with the mask is greater than or equal to the pattern.
MEIRecorderTriggerTriggerConditionLESS_THAN_OR_EQ	Triggers when the value at the specified address ANDed with the mask is less than or equal to the pattern.
MEIRecorderTriggerTriggerConditionNOT_EQ	Triggers when the value at the specified address ANDed with the mask is not equal to the pattern.
MEIRecorderTriggerTriggerCHANGE	Triggers when the value at the specified address ANDed with the **mask** changes. The **pattern** field is only used to set the initial bit pattern used to determine if a change occurs.
MEIRecorderTriggerTriggerMATCH	Triggers when the value at the specified address ANDed with the **mask** is equal to the specified **pattern**.
MEIRecorderTriggerTriggerREPEAT	Causes the trigger to remain active after it has triggered (not a single shot).

See Also

MEIRecorderTriggerUser | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerIndex

Definition

```
typedef enum MEIRecorderTriggerIndex {
    MEIRecorderTriggerIndexSTART,
    MEIRecorderTriggerIndexSTOP,
} MEIRecorderTriggerIndex;
```

Description

MEIRecorderTriggerIndex is an enumeration of indices to a data recorder's trigger logic.

MEIRecorderTriggerIndexSTART	Index to a data recorder's start trigger.
MEIRecorderTriggerIndexSTOP	Index to a data recorder's stop trigger.

See Also

MEIRecorderConfig | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerType

Definition

```
typedef enum MEIRecorderTriggerType {
    MEIRecorderTriggerTypeDISABLED,
    MEIRecorderTriggerTypeUSER,
} MEIRecorderTriggerType;
```

Description

MEIRecorderTriggerType is an enumeration of a data recorder's trigger logic types.

MEIRecorderTriggerTypeDISABLED	The data recorder trigger is not enabled.
MEIRecorderTriggerTypeUSER	The data recorder trigger is user configurable. See the MEIRecorderTriggerUser{.} structure for details.

See Also

MEIRecorderTrigger | MEIRecorderTriggerUser | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerUser

Definition

```
typedef struct  MEIRecorderTriggerUser {
    MEIRecorderTriggerCondition    condition;
    long                           *addr;
    unsigned long                  mask;
    unsigned long                  pattern;
    unsigned long                  count;
} MEIRecorderTriggerUser;
```

Description

MEIRecorderTriggerUser specifies the configurations for a user specified data recorder trigger.

condition	The logic that determines how to evaluate the addr, mask, and pattern. See the MEIRecorderTriggerCondition enumeration.
***addr**	A pointer to a controller address.
mask	A bit mask ANDed with the value at the controller address.
pattern	A bit pattern compared to the masked value at the controller address.
count	The number of records to collect when the recorder is triggered. This is valid for both start and stop triggers. The valid range is 0 to the recorder buffer size configured by mpiControlConfigSet(...). When used for the start trigger, the valid values range from -1 (continuous recording) to the maximum number of records available in the data recorder buffer. When used for the stop trigger, *count* records will be collected after the trigger has triggered.

See Also

MEIRecorderTrigger | mpiRecorderConfigGet | mpiRecorderConfigSet

MPIRecorderADDRESS_COUNT_MAX

Definition

```
#define  MPIRecorderADDRESS_COUNT_MAX  (32)
```

Description

MPIRecorderADDRESS_COUNT_MAX defines the maximum number of addresses the Recorder object supports.

See Also

MPIRecorderConfig

MEIRecorderMAX_AXIS_RECORDS

Definition

```
#define  MEIRecorderMAX_AXIS_RECORDS  (5)
```

Description

MEIRecorderMAX_AXIS_RECORDS defines the maximum number of MEIRecorderRecordAxis records that can be recorded by a single recorder at any one time.

See Also

MEIRecorderRecordAxis | mpiRecorderRecordConfig

MEIRecorderMAX_FILTER_RECORDS

Definition

```
#define  MEIRecorderMAX_FILTER_RECORDS   (10)
```

Description

MEIRecorderMAX_FILTER_RECORDS defines the maximum number of MEIRecorderRecordFilter records that can be recorded by a single recorder at any one time.

See Also

MEIRecorderRecordFilter | mpiRecorderRecordConfig

Recorder Buffer Size

The Data Recorder buffer size can be dynamically allocated. The MPIControlConfig{...} structure has a new element, called recordCount. This element allows the application to change the size of the recorder object's data buffer using the mpiControlConfigGet/Set(...) methods. The Record buffer size (the default is 3064 records) is defined within the MEIXmpDefaultEnabled_Records structure (*xmp.h*). Each record is the size of one memory word. Using a larger data buffer size can improve the performance of MotionScope running on a slow host or running in Client/Server mode over a congested network.

A new method, meiControlExtMemAvail(...), has been added which will return the size of external memory available for allocation. This value can be added to the current recordCount to expand the record buffer to the maximum possible size.

For more information, see the Dynamic Allocation of External Memory Buffers.

Return to Recorder Object's page

Sequence Objects

Introduction

A **Sequence** object manages a set of Commands. The sequence is constructed on the host from a list of commands, then downloaded and executed in the controller. Typically, applications only use Sequences for very small or simple autonomous tasks that require execution in the controller. Due to their embedded execution, debugging can be difficult. It is best to use the host application to execute MPI methods directly for optimum flexibility and performance.

If you are considering using a program Sequencer or Command objects, please contact your support engineer. We recommend that you do NOT implement complex Sequences on your own.

Commands are implemented using MPICommand objects. Information about the different types of commands can be found on MPICommandType and MPICommandParams. Sample applications for using sequencers can be found in the Sample Applications section. Search for application names starting with ***seq***. Seqkill.c is a good place to start.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiSequence**Create**	Create Sequence object
mpiSequence**Delete**	Delete Sequence object
mpiSequence**Validate**	Validate Sequence object

Configuration and Information Methods

mpiSequence**ConfigGet**	Get sequence config
mpiSequence**ConfigSet**	Set sequence config
mpiSequence**FlashConfigGet**	Get sequence flash config
mpiSequence**FlashConfigSet**	Set sequence flash config
mpiSequence**PageSize**	Set pageSize to number of command slots used by sequence
mpiSequence**Status**	Return sequence status

Event Methods

mpiSequence**EventNotifyGet**	Select an event mask for host notification of events
mpiSequence**EventNotifySet**	Enable host notification of sequence events
mpiSequence**EventReset**	Reset sequence events

Action Methods

meiSequence**Compile**	

mpiSequence**Load**	Load sequence commands into firmware
mpiSequence**Resume**	Resume execution of sequence
mpiSequence**Start**	Start execution of sequence
mpiSequence**Step**	Execute count steps of a stopped sequence
mpiSequence**Stop**	Stop sequence

Memory Methods

mpiSequence**Memory**	Set address used to access sequence memory
mpiSequence**MemoryGet**	Get bytes of sequence memory and put into application memory
mpiSequence**MemorySet**	Put (set) bytes of application memory into sequence memory

Relational Methods

mpiSequence**Control**	Get handle to Control
mpiSequence**Number**	Get index number of sequence

Command Methods

mpiSequence**Command**	Return handle to indexed command of sequence
mpiSequence**CommandAppend**	Append command to sequence
mpiSequence**CommandCount**	Count the number of commands in sequence
mpiSequence**CommandFirst**	Return handle to first command in sequence
mpiSequence**CommandIndex**	Return the index of a command in sequence
mpiSequence**CommandInsert**	Insert command into sequence
mpiSequence**CommandLast**	Return handle of last command in sequence
mpiSequence**CommandListGet**	Get list of commands in sequence
mpiSequence**CommandListSet**	Set list of commands in sequence
mpiSequence**CommandNext**	Get handle to next command in list
mpiSequence**CommandPrevious**	Get handle to previous command in list
mpiSequence**CommandRemove**	Remove command from list

Data Types

MPISequence**Config** / MEISequence**Config**

MPISequence**Message**

MPISequence**State**

MPISequence**Status**

MEISequence**Trace**

See Also

MPICommand

MPICommand**Type**

MPICommand**Params**

seqKill.c (sample application)

mpiSequenceCreate

Declaration

```
MPISequence mpiSequenceCreate(MPIControl control,
                              long       number,
                              long       pageSize)
```

Required Header: stdmpi.h

Description

mpiSequenceCreate creates a Sequence object associated with the program sequencer identified by ***number*** located on motion controller (control). SequenceCreate is the equivalent of a C++ constructor.

If	Then
number is **-1**	*SequenceCreate* selects the next unused program sequencer. If this is the first use of the program sequencer, then SequenceCreate will attempt to allocate pageSize firmware command slots.
pageSize is **-1**	*SequenceCreate* will allocate all remaining firmware command slots, which may prevent any more Sequence objects from being created.

Return Values	
handle	to a Sequence object
MPIHandleVOID	if the object could not be created

See Also

mpiSequenceDelete | mpiSequenceValidate

mpiSequenceDelete

Declaration

```
long mpiSequenceDelete(MPISequence  sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceDelete deletes a Sequence object and invalidates its handle (***sequence***). *SequenceDelete* is the equivalent of a C++ destructor.

All Command objects in a Sequence are deleted when the Sequence object is deleted.

sequence	a handle to the Sequence object.

Return Values	
MPIMessageOK	

See Also

mpiSequenceCreate | mpiSequenceValidate

mpiSequenceValidate

Declaration

```
long mpiSequenceValidate(MPISequence sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceValidate validates the Sequence object and its handle (***sequence***).

Return Values	
MPIMessageOK	

See Also

mpiSequenceCreate | mpiSequenceDelete

mpiSequenceConfigGet

Declaration

```
long mpiSequenceConfigGet(MPISequence          sequence,
                          MPISequenceConfig    *config,
                          void                 *external)
```

Required Header: stdmpi.h

Description

mpiSequenceConfigGet gets the configuration of a Sequence object (***sequence***) and 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 Sequence's configuration information in ***external*** is in addition to the Sequence's 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 MEISequenceConfig{} or is NULL.

Return Values	
MPIMessageOK	

See Also

mpiSequenceConfigSet | MEISequenceConfig

mpiSequenceConfigSet

Declaration

```
long mpiSequenceConfigSet(MPISequence          sequence,
                          MPISequenceConfig    *config,
                          void                 *external)
```

Required Header: stdmpi.h

Description

mpiSequenceConfigSet sets the configuration of a Sequence (***sequence***) 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 Sequence's configuration information in ***external*** is in addition to the Sequence's 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 MEISequenceConfig{} or is NULL.

Return Values	
MPIMessageOK	

See Also

mpiSequenceConfigGet | MEISequenceConfig

mpiSequenceFlashConfigGet

Declaration

```
long mpiSequenceFlashConfigGet(MPISequence       sequence,
                               void              *flash,
                               MPISequenceConfig *config,
                               void              *external)
```

Required Header: stdmpi.h

Description

mpiSequenceFlashConfigGet gets a Sequence's (***sequence***) flash configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Sequence's flash configuration information in ***external*** is in addition to the Sequence'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 MEISequenceConfig{} 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

mpiSequenceFlashConfigSet

mpiSequenceFlashConfigSet

Declaration

```
long mpiSequenceFlashConfigSet(MPISequence        sequence,
                               void               *flash,
                               MPISequenceConfig  *config,
                               void               *external)
```

Required Header: stdmpi.h

Description

mpiSequenceFlashConfigSet sets a Sequence's (***sequence***) flash configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Sequence's flash configuration information in ***external*** is in addition to the Sequence'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 MEISequenceConfig{} 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

MEISequenceConfig | mpiSequenceFlashConfigGet

mpiSequencePageSize

Declaration

```
long mpiSequencePageSize(MPISequence sequence,
                         long        *pageSize)
```

Required Header: stdmpi.h

Description

mpiSequencePageSize writes the *number* of command slots that are available to a Sequence (***sequence***, on its associated motion controller) to the contents of ***pageSize***.

Return Values	
MPIMessageOK	

See Also

mpiSequenceStatus

Declaration

```
long mpiSequenceStatus(MPISequence       sequence,
                       MPISequenceStatus *status,
                       void              *external)
```

Required Header: stdmpi.h

Description

mpiSequenceStatus returns the status of a Sequence (***sequence***), and writes it into the structure pointed to by ***status***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

Remarks

external should always be set to NULL.

sequence	a handle to a Sequence object
***status**	a pointer to Sequence's status structure
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MPISequenceStatus

mpiSequenceEventNotifyGet

Declaration

```
long mpiSequenceEventNotifyGet(MPISequence   sequence,
                               MPIEventMask  *eventMask,
                               void          *external)
```

Required Header: stdmpi.h

Description

mpiSequenceEventNotifyGet writes an event mask [that specifies the event types (generated by the Sequence ***sequence***, for which host notification has been requested] to the structure pointed to by ***eventMask***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

Remarks

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

Return Values	
MPIMessageOK	

See Also

MEIEventMask | mpiSequenceEventNotifySet

mpiSequenceEventNotifySet

Declaration

```
long mpiSequenceEventNotifySet(MPISequence   sequence,
                               MPIEventMask  eventMask,
                               void          *external)
```

Required Header: stdmpi.h

Description

mpiSequenceEventNotifySet requests host notification of the event(s) specified by ***eventMask*** and generated by a Sequence (***sequence***), and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

The mask of event types generated by a Sequence object consists of MPIEventMaskEXTERNAL. When a Sequence issues a Command of type MPICommandTypeEVENT, an event of type MPIEventTypeEXTERNAL is generated. The only event generated by a Sequence is MPIEventTypeEXTERNAL, which is generated when a Sequence issues a Command of type MPICommandTypeEVENT.

Remarks

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

To	Use "eventMask"
Disable host notification of all Sequence events	MPIEventTypeNONE
Enable host notification of all Sequence events	MPIEventMaskALL

Return Values	
MPIMessageOK	

See Also

MPIEventMaskEXTERNAL | MEIEventMask | mpiSequenceEventNotifyGet

mpiSequenceEventReset

Declaration

```
long mpiSequenceEventReset(MPISequence  sequence,
                           MPIEventMask eventMask)
```

Required Header: stdmpi.h

Description

mpiSequenceEventReset resets the event(s) that are specified in ***eventMask*** and generated by a Sequence (***sequence***). Your application should not call SequenceEventReset *until* one or more latchable events have occurred.

Return Values	
MPIMessageOK	

See Also

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

Event Notification Methods

meiSequenceCompile

Declaration

```
long meiSequenceCompile(MPISequence sequence)
```

Required Header: stdmei.h

Description

meiSequenceCompile "compiles" a ***sequence*** object by reading its list of Command objects and then creating an equivalent list of XMP commands.

sequence	a handle to the Sequence object.

Return Values	
MPIMessageOK	

Compiling Program Sequencer Commands

An MPICommand will "compile" into one or more MEIXmpCommand{}s, each of which takes up a slot in the MEIXmpCommandBuffer{}. In general, an MPICommand will compile into a single MEIXmpCommand{}, but an MPICommand of type MPICommandTypeMOTION [with a motionCommand of MPICommandMotionSTART (i.e. mpiMotionStart(...))] will require several MEIXmpCommand{}s.

How many sequencer commands an MPI sequence command compiles to depends on the number of axes and number of positions in the move. The next table shows how many xmp sequencer commands it takes to do the equivalent of an mpiMotionStart(...).

Number of Sequencer Commands to be equivalent to mpiMotionStart(...)

Number of required sequencer commands	To do this:
axisCount +	One MEIXmpCommand{} per axis to write the axis number to MEIXmpLinkBuffer{}.MSLink[].Axis[].AxisNumber
1+	1 + One MEIXmpCommand{} to write axisCount to MEIXmpLinkBuffer{}.MSLink[].Axes
1+	One MEIXmpCommand{} to write the MEIXmpMotionType{} to MS[].Mode.
(((axisCount*pointCount) + 3) / 4) +	One MEIXmpCommand{} for every four MEIXmpPoint{}s written to PointBuffer.Point[]

axisCount +	One MEIXmpCommand{} per axis to load the MEIXmpPoint (s)
1	One MEIXmpCommand{} to start the motion

See Also

mpiSequenceLoad

Declaration

```
long mpiSequenceLoad(MPISequence sequence,
                     MPICommand  command,
                     MPI_BOOL    start);
```

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

Description

mpiSequenceLoad loads the firmware command slots of a Sequence (***sequence***) as necessary, starting with the Command (***command***).

SequenceLoad is intended to be called initially by mpiSequenceStart(...) and called thereafter by mpiEventMgrService(...) (in response to reception of an *internal page fault event notification* from the firmware). Except when you are debugging a sequence via mpiSequenceStep(...), your application should never need to directly call SequenceLoad.

If	Then
command is MPIHandleVOID	*SequenceLoad* loads Commands starting with the first Command of the Sequence
start is not FALSE	*SequenceLoad* starts the sequence after the commands are loaded

Return Values	
MPIMessageOK	

See Also

mpiSequenceStart | mpiEventMgrService | mpiSequenceStep

mpiSequenceResume

Declaration

```
long mpiSequenceResume(MPISequence sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceResume resumes a Sequence (***sequence***) from the point where the Sequence has stopped (if execution has been stopped).

Return Values	
MPIMessageOK	

See Also

mpiSequenceStart

Declaration

```
long mpiSequenceStart(MPISequence sequence,
                      MPICommand  command)
```

Required Header: stdmpi.h

Description

mpiSequenceStart begins the execution of a Sequence (***sequence***), starting with the Command (***command***). If ***command*** is MPIHandleVOID, execution starts with the first command of the Sequence.

Return Values	
MPIMessageOK	

See Also

mpiSequenceStop

mpiSequenceStep

Declaration

```
long mpiSequenceStep(MPISequence sequence,
                     long        count)
```

Required Header: stdmpi.h

Description

mpiSequenceStep executes ***count*** steps (Commands) of a stopped Sequence (***sequence***). After executing the Commands, the Sequence will be in the MPISequenceStateSTOPPED state.

Return Values	
MPIMessageOK	

See Also

mpiSequenceStop

Declaration

```
long mpiSequenceStop(MPISequence sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceStop stops a Sequence (***sequence***), if execution has been started. A stopped Sequence can be resumed from the point where it has stopped.

Return Values	
MPIMessageOK	

See Also

mpiSequenceStart

mpiSequenceMemory

Declaration

```
long mpiSequenceMemory(MPISequence sequence,
                       void        **memory)
```

Required Header: stdmpi.h

Description

mpiSequenceMemory writes an address [used to access a Sequence's (sequence) memory] to the contents of ***memory***. This address (or an address calculated from it) is passed as the ***src*** argument to mpiSequenceMemoryGet(...) and as the ***dst*** argument to mpiSequenceMemorySet(...).

Return Values	
MPIMessageOK	

See Also

mpiSequenceMemoryGet | mpiSequenceMemorySet

mpiSequenceMemoryGet

Declaration

```
long mpiSequenceMemoryGet(MPISequence sequence,
                          void        *dst,
                          const void  *src,
                          long        count);
```

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

Description

mpiSequenceMemoryGet copies ***count*** bytes of a Sequence's (***sequence***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiSequenceMemorySet | mpiSequenceMemory

mpiSequenceMemorySet

Declaration

```
long mpiSequenceMemorySet(MPISequence sequence,
                          void        *dst,
                          const void  *src,
                          long        count);
```

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

Description

mpiSequenceMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Sequence's (***sequence***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiSequenceMemory | mpiSequenceMemoryGet

mpiSequenceControl

Declaration

```
MPIControl mpiSequenceControl(MPISequence sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceControl returns a handle to the Control object with which the Sequence object is associated.

sequence	a handle to the Sequence object.

Return Values	
MPIControl	a handle to the Sequence object
MPIHandleVOID	if ***sequence*** is invalid

See Also

mpiSequenceCreate | mpiControlCreate

mpiSequenceNumber

Declaration

```
long mpiSequenceNumber(MPISequence sequence,
                       long        *number)
```

Required Header: stdmpi.h

Description

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

Return Values	
MPIMessageOK	

See Also

mpiSequenceCommand

Declaration

```
MPICommand mpiSequenceCommand(MPISequence sequence,
                              long        index)
```

Required Header: stdmpi.h

Description

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

sequence	a handle to the Sequence object.
index	a position in the list.

Return Values	
handle	to the ***index***th Command of a Sequence (***sequence***)
MPIHandleVOID	if ***sequence*** is invalid if ***index*** is less than 0 if ***index*** is greater than or equal to **mpiSequenceCount(sequence)**
MPIMessageARG_INVALID	if ***index*** is a negative number.
MEIListMessageELEMENT_NOT_FOUND	if ***index*** is greater than or equal to the number of elements in the list.
MPIMessageHANDLE_INVALID	if ***sequence*** is an invalid handle.

See Also

mpiSequenceCommandAppend

Declaration

```
long mpiSequenceCommandAppend(MPISequence sequence,
                              MPICommand  command)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandAppend appends a Command (***command***) to a Sequence (***sequence***).

sequence	a handle to the Sequence object.
command	a handle to a Command object.

Return Values	
MPIMessageOK	
MPIMessageHANDLE_INVALID	
MPIMessageNO_MEMORY	

See Also

mpiSequenceCommandCount

Declaration

```
long mpiSequenceCommandCount(MPISequence sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandCount returns the number of elements on the list.

sequence	a handle to the Sequence object.

Return Values	
number of Commands	in a Sequence (***sequence***)
-1	if ***sequence*** is invalid
0	if ***sequence*** is empty

See Also

mpiSequenceCommandFirst

Declaration

```
MPICommand mpiSequenceCommandFirst(MPISequence  sequence)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandFirst returns the first element in the list. This function can be used in conjuntion with mpiSequenceCommandNext() in order to iterate through the list.

sequence	a handle to the Sequence object.

Return Values	
handle	to the first Command in a Sequence (***sequence***)
MPIHandleVOID	if ***sequence*** is invalid if ***sequence*** is empty
MPIMessageHANDLE_INVALID	

See Also

mpiSequenceCommandNext | mpiSequenceCommandLast

mpiSequenceCommandIndex

Declaration

```
long mpiSequenceCommandIndex(MPISequence  sequence,
                             MPICommand   command)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandIndex returns the position of "command" on the list.

sequence	a handle to the Sequence object.
command	a handle to a Command object.

Return Values	
index	of a Command (***command***) in a Sequence (***sequence***)
-1	if ***sequence*** is invalid if the Command (***command***) was not found in the Sequence (***sequence***)

See Also

mpiSequenceCommandInsert

Declaration

```
long mpiSequenceCommandInsert(MPISequence sequence,
                              MPICommand  command,
                              MPICommand  insert)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandInsert inserts a Command (***insert***) in a Sequence (***sequence***) just after the specified Command (***command***).

Return Values	
MPIMessageOK	

See Also

mpiSequenceCommandNext | mpiSequenceCommandLast

mpiSequenceCommandLast

Declaration

```
MPICommand mpiSequenceCommandLast(MPISequence sequence)
```

Required Header: stdmpi.h

Description

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

sequence	a handle to the Sequence object.

Return Values	
MPIMessageOK	

Return Values	
handle	to the last Command in a Sequence (***sequence***)
MPIHandleVOID	if ***sequence*** is invalid if ***sequence*** is empty
MPIMessageHANDLE_INVALID	

See Also

mpiSequenceCommandFirst | mpiSequenceCommandPrevious | mpiSequenceCommandNext

mpiSequenceCommandListGet

Declaration

```
long mpiSequenceCommandListGet(MPISequence  sequence,
                               long         *commandCount,
                               MPICommand   *commandList)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandListGet gets the Commands in a Sequence (***sequence***). *SequenceCommandListGet* writes the number of Commands [in a Sequence (***sequence***)] to the location (pointed to by ***commandCount***), and also writes an array (of ***commandCount*** Command handles) to the location (pointed to by ***commandList***).

Return Values	
MPIMessageOK	

See Also

mpiSequenceCommandListSet

mpiSequenceCommandListSet

Declaration

```
long mpiSequenceCommandListSet(MPISequence  sequence,
                               long         commandCount,
                               MPICommand   *commandList)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandListSet creates a Sequence (***sequence***) of ***commandCount*** Commands using the Command handles specified by ***commandList***. Any existing command Sequence is completely replaced.

The ***commandList*** parameter is the address of an array of ***commandCount*** Command handles, or is NULL (if ***commandCount*** is equal to zero).

You can also create a command Sequence incrementally (i.e., one command at a time), by using the Append and/or Insert methods. Use the List methods to examine and manipulate a command Sequence, regardless of how it was created.

Return Values	
MPIMessageOK	

See Also

mpiSequenceCommandListGet

mpiSequenceCommandNext

Declaration

```
MPICommand mpiSequenceCommandNext(MPISequence sequence,
                                  MPICommand  command)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandNext returns the next element following "command" on the list. This function can be used in conjuntion with mpiSequenceCommandFirst(...) in order to iterate through the list.

sequence	a handle to the Sequence object.
command	a handle to a Command object.

Return Values	
handle	to the Command following the Command (***command***) in a Sequence (***sequence***)
MPIHandleVOID	if ***sequence*** is invalid if ***command*** is the last command in a Sequence (***sequence***)
MPIMessageHANDLE_INVALID	

See Also

mpiSequenceCommandFirst | mpiSequenceCommandPrevious

mpiSequenceCommandPrevious

Declaration

```
MPICommand mpiSequenceCommandPrevious(MPISequence sequence,
                                      MPICommand  command)
```

Required Header: stdmpi.h

Description

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

sequence	a handle to the Sequence object.
command	a handle to a Command object.

Return Values	
handle	to the Command preceding the Command (***command***) in a Sequence (***sequence***)
MPIHandleVOID	if ***sequence*** is invalid if ***command*** is the first command in a Sequence (***sequence***)
MPIMessageHANDLE_INVALID	

See Also

mpiSequenceCommandLast | mpiSequenceCommandNext

mpiSequenceCommandRemove

Declaration

```
long mpiSequenceCommandRemove(MPISequence sequence,
                              MPICommand  command)
```

Required Header: stdmpi.h

Description

mpiSequenceCommandRemove removes a Command (***command***) from a Sequence (***sequence***).

Return Values	
MPIMessageOK	

See Also

MPISequenceConfig / MEISequenceConfig

Definition: MPISequenceConfig

```
typedef MPIEmpty    MPISequenceConfig;
```

Description

MPISequenceConfig is currently not supported and is reserved for future use.

Definition: MEISequenceConfig

```
typedef MPIEmpty    MEISequenceConfig;
```

Description

MEISequenceConfig is currently not supported and is reserved for future use.

See Also

mpiSequenceConfigGet | mpiSequenceConfigSet

MPISequenceMessage

Definition

```
typedef enum {

    MPISequenceMessageSEQUENCE_INVALID,
    MPISequenceMessageCOMMAND_COUNT,
    MPISequenceMessageCOMMAND_NOT_FOUND,
    MPISequenceMessageSTARTED,
    MPISequenceMessageSTOPPED,
} MPISequenceMessage;
```

Description

MPISequenceMessage is an enumeration of Sequence error messages that can be returned by the MPI library.

MPISequenceMessageSEQUENCE_INVALID

The sequence number is out of range. This message code is returned by mpiSequenceCreate(...) if the sequence number is less than zero or greater than or equal to MEIXmpMAX_PSs. This message code is also returned if the specified sequence number is not active in the controller. To correct this problem, use mpiControlConfigSet(...) to enable the sequence object, by setting the sequenceCount to greater than the sequence number. For example, to enable sequence 0 to 3, set sequenceCount to 4. This message code is returned by mpiSequenceLoad(...) if the sequence buffer size and the sequence page size are not equal. This indicates an internal MPI Library problem.

MPISequenceMessageCOMMAND_COUNT

The sequence command count is out of range. This message code is returned by mpiSequenceStart(...) or meiSequenceCompile(...) if the sequence command count is less than or equal to zero. To correct this problem, set the command count to a value greater than zero.

MPISequenceMessageCOMMAND_NOT_FOUND

The sequence command is not found. This message code is returned by mpiSequenceLoad(...), mpiSequenceStart(...), or meiSequenceCompile(...) if the specified command is not a member of the sequence. To correct this problem, specify a command that is a member of the sequence.

MPISequenceMessageSTARTED

The program sequencer is already running. This message code is returned by mpiSequenceResume(...), mpiSequenceStart(...), or mpiSequenceStep(...) if the program sequencer has already been started. If this is a problem, call mpiSequenceStop(...) to stop the program sequencer or monitor the sequence status and wait for the state to equal STOPPED.

MPISequenceMessageSTOPPED

The program sequencer is not running. This message code is returned by mpiSequenceStop(...) if the program sequencer has already been stopped. If this is a problem, call mpiSequenceStart(...) to start the program sequencer.

See Also

MPISequenceState

Definition

```
typedef enum {
    MPISequenceStateSTOPPED = 0,
    MPISequenceStateSTARTED,
} MPISequenceState;
```

Description

MPISequenceState is an enumeration of fan status bit for use in the MPIControlFanStatusMask. The status bits represent the present status condition(s) for the fan controller on a given Control object.

MPISequenceStateSTOPPED	Means that the XMP's on-board program sequencer state is stopped. The program sequencer is in this state after it is created, and is not running. If the program sequencer has already been started, then a call to the MPI method mpiSequenceStop will stop the sequencer, and the sequencer state will be MPISequenceStateSTOPPED.
MPISequenceStateSTARTED	Means that the XMP's on-board program sequencer state is running. The program sequencer is in this state after it has been created, and successfully started with a call to the MPI method mpiSequenceStart.

See Also

MPISequenceStatus

Definition

```
typedef struct MPISequenceStatus {
    MPICommand          command;
    MPISequenceState    state;
} MPISequenceStatus;
```

Description

MPISequenceStatus is a status structure for MPISequence objects.

command	The current command of the MPISequence object
state	The current state of the MPISequence object

See Also

MPISequence | mpiSequenceStatus

MEISequenceTrace

Definition

```
typedef enum {
    MEISequenceTraceLOAD,
} MEISequenceTrace;
```

Description

MEISequenceTrace sets tracing on for the mpiSequenceLoad(...) method.

See Also

MPISequence | MEITrace | mpiSequenceLoad

Service Objects

Introduction

A **Service** object creates and handles threads that help event managers dispatch events. Typically, one will use a Service Object to create threads that will call mpiEventMgrService whenever an XMP interrupt occurs. They are a convenient way to have a program automatically deal with event managers and events. Thread handling is something that is different on every operating system. Service objects may therefore have different behaviors on different operating systems. Programmers that are experienced in multi-threaded application programming will probably want to program their own threads that will call mpiEventMgrService.

NOTE: The Service object is not part of the standard MPI. In order to use the Service Object, the file, *apputil.h* needs to be included by your code and the apputil library needs to be linked to your application.

Methods

Create, Delete, Validate Methods

service**Create**	Create a Service for EventMgr and the threads necessary for it to run.
service**Delete**	Stop all threads belonging to the Service and deletes the Service.

Configuration and Information Methods

service**Enable**	Enable or disables the Service.

ServiceCreate

Declaration

```
const Service serviceCreate(MPIEventMgr eventMgr,
                            long        priority,
                            long        sleep)
```

Required Header: service.h

Description

ServiceCreate creates threads for each control associated with ***eventMgr***, flushes ***eventMgr***, and starts threads with priority that call mpiEventMgrService(eventMgr, .) every ***sleep*** milliseconds.

priority

is a platform-specific variable.

If "priority" is	Then
-1	The operating system will choose some default priority for the service's threads. See Default Thread Parameters table below.
>0	*ServiceCreate* will attempt to assign the priority to all of the service's threads.

Default Thread Parameters

These are the default thread priorities (when -1 is specified).

OS	Priority
Windows	THREAD_PRIORITY_TIME_CRITICAL
RTSS	RT_PRIORITY_MAX
VxWorks	0
Linux	Maximum (defined by pthread schedule policy max priority)
Solaris	Maximum (defined by pthread schedule policy max priority)

sleep

If "sleep" is	Then
-1	*ServiceCreate* will attempt to create interrupt driven threads.
0	*ServiceCreate* will create threads that call **mpiEventMgrService(eventMgr,...)** as quickly as possible.
>0	*ServiceCreate* will create threads that attempt to call **mpiEventMgrService(eventMgr,...)** every ***sleep*** milliseconds.

Return Values	
handle	to a Service object
MPIHandleVOID	if the Service could not be created

See Also

mpiEventMgrService | ServiceDelete

ServiceDelete

Declaration

```
long serviceDelete(Service service)
```

Required Header: service.h

Description

ServiceDelete alerts all threads that they should end, waits for all threads to end, and frees the memory allocated to ***service***.

Return Values	
MPIMessageOK	

See Also

ServiceCreate

ServiceEnable

Declaration

```
long serviceEnable(Service service,
                   long    enabled)
```

Required Header: service.h

Description

ServiceEnable enables or disables all threads belonging to Service.

If "enabled" is	Then
False	*serviceEnable* will disable ***service***.
True	*serviceEnable* will enable ***service***.

Return Values	
handle	to a Service object
MPIHandleVOID	if the Service could not be created

See Also

SqNode Objects

Introduction

A **SqNode** object manages a single SynqNet network node connected to a SynqNet network. It represents the physical network node. It contains information about the node, as well as its status and configuration. It provides read/write access to the node via network cyclic data and service commands. It also provides an interface to any drives connected to the node.

During network initialization, the SynqNet nodes are discovered and mapped to the SynqNet object. The number of motors per SqNode is determined and mapped to the controller's motor objects. Each node connected to a controller is assigned a number (0, 1, 2, etc) in the order it is discovered. The node number is used to index the SqNode objects.

| Error Messages |

Methods

Create, Delete, Validate Methods

meiSqNode**Create**

meiSqNode**Delete**

meiSqNode**Validate**

Configuration and Information Methods

rmbAnalogInRangeGet

rmbAnalogInRangeSet

meiSqNode**Command**

meiSqNode**ConfigGet**

meiSqNode**ConfigSet**

meiSqNode**FlashConfigGet**

meiSqNode**FlashConfigSet**

meiSqNode**FpgaDefaultFileName**

meiSqNode**FpgaFilenameVerify**

meiSqNode**Info**

meiSqNode**Status**

meiSqNode**UserDataGet**

meiSqNode**UserDataSet**

Drive Interface Methods

meiSqNode**DriveConfigGet**

meiSqNode**DriveConfigSet**

meiSqNode**DriveInfo**
meiSqNode**DriveMapParamCount**
meiSqNode**DriveMapParamList**
meiSqNode**DriveMapConfigCount**
meiSqNode**DriveMapConfigList**
meiSqNode**DriveMapParamFileGet**
meiSqNode**DriveMapParamFileSet**
meiSqNode**DriveMonitor**
meiSqNode**DriveMonitorInfo**
meiSqNode**DriveMonitorConfigGet**
meiSqNode**DriveMonitorConfigSet**
meiSqNode**DriveParamCalculate**
meiSqNode**DriveParamClear**
meiSqNode**DriveParamGet**
meiSqNode**DriveParamListGet**
meiSqNode**DriveParamListSet**
meiSqNode**DriveParamReload**
meiSqNode**DriveParamRestore**
meiSqNode**DriveParamSet**
meiSqNode**DriveParamStore**

I/O Methods

meiSqNode**AnalogIn**
meiSqNode**AnalogInPtr**
meiSqNode**AnalogOutPtr**
meiSqNode**AnalogOutGet**
meiSqNode**AnalogOutSet**
meiSqNode**DigitalIn**
meiSqNode**DigitalInPtr**
meiSqNode**DigitalOutPtr**
meiSqNode**DigitalOutGet**
meiSqNode**DigitalOutSet**
meiSqNode**SegmentAnalogIn**
meiSqNode**SegmentAnalogOutGet**
meiSqNode**SegmentAnalogOutSet**
meiSqNode**SegmentInfo**
meiSqNode**SegmentDigitalIn**
meiSqNode**SegmentDigitalOutGet**
meiSqNode**SegmentDigitalOutSet**

meiSqNode**SegmentMemoryGet**
meiSqNode**SegmentMemorySet**
meiSqNode**SegmentParamDefault**
meiSqNode**SegmentParamStore**
meiSqNode**SegmentParamClear**
meiSqNode**SegmentParamGet**
meiSqNode**SegmentParamSet**
meiSqNode**SegmentUserDataGet**
meiSqNode**SegmentUserDataSet**

Action Methods

meiSqNode**Download**
meiSqNode**FlashErase**
meiSqNode**FpgaFileNameVerify**
meiSqNode**NetworkObjectNext**
meiSqNode**StatusClear**
meiSqNode**Verify**

Event Methods

meiSqNode**EventNotifyGet**
meiSqNode**EventNotifySet**
meiSqNode**EventReset**

Memory Methods

meiSqNode**Memory**
meiSqNode**MemoryGet**
meiSqNode**MemorySet**

Relational Methods

meiSqNode**Control**
meiSqNode**Number**

Data Types

RMBAnalogInRange

MEISqNode**Channel**
MEISqNode**CmdHeader**

MEISqNode**StatusIoFaults**

MEISqNode**StatusPacketError**

MEISqNode**UserData**

Constants

MEISqNode**ConfigControlLatencyMIN_LIMIT**

MEISqNode**ConfigControlLatencyMAX_LIMIT**

MEISqNode**Drive_Param_MAX_STRING_LENGTH**

MEISqNode**ID_CHAR_MAX**

MEISqNode**FILENAME_MAX**

MEISqNode**ManufacturerDATA_CHAR_MAX**

MEISqNode**MaxFEEDBACK_SECONDARY**

MEISqNode**MaxMOTORS**

MEISqNode**NOT_AVAILABLE**

MEISqNode**SEGMENT_MAX**

MEISqNode**SEGMENT_PARAMS_MAX**

MEISqNode**SEGMENT_MEMORY_MAX**

MEISqNode**STATUS_NOT_AVAILABLE**

MEISqNode**UserDATA_CHAR_MAX**

MEISqNode**SegmentInfoMANUFACTURER_LENGTH**

MEISqNode**SegmentInfoMODEL_NAME_LENGTH**

MEISqNode**SegmentInfoSERIAL_NUMBER_LENGTH**

MEISqNode**SegmentUserDATA_CHAR_MAX**

MEI**DriveMapParamMAX_STRING_LENGTH**

MEI**FPGARINCONREV**

MEI**FpgaSqMACVersionDEFAULT**

MEI**FpgaSqMACVersionMIN**

MEI**FpgaSqMACVersionMAX**

MEI**FpgaSqNodeVersionDEFAULT**

MEI**FpgaSqNodeVersionMIN**

MEI**FpgaSqNodeVersionMAX**

meiSqNodeCreate

Declaration

```
MEISqNode meiSqNodeCreate(MPIControl  control,
                          long        number)
```

Required Header: stdmei.h

Description

meiSqNodeCreate creates a SqNode object identified by ***number***, which is associated with a control object.

SqNodeCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	an index to the SqNode. The first node number is 0, the second is 1, etc.

Return Values	
handle	to a SqNode object. After creating a SqNode object it must be validated using meiSqNodeValidate(...).
MPIHandleVOID	if the object could not be created

See Also

meiSqNodeDelete | meiSqNodeValidate

meiSqNodeDelete

Declaration

```
long meiSqNodeDelete(MEISqNode  node);
```

Required Header: stdmei.h

Description

meiSqNodeDelete deletes a SqNode object and invalidates its handle.

SqNodeDelete is the equivalent of a C++ destructor.

node	a handle of the SqNode object to delete in the reverse order to avoid memory leaks.

Return Values	
MPIMessageOK	

See Also

meiSqNodeCreate | meiSqNodeValidate

meiSqNodeValidate

Declaration

```
long meiSqNodeValidate(MEISqNode  node);
```

Required Header: stdmei.h

Description

meiSqNodeValidate validates a SqNode object and its handle.

SqNodeValidate is the equivalent of a C++ constructor.

node	a handle to a SynqNet node object.

Return Values	
MPIMessageOK	

See Also

meiSqNodeCreate | meiSqNodeDelete

rmbAnalogInRangeGet

Declaration

```
long rmbAnalogInRangeGet(MEISqNode          sqNode,
                         long               channel,
                         RMBAnalogInRange   *analogInRange);
```

Required Header: stdmei.h

Description

The analog inputs on RMB-10V2 nodes support software configurable input ranges. The **rmbAnalogInRangeGet** function allows the current input range for each channel to be read.

sqNode	a handle to a SynqNet node object.
channel	the index of the analogue input channel.
***analogInRange**	a pointer to where the current input slice will be written by this function.

Return Values	
MPIMessageOK	

See Also

rmbAnalogInRangeSet | Analog Inputs on RMB Nodes

rmbAnalogInRangeSet

Declaration

```
long rmbAnalogInRangeSet(MEISqNode          sqNode,
                         long               channel,
                         RMBAnalogInRange   analogInRange);
```

Required Header: stdmei.h

Description

The analog inputs on RMB-10V2 nodes support software configurable input ranges. The **rmbAnalogInRangeSet** function allows you to change the input range for each channel.

sqNode	a handle to a SynqNet node object.
channel	the index of the analog input channel.
analogInRange	the new analogue input range.

Return Values	
MPIMessageOK	

See Also

rmbAnalogInRangeGet | Analog Inputs on RMB Nodes

meiSqNodeCommand

Declaration

```
long meiSqNodeCommand(MEISqNode         node,
                      MEISqNodeCommand  *command,
                      MEISqNodeResponse *response);
```

Required Header: stdmei.h

Description

meiSqNodeCommand sends a service command to a SynqNet node using the data from the structure pointed to by command and writes the response into the structure pointed to by response. Service commands occur across the SynqNet network through a service channel. In SYNQ mode there is one service channel for each node. In ASYNQ mode there is one service channel for all nodes. The controller sends the command and waits for a response using a 4 state handshake. In SYNQ mode the service command data is sent through the cyclic packets, but due to the handshaking, it is not considered a cyclic operation.

For SynqNet nodes that have drive memory interfaces, service commands can be sent to drives. Also, the service commands supports access to drive data memory, program memory, I/O memory, and direct commands. Please see the MEISqNodeCommand{.} and MEISqNodeResponse{.} structures for more information. And be sure to consult the drive's header file in the (C:\MEI)\XMP\sqNodeLib \include directory, as well as, the drive manufacturer's manual for valid drive addresses.

node	a handle to a SynqNet node object
***command**	a pointer to a SynqNet node command structure
***response**	a pointer to a SynqNet node response structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEISqNodeCommand | MEISqNodeResponse | MEISqNodeCmdHeader | MEISqNodeCmdType | MEISqNodeDataSize | MEISqNodeMemory | MEISqNodeChannel

meiSqNodeConfigGet

Declaration

```
long meiSqNodeConfigGet(MEISqNode        node,
                        MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeConfigGet reads a SynqNet node's (***node***) configuration and writes it into the structure pointed to by ***config***.

node	a handle to a SynqNet node object.
***config**	a pointer to a SynqNet node config structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveConfigGet

meiSqNodeConfigSet

Declaration

```
long meiSqNodeConfigSet(MEISqNode        node,
                        MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeConfigSet writes a SynqNet node's (***node***) configuration using data from the structure pointed to by ***config***.

node	a handle to a SynqNet node object
***config**	a pointer to a SynqNet node config structure

Return Values	
MPIMessageOK	
MEIMessageARG_INVALID	
MEISqNodeMessageFEEDBACK_MAP_INVALID	

See Also

meiSqNodeInfo | meiSqNodeDriveConfigSet | meiSynqNetFlashTopologySave

meiSqNodeFlashConfigGet

Declaration

```
long meiSqNodeFlashConfigGet(MEISqNode        node,
                             void             *flash,
                             MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeFlashConfigGet reads a SynqNet node's flash configuration and writes it into the structure pointed to by ***config***.

node	a handle to a SynqNet node 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 SynqNet node config structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeFlashConfigSet

meiSqNodeFlashConfigSet

Declaration

```
long meiSqNodeFlashConfigSet(MEISqNode        node,
                             void             *flash,
                             MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeFlashConfigSet sets a SynqNet Node (***node***) flash configuration using data from the structure pointed to by ***config***.

NOTE: The network topology must first be saved before changing node config values in Flash memory. These values will also be cleared when network topology is cleared using meiSynqNetFlashTopologyClear(...).

node	a handle to a SynqNet node 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 SynqNet node config structure.

Return Values	
MPIMessageOK	
MEISqNodeMessageFEEDBACK_MAP_INVAILD	
MPIMessageARG_INVALID	
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	

See Also

meiSqNodeFlashConfigGet | Flash Objects | meiSynqNetFlashTopologySave

meiSqNodeFpgaDefaultFileName

Declaration

```
long meiSqNodeFpgaDefaultFileName(MEISqNode           sqNode,
                                  MEISqNodeFileName   *fileName);
```

Required Header: stdmei.h

Description

meiSqNodeFpgaDefaultFileName provides the default image filename for an sqNode.

sqNode	handle to a SqNode object.
***fileName**	a pointer to a structure that has space allocated to hold an FPGA filename.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSqNodeFpgaFileNameVerify

Declaration

```
long meiSqNodeFpgaFileNameVerify(MEISqNode      sqNode,
                                 const  char*   fileName);
```

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

Description

meiSqNodeFpgaFileNameVerify verifies that the filename provided is compatible with a given sqNode.

sqNode	a handle to an SqNode object.
fileName	a pointer to a string containing the name of an SqNode image file.

Return Values	
MPIMessageOK	
MEISqNodeMessageFILE_NODE_MISMATCH	

See Also

meiSqNodeInfo

Declaration

```
long meiSqNodeInfo(MEISqNode       node,
                   MEISqNodeInfo   *info);
```

Required Header: stdmei.h

Description

meiSqNodeInfo reads a SynqNet node's information and writes it into a structure pointed to by ***info***. The info structure contains read only data about the node.

The RMB-10V, RMB-10V2 and some Trust nodes support analog inputs. MPI support has been added to support the reading of node-based analog inputs. The number of analog inputs a node supports can be determined with meiSqNodeInfo(...). An analog input value can be read with meiSqNodeAnalogIn(...). The analog to digital converted value is scaled from -1.0 to +1.0, where +1.0 is a full-scale positive voltage. The input range of the ADC is hardware-specific.

node	a handle to a SynqNet node object.
***info**	a pointer to a drive specific information structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

Sample Code

```
MEISqNodeInfo sqNodeInfo;
meiSqNodeInfo( sqNode0, &sqNodeInfo );

long x = sqNodeInfo.io.digitalInputCount;
```

See Also

meiSqNodeDriveInfo | meiSqNodeConfigGet | meiSqNodeDriveConfigGet

meiSqNodeStatus

Declaration

```
long meiSqNodeStatus(MEISqNode          node,
                     MEISqNodeStatus    *status);
```

Required Header: stdmei.h

Description

meiSqNodeStatus reads status from the ***node*** associated with the SynqNet object and writes it into the structure pointed to by ***status***. The SynqNet node status structure contains error counters and event mask data.

NOTE: This data requires service commands to acces the data on the node. As a result, it may take up to 9 servo cycles to read the data. At the default sample rate of 2kHz, this would translate to 4.5ms.

node	a handle to a SynqNet node object.
***status**	pointer to a SynqNet status structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSynqNetStatus | meiSqNodeInfo

meiSqNodeUserDataGet

Declaration

```
long meiSqNodeUserDataGet(MEISqNode          node,
                          MEISqNodeUserData  *data);
```

Required Header: stdmei.h

Description

meiSqNodeUserDataGet reads the user data from the node.

node	a handle to a SynqNet node object.
***data**	a pointer to a MEISqNodeUserData structure, allocated on the host.

Return Values	
MPIMessageOK	

See Also

meiSqNodeUserDataSet | Accessing Digital Data

meiSqNodeUserDataSet

Declaration

```
long meiSqNodeUserDataSet(MEISqNode          node,
                          MEISqNodeUserData  *data);
```

Required Header: stdmei.h

Description

meiSqNodeUserDataSet writes the user data to the node.

node	a handle to a SynqNet node object.
***data**	a pointer to a MEISqNodeUserData structure, allocated on the host.

Return Values	
MPIMessageOK	

See Also

meiSqNodeUserDataGet | MEISqNodeUserData | Accessing Digital Data

meiSqNodeDriveConfigGet

Declaration

```
long meiSqNodeDriveConfigGet(MEISqNode  node,
                             long       driveIndex, /* relative to the node */
                             void       *config);   /* node specific        */
```

Required Header: stdmei.h

Description

meiSqNodeDriveConfigGet reads a SynqNet node's drive configuration and writes it into a drive specific structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive configuration can be read if the drive interface hardware supports a communication channel to the drive processor. The drive interface (s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The drive configuration structure is drive specific. The SqNodeLib includes the drive specific structures and methods. Please see the drive's header file in the (C:\MEI)\XMP\sqNodeLib\include directory, as well as, the drive manufacturer's documentation for details. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and its type.

node	a handle to a SynqNet node object
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive specific configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo | meiSqNodeConfigGet

meiSqNodeDriveConfigSet

Declaration

```
long meiSqNodeDriveConfigSet(MEISqNode  node,
                             long       driveIndex, /* relative to the node */
                             void       *config);   /* node specific       */
```

Required Header: stdmei.h

Description

meiSqNodeDriveConfigSet writes a SynqNet node's drive configuration from a drive specific structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive configuration can be written if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The drive configuration structure is drive specific. The SqNodeLib includes the drive specific structures and methods. Please see the drive manufacturer's documentation for details. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and its type.

node	a handle to a SynqNet node object
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive specific configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo | meiSqNodeConfigSet

meiSqNodeDriveInfo

Declaration

```
long meiSqNodeDriveInfo(MEISqNode           node,
                        long                driveIndex, /* relative to the node */
                        MEISqNodeDriveInfo  *info,
                        void                *external);  /* node specific */
```

Required Header: stdmei.h

Description

meiSqNodeDriveInfo reads a SynqNet node's drive information and writes it into a drive specific structure pointed to by ***info***. The drive info structure contains read only data. SynqNet nodes may support one or more drive interfaces. The drive information can be read if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The drive information structure is drive specific. The SqNodeLib includes the drive specific structures and methods. Please see the drive's header file for the drive specific information structures, as well as, the drive manufacturer's documentation for details. All supported drive header files are located in the (C:\MEI)\XMP\sqNodeLib\include directory. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and its type.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***info**	a pointer to a structure that contains general drive information.
***external**	a pointer to a drive specific information structure. See the appropriate drive vendor *.h for definition. (NOTE: it can be NULL)

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeConfigGet | meiSqNodeDriveConfigGet

meiSqNodeDriveMapParamCount

Declaration

```
long meiSqNodeDriveMapParamCount (MEISqNode     sqNode,
                                  MEIDriveMap   driveMap,
                                  long          driveIndex,
                                  long          *paramsCount);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamCount scans the drive map file for a drive entry that matches this node on the network. If an entry is found, then this function returns the number of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiSqNodeDriveMapParamList(...) function. First, this function is called in order to get the size of the drive parameter list. Then the user can use this size to allocate enough memory to hold the complete parameter list before calling meiSqNodeDriveMapParamList(...) to fill in the list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***paramsCount**	pointer to the variable that will be set by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamList

meiSqNodeDriveMapParamList

Declaration

```
long meiSqNodeDriveMapParamList (MEISqNode          sqNode,
                                 MEIDriveMap        driveMap,
                                 long               driveIndex,
                                 long               paramsCount,
                                 MEIDriveParamInfo  *driveParamInfo);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamList scans the drive map file for an entry that matches the node on the network. If a drive entry is found, this function writes the drive parameter information about each of the drive parameters to the ***driveParamInfo*** list.

This function is normally used with the meiSqNodeDriveMapParamCount(...) function. The meiSqNodeDriveMapParamCount(...) function is called first to get the size of the parameter list, the user can then use this size to allocate enough memory to hold the complete parameter list before calling this function to fill in the parameter list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
paramsCount	the number of drive parameter information records that can be written to the *driveParamInfo* list.
***driveParamInfo**	pointer to the list of drive parameter information records that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamCount

meiSqNodeDriveMapConfigCount

Declaration

```
long meiSqNodeDriveMapConfigCount (MEISqNode     sqNode,
                                   MEIDriveMap   driveMap,
                                   long          driveIndex,
                                   long          *configCount);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapConfigCount scans the drive map file for a drive entry that matches this node on the network. If an entry is found, this function returns the number of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiSqNodeDriveMapConfigList function. This function is first called to get the size of the drive configuration list. Then the user can use this size to allocate enough memory to hold the complete configuration list before calling meiSqNodeDriveMapConfigList(...) to fill in the list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***configCount**	pointer to the variable that will be set by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapConfigList

meiSqNodeDriveMapConfigList

Declaration

```
long meiSqNodeDriveMapConfigList (MEISqNode     sqNode,
                                  MEIDriveMap   driveMap,
                                  long          driveIndex,
                                  long          configCount,
                                  long          *configList);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapConfigList scans the drive map file for a drive entry that matches this node on the network. If an entry is found, this function returns the list of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiSqNodeDriveMapConfigCount(...) function. The meiSqNodeDriveMapConfigCount(...) function is first called to get the size of the drive configuration list. Then the user can use this size to allocate enough memory to hold the complete configuration list before calling this function to fill in the list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
configCount	the number of drive parameter information records that can be written to the *configList* list.
***configList**	pointer to the list of drive parameters that make up the drive configuration that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapConfigCount

meiSqNodeDriveMapParamFileGet

Declaration

```
long meiSqNodeDriveMapParamFileGet(MEISqNode     sqNode,
                                   MEIDriveMap   driveMap,
                                   long          driveIndex,
                                   char          *driveConfigFilename,
                                   long          append);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamFileGet saves the current set of drive parameters in the drive to the ***driveConfigFilename***.

sqNode	a handle to the SynqNet node object.
driveMap	a handle to the DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***driveConfigFilename**	the name of the file that holds the stored drive configuration file.
append	1 = The new data is appended to the existing drive configuration file if it exists. 0 = A new drive configuration file is created to hold the drive parameters. If a file already exists, it will be overwritten.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamFileSet

SynqNet Drive Parameters

meiSqNodeDriveMapParamFileSet

Declaration

```
long meiSqNodeDriveMapParamFileSet(MEISqNode                  sqNode,
                                   MEIDriveMap                driveMap,
                                   long                       driveIndex,
                                   char                       *driveConfigFilename,
                                   MEISqNodeDriveParamCallback callback,
                                   long                       warning);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamFileSet loads the drive parameters stored in the file named ***driveConfigFilename*** into the drive.

sqNode	a handle to the SynqNet node object.
driveMap	a handle to the DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***driveConfigFilename**	the name of the file that holds the stored drive configuration file.
callback	A callback function that this function calls to indicate if the function is changing the value of a drive parameter or setting a new drive parameter that has failed. Passing NULL for this parameter will disable the callback feature.
warning	0 = if setting a drive parameter fails, this function will fail immediately. 1 = if setting a drive parameter fails, then the function will continue with the remaining drive parameters and generate a warning by calling the callback function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamFileGet

SynqNet Drive Parameters

meiSqNodeDriveMonitor

Declaration

```
long meiSqNodeDriveMonitor(MEISqNode              node,
                           long                   driveIndex, /* relative
to
                                                              the node */
                           MEISqNodeMonitorValue  *value);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitor reads the monitor fields from the drive and writes them into the structure pointed to by ***value***.

node	a handle to a SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***value**	pointer to a structure of monitor values

Return Values	
MPIMessageOK	

See Also

MEISqNodeMonitorValue

meiSqNodeDriveMonitorInfo

Declaration

```
long  meiSqNodeDriveMonitorInfo(MEISqNode                  node,
                                long                       driveIndex,
                                MEISqNodeDriveMonitorInfo  *info);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitorInfo reads a SynqNet node drive's monitor information and writes it into a structure pointed to by ***info***. The ***info*** structure contains read only data about the number of supported drive monitor fields, their locations, and how to decode them.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***info**	a pointer to a MEISqNodeDriveMonitorInfo structure. It contains information about the number of monitor fields, their locations, and how to decode them.

Return Values	
MPIMessageOK	

See Also

MEISqNodeDriveMonitorInfo

meiSqNodeDriveMonitorConfigGet

Declaration

```
long meiSqNodeDriveMonitorConfigGet(MEISqNode                node,
                                    long                     driveIndex, /* relative to
                                                                            the node */
                                    MEISqNodeDriveMonitorConfig *config);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitorConfigGet reads a SynqNet node's drive monitor configuration and writes it into a structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive monitor configuration can be read if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The SynqNet network packets have some extra fields that can be configured to read drive data every sample. Each monitor field is 32 bits. SynqNet nodes with drive interfaces that support drive monitoring can be configured to transmit the data. The drive manufacturer determines what data is available for monitoring. The monitor data can be specified by a predetermined index or memory address. Please see the drive's header file for the drive specific configuration structures, as well as, the drive manufacturer's documentation for details. All supported drive header files are located in the (C:\MEI)\XMP\sqNodeLib\include directory.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive monitor configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo

meiSqNodeDriveMonitorConfigSet

Declaration

```
long meiSqNodeDriveMonitorConfigSet(MEISqNode  node,
                                    long       driveIndex, /* relative to
                                                              the node */
                  MEISqNodeDriveMonitorConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitorConfigSet writes a SynqNet node's drive monitor configuration from a structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive monitor configuration can be written if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The SynqNet network packets have some extra fields that can be configured to read drive data every sample. Each monitor field is 32 bits. SynqNet nodes with drive interfaces that support drive monitoring can be configured to transmit the data. The drive manufacturer determines what data is available for monitoring. The monitor data can be specified by a predetermined index or memory address. Please see the drive's header file for the drive specific configuration structures, as well as, the drive manufacturer's documentation for details. All supported drive header files are located in the (C:\MEI)\XMP\sqNodeLib \include directory.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive monitor configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo

meiSqNodeDriveParamCalculate

Declaration

```
long meiSqNodeDriveParamCalculate(MEISqNode    sqNode,
                                  long         driveIndex);
```

Required Header: stdmei.h

Description

Some drives need to calculate some internal quantities after a drive parameter has been changed. The **meiSqNodeDriveParamCalculate** function will instruct the drive to calculate its internal quantities. This feature is not supported or required by all drives.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamClear

Declaration

```
long meiSqNodeDriveParamClear(MEISqNode    sqNode,
                              long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamClear clears the previously saved drive by loading the default set of drive parameters into the current and non-volatile storage on the drive. These drive parameters will be used each time this drive is subsequently started (after a power-on or network reset). The default drive parameters will take effect immediately.

NOTE: This function may not be supported by all drives. The default set of drive parameters may be different between different drive types and different drive manufactures.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamReload

meiSqNodeDriveParamGet

Declaration

```
long meiSqNodeDriveParamGet(MEISqNode                node,
                            long                     driveIndex,
                            long                     param,
                            MEIDriveMapParamType     paramType,
                            MEIDriveMapParamValue    *value);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamGet reads a drive parameter from the drive and fills in the appropriate field of the union pointed to by ***value***. The ***paramType*** defines the type of data that is read from the drive and also defines which field will be used in the ***value*** union.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
param	an index for the drive parameter that is being accessed.
paramType	the type of the data read from the drive and which field will be used in the *value* union.
***value**	a pointer to the union that will be filled in.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamSet

meiSqNodeDriveParamListGet

Declaration

```
long meiSqNodeDriveParamListGet(MEISqNode                node,
                                long                     driveIndex,
                                long                     size,
                                long                     *paramList,
                                MEIDriveMapParamType     *paramTypes,
                                MEIDriveMapParamValue    *paramValues);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamListGet reads a series of drive parameters from the drive and fills in the appropriate fields of the unions pointed to by ***paramValues***. The ***paramTypes*** defines the type of data that is read from the drive and also defines which fields in the ***paramValues*** unions are going to be used.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
size	the number of drive parameters to be read.
***paramList**	a pointer to a list of the drive parameter indexes that are being accessed.
***paramTypes**	a pointer to a list of drive parameter types to be read from the drive and which field in the paramValues union is going to be used.
***paramValues**	a pointer to a list of unions that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamListSet

meiSqNodeDriveParamListSet

Declaration

```
long meiSqNodeDriveParamListSet(MEISqNode                node,
                                long                     driveIndex,
                                long                     size,
                                long                     *paramList,
                                MEIDriveMapParamType     *paramTypes,
                                MEIDriveMapParamValue    *paramValues);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamListSet writes a series of drive parameters pointed to by ***value*** to the drive. The ***paramTypes*** defines each type of data item that is written to the drive and also defines which field in the ***paramValues*** unions are going to be used.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
size	the number of drive parameters to be written.
***paramList**	a pointer to a list of drive parameter types to be written to the drive and which field in the paramValues union is going to be used.
***paramTypes**	a pointer to a list of the drive parameter indexes that are being accessed.
***paramValues**	a pointer to a list of unions that will be written to the drive.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamListGet

meiSqNodeDriveParamReload

Declaration

```
long meiSqNodeDriveParamReload(MEISqNode    sqNode,
                               long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamReload overwrites the current set of drive parameters with the set from the non-volatile storage on the drive. These new drive parameters will take effect immediately.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamClear

meiSqNodeDriveParamRestore

Declaration

```
long meiSqNodeDriveParamRestore(MEISqNode    sqNode,
                                long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamRestore loads the default set of drive parameters into current set of drive parameters on the drive. The default drive parameters will take effect immediately.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamStore

meiSqNodeDriveParamSet

Declaration

```
long meiSqNodeDriveParamSet(MEISqNode                node,
                            long                     driveIndex,
                            long                     param,
                            MEIDriveMapParamType     paramType,
                            MEIDriveMapParamValue    *value);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamSet writes the drive parameter that is pointed to by ***value*** to the drive. The ***paramType*** defines the type of data that is written to the drive and also defines which field will be used in the ***value*** union.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
param	an index for the drive parameter that is being accessed.
paramType	the type of data being written to the drive and which field will be used in the *value* union.
***value**	pointer to the union that will be written to the drive.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamGet

meiSqNodeDriveParamStore

Declaration

```
long meiSqNodeDriveParamStore(MEISqNode    sqNode,
                              long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamStore saves the drive parameters into non-volatile storage on the drive. These drive parameters will be used each time the drive is subsequently started (after a power-on or network reset).

NOTE: This function may not be supported by all drives.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamRestore

meiSqNodeAnalogIn

Declaration

```
long meiSqNodeAnalogIn(MEISqNode     node,
                       long          channel,
                       long          *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00. meiSqNodeAnalogIn replaced meiSqNodeAnalogInput.

Description

meiSqNodeAnalogIn gets the current state of an analog input on a SynqNet node.

NOTE: meiSqNodeAnalogIn replaced meiSqNodeAnalogInput in the MPI library.

node	a handle to a SynqNet node object.
channel	the index of the analog input channel (with respect to the node).
***state**	a pointer to where the current state of the input is written by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code reads the current state of the first analog input on Node 2:

```
long x;
meiSqNodeAnalogIn( sqNode2, 0, &x );
```

See Also

Accessing Analog Data | MEISqNodeInfoIo

meiSqNodeAnalogInPtr

Declaration

```
long meiSqNodeAnalogInPtr(MEISqNode   sqNode,
                          long        channel,
                          long        *mask,
                          long        **ptr);
```

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

Description

meiSqNodeAnalogInPtr gets a controller memory pointer and mask of an analog input (**channel** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

sqNode	a handle to a SynqNet node object.
channel	an index to a node's analog input channel.
***mask**	a pointer to a bitwise mask that corresponds to the specified word.
****ptr**	a pointer to a controller address that contains the analog input word.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record analog channel 2 on node 1.

```
long mask;
long * pointer;
meiSqNodeAnalogInPtr( sqNode1, 2, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalInPtr
meiSqNodeDigitalOutPtr
meiSqNodeAnalogOutPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeAnalogOutPtr

Declaration

```
long meiSqNodeAnalogOutPtr(MEISqNode   sqNode,
                           long        channel,
                           long        *mask,
                           long        **ptr);
```

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

Description

meiSqNodeAnalogOutPtr gets a controller memory pointer and mask of an analog output (**channel** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

Analog values are held as 16 bit words either in the upper or lower two bytes of a controllers memory location hence the mask obtained from this function will either be 0xFFFF0000 or 0x0000FFFF.

sqNode	a handle to a SynqNet node object.
channel	an index to a node's analog output channel.
***mask**	a pointer to a bitwise mask that corresponds to the specified word.
****ptr**	a pointer to a controller address that contains the analog output word.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record analog output channel 2 on node 1.

```
long mask;
long * pointer;
meiSqNodeAnalogOutPtr( sqNode1, 2, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalInPtr
meiSqNodeDigitalOutPtr
meiSqNodeAnalogInPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeAnalogOutGet

Declaration

```
long meiSqNodeAnalogOutGet(MEISqNode     node,
                           long          channel,
                           long          *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeAnalogOutGet reads the current state of an analog output on a SynqNet node.

node	a handle to a SynqNet node object.
channel	the index of the analog output channel (with respect to the node).
***state**	a pointer to where the current state of the output is written by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeAnalogSet | Accessing Analog Data | MEISqNodeInfoIo

meiSqNodeAnalogOutSet

Declaration

```
long meiSqNodeAnalogOutSet(MEISqNode     node,
                           long          channel,
                           long          state,
                           MPI_BOOL      wait);
```

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

Description

meiSqNodeAnalogOutSet changes the current state of an analog output on a SynqNet node.

node	a handle to a SynqNet node object.
channel	the index of the analog output channel (with respect to the node).
state	the desired state of the analog output.
wait	determines what happens if two output functions are called in short succession. See Overview of Motor I/O: Output Waits.

Return Values	
MPIMessageOK	

See Also

meiSqNodeAnalogOutGet | Accessing Analog Data | MEISqNodeInfoIo

meiSqNodeDigitalIn

Declaration

```
long meiSqNodeDigitalIn(MEISqNode      node,
                        long           bitStart,
                        long           bitCount,
                        unsigned long  *state );
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeDigitalIn gets the current state of multiple digital inputs on the specified SynqNet node.

node	a handle to a SynqNet node object.
bitStart	the first bit.
bitCount	the number of bits to be read.
***state**	a pointer to where the current state of the input bits is written to by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code will get the current state of digital inputs 3, 4, and 5 of Node 0 (in the example network these bits spread over segments 0 and 1):

```
long x;
meiSqNodeDigitalIn( sqNode0, 3, 3, &x );
```

See Also

Accessing Digital I/O | SQIO-DIN32DOUT32: Input Pinouts

meiSqNodeDigitalInPtr

Declaration

```
long meiSqNodeDigitalInPtr(MEISqNode   sqNode,
                           long        bit,
                           long        *mask,
                           long        **ptr);
```

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

Description

meiSqNodeDigitalInPtr gets a controller memory pointer and mask of a digital input (**bit** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

sqNode	a handle to a SynqNet node object.
bit	an index to a node's digital input bit.
***mask**	a pointer to a bitwise mask that corresponds to the specified bit.
****ptr**	a pointer to a controller address that contains the digital input bit.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record bit 5 on node 1.

```
long mask;
long * pointer;
meiSqNodeDigitalInPtr( sqNode1, 5, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalOutPtr
meiSqNodeAnalogInPtr
meiSqNodeAnalogOutPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeDigitalOutPtr

Declaration

```
long meiSqNodeDigitalOutPtr(MEISqNode   sqNode,
                            long        bit,
                            long        *mask,
                            long        **ptr);
```

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

Description

meiSqNodeDigitalOutPtr gets a controller memory pointer and mask of a digital output (**bit** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

sqNode	a handle to a SynqNet node object.
bit	an index to a node's digital input bit.
***mask**	a pointer to a bitwise mask that corresponds to the specified bit.
****ptr**	a pointer to a controller address that contains the digital output bit.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record bit 5 on node 1.

```
long mask;
long * pointer;
meiSqNodeDigitalOutPtr( sqNode1, 5, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalInPtr
meiSqNodeAnalogInPtr
meiSqNodeAnalogOutPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeDigitalOutGet

Declaration

```
long meiSqNodeDigitalOutGet(MEISqNode       node,
                            long            bitStart,
                            long            bitCount,
                            unsigned  long  *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeDigitalOutGet reads the current state of the multiple digital output bits on the specified SynqNet node.

node	a handle to a SynqNet node object.
bitStart	the first bit.
bitCount	the number of bits to be read.
***state**	a pointer to where the current state of the output bits is written to by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDigitalOutSet | Accessing Digital I/O | SQIO-DIN32DOUT32: Output Pinouts

meiSqNodeDigitalOutSet

Declaration

```
long meiSqNodeDigitalOutSet(MEISqNode       node,
                            long            bitStart,
                            long            bitCount,
                            unsigned  long  state,
                            MPI_BOOL        wait);
```

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

Description

meiSqNodeDigitalOutSet changes the state of multiple digital outputs on the specified SynqNet node.

node	a handle to a SynqNet node object.
bitStart	the first bit.
bitCount	the number of bits to be changed.
state	a pointer to where the current state of the output bits is written to by this function.
wait	Boolean flag indicating if the new output state is applied immediately or if a wait is inserted so that any previous output set is transmitted over SynqNet and applied to the output before this function. You should be able to use TRUE for this argument in most applications.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code will set the top 16 bits and clear the bottom 16 bits of Node 1:

```
long x = 0xFFFF0000;
meiSqNodeDigitalOutSet( sqNode1, 0, 32, &x, TRUE );
```

The following MPI code will only set the fifth bit on a node 0 (the second bit of the segment 1):

```
meiSqNodeDigitalOutSet( sqNode0, 5, 1, 1, TRUE );
```

The following MPI code with ***wait*** true will ensure that the physical pin will definitely generate a short pulse.

```
meiSqNodeDigitalOutSet( sqNode1, 0, 1, 1, /*wait*/ 1 );
meiSqNodeDigitalOutSet( sqNode1, 0, 1, 0, /*wait*/ 1 );
```

The following MPI code shows the use of not introducing a ***wait***. This code is updating two bits on the same node. Both calls will immediately exit and since the timing of these two bits in this application are not related, this code can execute faster.

```
meiSqNodeDigitalOutSet( sqNode1, 7, 1, 1, /*wait*/ 0 );
meiSqNodeDigitalOutSet( sqNode1, 8, 1, 1, /*wait*/ 0 );
```

See Also

meiSqNodeDigitalOutGet | Accessing Digital I/O | SQIO-DIN32DOUT32: Output Pinouts

meiSqNodeSegmentAnalogIn

Declaration

```
long meiSqNodeSegmentAnalogIn(MEISqNode    node,
                              long         segment,
                              long         channel,
                              long         *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeSegmentAnalogIn gets the current state of an analog input on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
channel	the index of the analog input channel (with respect to the slice). For more information, please see the Overview of MPI I/O.
***state**	a pointer to where the current state of the input is written by this function.

Return Values	
MPIMessageOK	

See Also

Accessing Analog Data | MEISqNodeInfoIo | Overview of MPI I/O

meiSqNodeSegmentAnalogOutGet

Declaration

```
long meiSqNodeSegmentAnalogOutGet(MEISqNode    node,
                                  long         segment,
                                  long         channel,
                                  long         *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeSegmentAnalogOutGet gets the current state of an analog output on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
channel	the index of the analog output channel (with respect to the slice). For more information, please see the Overview of MPI I/O.
***state**	a pointer to where the current state of the output is written by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeSegmentAnalogOutSet | Accessing Analog Data | MEISqNodeInfoIo

Overview of MPI I/O

meiSqNodeSegmentAnalogOutSet

Declaration

```
long meiSqNodeSegmentAnalogOutSet(MEISqNode    node,
                                  long         segment,
                                  long         channel,
                                  long         state,
                                  MPI_BOOL     wait);
```

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

Description

meiSqNodeSegmentAnalogOutSet changes the current state of an analog output on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
channel	the index of the analog input channel (with respect to the slice). For more information, please see the Overview of MPI I/O.
state	the desired state of the analog output.
wait	determines what happens if two output functions are called in short succession. See Overview of Motor I/O: Output Waits.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code sets the analog output Number 2 on Segment 1 of Node 1 to +5V.

```
meiSqNodeSegmentAnalogOutSet( sqNode1, 1, 2, 0x3FFF );
```

See Also

meiSqNodeSegmentAnalogOutGet | Accessing Analog Data | MEISqNodeInfoIo

Overview of MPI I/O

meiSqNodeSegmentInfo

Declaration

```
long meiSqNodeSegmentInfo(MEISqNode              node,
                          long                   segment,
                          MEISqNodeSegmentInfo   *info);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeSegmentInfo reads the constant data about a segment on a SynqNet node and fills in the structure pointer by the ***info*** argument.

node	a handle to a SynqNet node object
segment	the index of the slice / module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
***info**	a pointer to a structure that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code shows how to read the number of digital inputs on Slice 1 of Node 0.

```
MEISqNodeSegmentInfo segmentInfo;
meiSqNodeSegmentInfo( sqNode0, 1, &segmentInfo );

long x = segmentInfo.digitalInCount;
```

See Also

MEISqNodeSegmentInfo | What Information is Available About Each I/O Segment? | Overview of MPI I/O

meiSqNodeSegmentDigitalIn

Declaration

```
long meiSqNodeSegmentDigitalIn(MEISqNode      node,
                               long           segment,
                               long           bitStart,
                               long           bitCount,
                               unsigned long  *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeSegmentDigitalIn gets the current state of multiple digital inputs on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
bitStart	the first bit
bitCount	the number of bits to be read.
***state**	a pointer to a long word that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code will get the current state of all the digital inputs (the following slice has 8 inputs) on segment 1 of node 0:

```
long x;
meiSqNodeSegmentDigitalIn( sqNode0, 1, 0, 8, &x );
```

See Also

Accessing Digital I/O

Overview of MPI I/O

meiSqNodeSegmentDigitalOutGet

Declaration

```
long meiSqNodeSegmentDigitalOutGet(MEISqNode     node,
                                   long          segment,
                                   long          bitStart,
                                   long          bitCount,
                                   unsigned long *state);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

meiSqNodeSegmentDigitalOutGet changes the state of multiple digital outputs on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
bitStart	the first bit
bitCount	the number of bits to be read.
state	a pointer to where the current state of the output bits is written to by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeSegmentDigitalOutSet | Accessing Digital I/O | Overview of MPI I/O

meiSqNodeSegmentDigitalOutSet

Declaration

```
long meiSqNodeSegmentDigitalOutSet(MEISqNode      node,
                                   long           segment,
                                   long           bitStart,
                                   long           bitCount,
                                   unsigned long  state,
                                   MPI_BOOL       wait);
```

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

Description

meiSqNodeSegmentDigitalOutSet sets the current state of the multiple digital output bits on the specified slice/module on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
bitStart	the first bit
bitCount	the number of bits to be read.
state	a pointer to where the current state of the output bits is written to by this function.
wait	a Boolean flag indicating if the new output state is applied immediately or a wait is inserted so that any previous output set is transmitted over SynqNet. You should be able to use TRUE for this argument in most applications.

Return Values	
MPIMessageOK	

See Also

meiSqNodeSegmentDigitalOutGet | Accessing Digital I/O | Overview of MPI I/O

meiSqNodeSegmentMemoryGet

Declaration

```
long  meiSqNodeSegmentMemoryGet(MEISqNode  sqNode,
                                long       segment,
                                long       memoryStart,
                                long       memoryCount,
                                char       *data);
```

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

Description

meiSqNodeSegmentMemoryGet gets the current value of the specified slice memory registers from the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
memoryStart	the first memory register to get.
memoryCount	the number of memory registers to get.
***data**	a pointer to the array of memory registers that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

Here is the code to read the first ten memory parameters on slice 1.

```
char memory[10];
meiSqNodeSegementMemoryGet( sqNode, 1, 0, 10, memory );
```

See Also

Overview of MPI I/O | meiSqNodeSliceMemorySet

meiSqNodeSegmentMemorySet

Declaration

```
long  meiSqNodeSegmentMemorySet(MEISqNode  sqNode,
                                long       segment,
                                long       memoryStart,
                                long       memoryCount,
                                char       *data);
```

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

Description

meiSqNodeSegmentMemorySet changes the current value of the specified slice memory registers on the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
memoryStart	the first memory register to set.
memoryCount	the number of memory registers to set.
***data**	a pointer to the array of memory registers that be written to the segment by this function.

Return Values	
MPIMessageOK	

Sample Code

Here is the code read the first 2 memory parameters on slice 1.

```
char memory[10];
meiSqNodeSegementMemoryGet( sqNode, 1, 0, 10, memory );
```

See Also

Overview of MPI I/O | meiSqNodeSliceMemoryGet

meiSqNodeSegmentParamDefault

Declaration

```
long  meiSqNodeSegmentParamDefault(MEISqNode  sqNode);
```

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

Description

meiSqNodeSegmentParamDefault overwrites the current set of parameters with the default set of parameters.

sqNode	a handle to a SqNode object

Return Values	
MPIMessageOK	

Sample Code

```
meiSqNodeSegementMemorySet( sqNode );
```

See Also

Overview of MPI I/O | meiSqNodeSegmentParamStore | meiSqNodeSegmentParamClear

meiSqNodeSegmentParamStore

Declaration

```
long  meiSqNodeSegmentParamStore(MEISqNode  sqNode);
```

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

Description

meiSqNodeSegmentParamStore copies the current working set of parameters to the set of stored parameters held in non-volatile memory. This set of parameters will subsequently be used from a node power-on or reset.

sqNode	a handle to a SqNode object.

Return Values	
MPIMessageOK	

Sample Code

```
meiSqNodeSegementMemoryStore( sqNode );
```

See Also

Overview of MPI I/O | meiSqNodeSegmentParamDefault | meiSqNodeSegmentParamClear

meiSqNodeSegmentParamClear

Declaration

```
long  meiSqNodeSegmentParamClear(MEISqNode  sqNode);
```

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

Description

meiSqNodeSegmentParamClear function will delete any slice parameters previously stored on non-volatile memory. The next time the node powers up, the default parameters will be loaded on the slices.

sqNode	a handle to a SqNode object.

Return Values	
MPIMessageOK	

Sample Code

```
meiSqNodeSegementMemoryClear( sqNode );
```

See Also

Overview of MPI I/O | meiSqNodeSegmentParamStore | meiSqNodeSegmentParamDefault

meiSqNodeSegmentParamGet

Declaration

```
long  meiSqNodeSegmentParamGet(MEISqNode  sqNode,
                               long       segment,
                               long       paramStart,
                               long       paramCount,
                               char       *params);
```

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

Description

meiSqNodeSegmentParamGet gets the current value of the specified slice parameters from the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
paramStart	the first parameter to get.
paramCount	the number of parameters to get.
***params**	a pointer to the array of parameters that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

For example to get the first two slice parameters on slice 1.

```
char parameters[2];
meiSqNodeSegmentParamGet( sqNode0, 1, 0, 2, parameters );
```

See Also

meiSqNodeSliceParamSet | Overview of MPI I/O

meiSqNodeSegmentParamSet

Declaration

```
long  meiSqNodeSegmentParamSet(MEISqNode  sqNode,
                               long       segment,
                               long       paramStart,
                               long       paramCount,
                               char       *params);
```

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

Description

meiSqNodeSegmentParamSet gets the current value of the specified slice parameters from the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
paramStart	the first parameter to set.
paramCount	the number of parameters to set.
***params**	a pointer to the array of parameters that be written to the segment by this function.

Return Values	
MPIMessageOK	

Sample Code

For example to get the first two slice parameters on slice 1.

```
char parameters[2]; = { '\x00', '\x0F' };
meiSqNodeSegmentParamSet( sqNode0, 1, 0, 2, parameters );
```

See Also

Overview of MPI I/O | meiSqNodeSliceParamGet

meiSqNodeSegmentUserDataGet

Declaration

```
long meiSqNodeSegmentUserDataGet(MEISqNode                  node,
                                 long                       segment,
                                 MEISqNodeSegmentUserData   *data);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

Modules attached to a SQID node have a small section of non-volatile memory that can be used for any purpose by the user. **meiSqNodeSegmentUserDataGet** gets the user data from a segment on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
data	a pointer to where the user data is written by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code reads the contents on the non-volatile memory on segment 0 of node 1:

```
MEISqNodeSegmentUserData nodeData;

meiSqNodeSegmentUserDataGet( sqNode1, 0, &nodeData );
```

See Also

meiSqNodeSegmentUserDataSet | MPI Overview I/O: User Data | Overview of MPI I/O

meiSqNodeSegmentUserDataSet

Declaration

```
long meiSqNodeSegmentUserDataSet(MEISqNode                  node,
                                 long                       segment,
                                 MEISqNodeSegmentUserData   *data);
```

Required Header: stdmei.h
Change History: Added in the 03.02.00

Description

Modules attached to a SQID node have a small section of non-volatile memory that can be used for any purpose by the user. **meiSqNodeSegmentUserDataSet** changes the user data stored on a segment on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
data	a pointer to the new user data.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code changes the contents on the non-volatile memory to a string:

```
MEISqNodeSegmentUserData nodeData;

strncpy( nodeData.data, “test” , MEISqNodeUserDATA_CHAR_MAX );

meiSqNodeSegmentUserDataSet( sqNode1, 0,&nodeData );
```

See Also

meiSqNodeSegmentUserDataGet | MPI Overview I/O: User Data | Overview of MPI I/O

meiSqNodeDownload

Declaration

```
long meiSqNodeDownload(MEISqNode                  node,
                       MEISqNodeDownloadParams    *params);
```

Required Header: stdmei.h

Description

meiSqNodeDownload SqNodeDownload reads a binary image from a file and writes it into a SynqNet node's non-volatile storage. SynqNet nodes may support one or more drive interfaces. SqNodeDownload can also write binary images to a drives' non-volatile storage if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The SynqNet node binary files are node specific. Please see the Node Binary Files: Product Table.

The SynqNet drive binary files are drive specific. The SqNodeLib includes the drive specific code necessary to support various hardware download protocols. Please see the drive manufacturer's documentation for details. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and it's type.

The binary download process requires a significant amount of time, probably between 5 to 30 seconds, depending on the node/drive type and file size. A callback function pointer is provided in the MEISqNodeDownloadParams structure for the calling application to monitor the download progress.

node	a handle to a SynqNet node object
***params**	a pointer to the download parameters structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSynqNetInfo | MEISqNodeDownloadParams | MEISqNodeChannel | MEISqNodeCallback

meiSqNodeFlashErase

Declaration

```
long meiSqNodeFlashErase(MEISqNode     sqNode);
```

Required Header: stdmei.h

Description

meiSqNodeFlashErase brings the SynqNet network down to discovery mode, sends a service command down to the node that erases its runtime flash, and leaves the network down. The next time the network is brought up to Synq mode the node will be running off its boot image.

node	a handle to a SynqNet node object.

Return Values	
MPIMessageOK	

See Also

meiSqNodeNetworkObjectNext

Declaration

```
long meiSqNodeNetworkObjectNext(MEISqNode              node,
                                MEINetworkPort         port,
                                MEINetworkObjectInfo   *info);
```

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

Description

meiSqNodeNetworkObjectNext gets the information for the neighboring network object (device) connected to the specified port and writes the information into the structure pointed to by ***info***.

NOTE: This info.type value may be MEINetworkObjectTypeNONE if there is nothing connected to the given port.

node	a handle to a SynqNet node object.
port	specifies the node's IN or OUT port.
***info**	a pointer to the next object's info structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSynqNetNetworkObjectNext | MEINetworkObjectInfo

Version Utility

meiSqNodeStatusClear

Declaration

```
long meiSqNodeStatusClear(MEISqNode     node);
```

Required Header: stdmei.h

Description

meiSqNodeStatusClear clears node CRC errors on all ports, clears node Packet errors, clears node ioAbort state, and resets SqNode events.

node	a handle to a SynqNet node object

Return Values	
MPIMessageOK	

See Also

MEIEventTypeSQNODE

meiSqNodeVerify

Declaration

```
long meiSqNodeVerify(MEISqNode                  node,
                     MEISqNodeDownloadParams    *params);
```

Required Header: stdmei.h

Description

meiSqNodeVerify verifies that the runtime image on a sqNode matches the data contained in a provided image file.

node	a handle to SqNode object.
***params**	a pointer to parameters used in the verify routine.

Return Values	
MPIMessageOK	
MEISqNodeMessageVERIFY_FAIL	

See Also

meiSqNodeEventNotifyGet

Declaration

```
long meiSqNodeEventNotifyGet(MEISqNode       node,
                             MPIEventMask    *eventMask,
                             void            *external);
```

Required Header: stdmei.h

Description

meiSqNodeEventNotifyGet reads the event mask (that specifies the event types 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).

Use the event mask macros mpiEventMaskGET(...), mpiEventMaskBitGET(...), etc. to decode the eventMask.

The event notification data in external is in addition to the event notification data in eventMask. If external is NULL, the event notification data will not be copied to the external pointer.

Remarks

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

node	a handle to a SynqNet node object
***eventMask**	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
***external**	pointer to external

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MPI/MEIEventType | MEIEventNotifyData | MEIEventStatusInfo

meiSqNodeEventNotifySet

Declaration

```
long meiSqNodeEventNotifySet(MEISqNode       node,
                             MPIEventMask    eventMask,
                             void            *external);
```

Required Header: stdmei.h

Description

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

Use the event mask macros meiEventMaskSQNODE(...), mpiEventMaskSET(...), mpiEventMaskBitSET(...), mpiEventMaskCLEAR(...), etc. to create the eventMask.

The event notification data in external is in addition to the event notification data in eventMask. If external is NULL, the event notification data will not be copied to the external pointer.

Remarks

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

The MEIEventNotifyData structure is an array of controller addresses, whose contents are placed into the MEIEventStatusInfo structure (of all events generated by this object).

node	a handle to a SynqNet node object
eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
***external**	pointer to external

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEI/MPIEventType | MEIEventNotifyData | MEIEventStatusInfo

meiSqNodeEventReset

Declaration

```
long meiSqNodeEventReset(MEISqNode      sqNode,
                         MPIEventMask   eventMask);
```

Required Header: stdmei.h

Description

meiSqNodeEventReset is a method used to reset events that have been latched on a node. Events that can be reset by this method include:

See MEIEventType.

/* SqNode events */
MEIEventTypeSQNODE_IO_ABORT,
MEIEventTypeSQNODE_NODE_DISABLE,
MEIEventTypeSQNODE_NODE_ALARM,
MEIEventTypeSQNODE_ANALOG_POWER_FAULT,
MEIEventTypeSQNODE_USER_FAULT,
MEIEventTypeSQNODE_NODE_FAILURE,

sqNode	a handle to a SynqNet node object
eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiControlEventReset | mpiMotionEventReset | mpiMotorEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | mpiAxisEventReset

Event Notification Methods

meiSqNodeMemory

Declaration

```
long meiSqNodeMemory(MEISqNode     node,
                     void          **memory);
```

Required Header: stdmei.h

Description

meiSqNodeMemory writes an address (that can be used to access SqNode memory) to the contents of memory. This address (or an address calculated from it) can be passed as the src argument to mpiSqNodeMemoryGet(...) or the dst argument to mpiSqNodeMemorySet(...).

node	a handle to a SynqNet node object
****memory**	a pointer to an SqNode memory address.

Return Values	
MPIMessageOK	

See Also

meiSqNodeMemoryGet | meiSqNodeMemorySet

meiSqNodeMemoryGet

Declaration

```
long meiSqNodeMemoryGet(MEISqNode     node,
                        void          *dst,
                        const void    *src,
                        long          count);
```

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

Description

meiSqNodeMemoryGet reads count bytes of an SqNode's memory, starting from address ***src*** and writes it to application memory, starting at address ***dst***.

node	a handle to a SynqNet node object
***dst**	pointer to the destination address in application memory
***src**	pointer to the source address in SqNode memory
count	number of bytes to copy

Return Values	
MPIMessageOK	

See Also

meiSqNodeMemory | meiSqNodeMemorySet

meiSqNodeMemorySet

Declaration

```
long meiSqNodeMemorySet(MEISqNode    node,
                        void         *dst,
                        const  void  *src,
                        long         count);
```

Required Header: stdmei.h

Description

meiSqNodeMemorySet reads count bytes of application memory, starting from address ***src*** and writes it to an SqNode's memory, starting at address ***dst***.

node	a handle to a SynqNet node object
***dst**	pointer to the destination address in SqNode memory
***src**	pointer to the source address in application memory
count	number of bytes to copy

Return Values	
MPIMessageOK	

See Also

meiSqNodeMemory | meiSqNodeMemoryGet

meiSynqNetControl

Declaration

```
MPIControl meiSqNodeControl(MEISqNode node);
```

Required Header: stdmei.h

Description

meiSqNodeControl returns a handle to the control object associated with the SqNode object.

node	a handle to a SynqNet node object

Return Values	
MPIControl	a handle to a control object
MPIHandleVOID	if node is not valid

See Also

meiSqNodeCreate | mpiControlCreate

meiSqNodeNumber

Declaration

```
long meiSqNodeNumber(MEISqNode    node,
                     long         *number);
```

Required Header: stdmei.h

Description

meiSqNodeNumber reads the index of a SynqNet ***node*** and writes it into the contents of a long pointed to by ***number***. Each SqNode associated with a controller is indexed by a identification number (0, 1, 2, etc.).

node	a handle to a SynqNet node object
***number**	a pointer to the index of a SynqNet node.

Return Values	
MPIMessageOK	

See Also

meiSynqNetInfo | meiSynqNetNumber

RMBAnalogInRange

Definition

```
typedef enum {
    RMBAnalogInRange10V,
    RMBAnalogInRange5V,
    RMBAnalogInRange2_5V,
    RMBAnalogInRange1_25V,
} RMBAnalogInRange;
```

Change History: Added in the 03.02.00

Description

RMBAnalogInRange enumeration has the four analog input ranges that can be configured on RMB_10V2 nodes.

RMBAnalogInRange10V	The analog voltage range is within ±10V
RMBAnalogInRange5V	The analog voltage range is within ±5V
RMBAnalogInRange2_5V	The analog voltage range is within ±2.5V
RMBAnalogInRange1_25V	The analog voltage range is within ±1.25V

See Also

Analog Inputs on RMB Nodes

MEISqNodeChannel

Definition

```
typedef enum MEISqNodeChannel {
    MEISqNodeChannelDRIVE0,
    MEISqNodeChannelDRIVE1,
    MEISqNodeChannelDRIVE2,
    MEISqNodeChannelDRIVE3,
    MEISqNodeChannelDRIVE4,
    MEISqNodeChannelDRIVE5,
    MEISqNodeChannelDRIVE6,
    MEISqNodeChannelDRIVE7,
    MEISqNodeChannelNODE,
} MEISqNodeChannel;
```

Description

MEISqNodeChannel is an enumeration of communication interfaces to a node. All SynqNet nodes support a single NODE channel to the network interface device. SynqNet nodes may support one or more drive channels to a drive processor. DRIVE channels are indexed by an enumeration (DRIVE0, DRIVE1, DRIVE2, etc.).

MEISqNodeChannelDRIVE0	interface to drive number 0
MEISqNodeChannelDRIVE1	interface to drive number 1
MEISqNodeChannelDRIVE2	interface to drive number 2
MEISqNodeChannelDRIVE3	interface to drive number 3
MEISqNodeChannelDRIVE4	interface to drive number 4
MEISqNodeChannelDRIVE5	interface to drive number 5
MEISqNodeChannelDRIVE6	interface to drive number 6
MEISqNodeChannelDRIVE7	interface to drive number 7
MEISqNodeChannelNODE	interface to the node device

See Also

meiSqNodeDownload

MEISqNodeCmdHeader

Definition

```
typedef struct MEISqNodeCmdHeader {
    MEISqNodeChannel    channel;    /* internal node destination */
    MEISqNodeMemory     memory;
    MEISqNodeDataSize   size;
    MEISqNodeCmdType    type;       /* read/write command */
} MEISqNodeCmdHeader;
```

Description

MEISqNodeCmdHeader specifies the service command communication interface to the device, the memory region on the device to access, the data size, and type.

channel	Communication interface to a device. See MEISqNodeChannel.
memory	The memory region to access. See MEISqNodeMemory.
size	The length of data to send or receive. See MEISqNodeDataSize.
type	The service command action (read or write). See MEISqNodeCmdType.

See Also

MEISqNodeCommand

MEISqNodeCmdType

Definition

```
typedef enum MEISqNodeCmdType {
    MEISqNodeCmdTypeREAD,
    MEISqNodeCmdTypeWRITE,
} MEISqNodeCmdType;
```

Description

MEISqNodeCmdType is an enumeration of service command types to send to a node or drive.

MEISqNodeCmdTypeREAD	read data
MEISqNodeCmdTypeWRITE	write data

See Also

MEISqNodeCmdHeader

MEISqNodeCommand

Definition

```
typedef struct MEISqNodeCommand {
    MEISqNodeCmdHeader  header;
    unsigned long       address;  /* command resgisters */
    unsigned long       data;     /* command data */
} MEISqNodeCommand;
```

Description

MEISqNodeCommand specifies the service command. It includes a header structure (channel, memory, size, and type), a destination address, and the data.

header	A structure that specifies the channel, memory region, and data size. See MEISqNodeCmdHeader.
address	A memory location to read or write the data.
data	The command data to send.

See Also

MEISqNodeResponse

MEISqNodeConfig

Definition

```
typedef struct MEISqNodeConfig {
    char                                userLabel[MEIObjectLabelCharMAX
+1];
                                        /* +1 for NULL terminator */
    MEISqNodeConfigAlarm                nodeAlarm;
    MEISqNodeConfigIoAbort              ioAbort;
    MEISqNodeConfigPacketError          upStreamError;
    MEISqNodeConfigPacketError          downStreamError;
    MEISqNodeConfigUserFault            userFault;

    MEISqNodeConfigControlLatency       controlLatency;
    MEISqNodeFeedbackSecondary          feedbackSecondary
                                        [MEISqNodeMaxFEEDBACK_SECONDARY];
} MEISqNodeConfig;
```

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

Description

MEISqNodeConfig specifies the SynqNet node configurations.

userLabel	Consists of 16 characters that are used to label the sqNode object for user identification purposes. The userLabel field is NOT used by the controller.
nodeAlarm	A structure to configure a SynqNet node's trigger conditions for the Node Alarm output bit. The node alarm circuit is node specific, but is intended to notify users when the node has a problem. The nodeAlarm occurs on an ioAbort, DedicatedInAMP_FAULT (one per motor/drive) or an FPGA fails to operate with run-time code. See MEISqNodeConfigIoAbort and MEISqNodeConfigNodeAlarm for the trigger configurations.
ioAbort	A structure to configure a SynqNet node's trigger conditions for an I/O Abort action. When an ioAbort is triggered, the SynqNet node's outputs are disabled (set to the power-on condition). **NOTE**: The outputs are disabled on all SynqNet partner nodes, with the exception of the remote motion block's (rmb's) user outputs. For the rmb only, a user limit must be configured to disable the rmb's user output when an ioAbort occurs. See MEISqNodeConfigIoAbort for the trigger configurations.

upStreamError	A structure used to configure the fault and failure limits for the upstream SynqNet packets. The controller keeps track of how many bad packets are received from the Node and performs the appropriate actions when the fault and fail limits are reached. See MEISqNodeConfigPacketError for appropriate ranges and resulting actions. **NOTE**: Saving the upStreamError values to non-volatile flash memory is currently not supported. These values need to be set after each controller reset or power on.
downStreamError	A structure used to configure the fault and failure limits for the downstream SynqNet packets. The node keeps track of how many bad packets are received from the controller and performs the appropriate actions when the fault and fail limits are reached. See MEISqNodeConfigPacketError for appropriate ranges and resulting actions.
userFault	A structure to configure the trigger conditions for a SynqNet node user fault. When a user fault is triggered, a node ioAbort and/or an action on each motor will occur. See MEISqNodeConfigUserFault for the trigger configurations.
feedbackSecondary	A structure to configure the secondary encoder resources on the node. See MEISqNodeFeedbackSecondary for more information.
controlLatency	MEISqNodeConfigControlLatency structure to configure the minimum and maximum control latency limits.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet | MEISqNodeConfigPacketError

MEISqNodeConfigAlarm

Definition

```
typedef struct MEISqNodeConfigAlarm {
    unsigned long mask;  /* One bit per drive/motor. Triggered by
                            the MEIMotorDedicatedInAMP_FAULT input. */
    MPI_BOOL      notCyclicEnable;  /* allow nodeAlarm to be asserted
when
                                       the node is not in cyclic mode */
    MPI_BOOL      ioAbortEnable;  /* allow ioAbort to assert nodeAlarm */
    MPI_BOOL      ioFaultEnable;  /* allow ioFault to assert nodeAlarm */
} MEISqNodeConfigAlarm;
```

Change History: Modified in the 03.04.00.

Description

MEISqNodeConfigAlarm specifies the input trigger for the SynqNet node alarm output. The input triggers are the MEIMotorDedicatedInAMP_FAULT bits for each motor/drive interface.

mask	Each bit in the mask represents a motor or drive interface. For example, a value of 0x3 will trigger the node alarm output when either motor 0's OR motor 1's MEIMotorDedicatedInAMP_FAULT bit is TRUE.
notCyclicEnable	This Boolean variable is used to specify whether or not a node can receive an alarm when it is not in cyclic mode. TRUE = node alarm can be asserted in any mode. FALSE = node alarm can only be asserted in cyclic mode.
ioAbortEnable	This Boolean variable is used to specify the effect an I/O abort will have on the node alarm output. TRUE = an I/O abort will trigger a node alarm. FALSE = an I/O abort will not trigger a node alarm.
ioFaultEnable	This Boolean variable is used to specify the effect an I/O fault will have on the node alarm output. TRUE = an I/O fault will trigger a node alarm. FALSE = an I/O fault will not trigger a node alarm.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet

MEISqNodeConfigControlLatency

Definition

```
typedef struct MEISqNodeConfigControlLatency {
   double minimum;  /* uS  - if > 0, this will insure
                       a minimum latency value */
   double maximum;  /* uS  - if > 0, an error will occur
                       if this value is exceeded */
} MEISqNodeConfigControlLatency;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeConfigControlLatency is a structure that contains boundary values that ensure that the node's Control Latency falls within a given range. The SynqNet configuration can be read with meiSqNodeConfigGet(...) and can be written with meiSqNodeConfigSet(...).

minimum	Lower limit for the control latency configuration. This will ensure that a minimum node control latency will be used in the SynqNet timing calculations. Set to zero to disable minimum node control latency.
maximum	Upper limit for the control latency configuration. This will force mpiControlInit(...) or meiSynqNetInit(...) to return MEISynqNetMessageNODE_LATENCY_EXCEEDED if this value is exceeded. Set to zero to disable maximum node control latency check.

See Also

MEISqNodeConfig | meiSqNodeConfigSet | meiSqNodeConfigGet | MEISynqNetTiming

Node Control Latency | Cable Length Discovery Uncertainty Factor | SynqNet Cable Length

MEISqNodeConfigIoAbort

Definition

```
typedef struct MEISqNodeConfigIoAbort {
  MEISqNodeConfigTrigger  synqLost;    /* communication error */
  MEISqNodeConfigTrigger  nodeDisable; /* external input */
  MEISqNodeConfigTrigger  powerFault;  /* analog power failure */
  MPI_BOOL                userFault;   /* TRUE = user fault causes ioabort */
} MEISqNodeConfigIoAbort;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeConfigIoAbort specifies the SynqNet node configurations to generate an I/O Abort action.

When an ioAbort is triggered, the SynqNet node's outputs are disabled (set to the power-on condition) and all axes on motion supervisors associated with the node are aborted and enter the error state.

NOTE: The outputs are disabled on all SynqNet partner nodes, with the exception of the remote motion block's (rmb's) user outputs. For the rmb only, a user limit must be configured to disable the rmb's user output when an ioAbort occurs.

When the I/O Abort conditions are cleared, the states of the axes may be cleared with a call to mpiMotionAction(..., MPIActionRESET). The IoAbort is triggered when any one or more of the following enabled configurations occur.

synqLost	Occurs when a SynqNet node drops out of SYNQ (cyclic) mode to SYNQ_LOST mode. See MEISqNodeConfigTrigger.
nodeDisable	An input bit to the SynqNet node. The node disable circuit is node specific, but is intended to shutdown the node via the IoAbort. See MEISqNodeConfigTrigger.
powerFault	An input bit to the SynqNet node. The power fault circuit is node specific, but is usually connected to an analog power monitor. Typically, when the DAC power or other analog component power is either too high or drops below a threshold, the power fault is triggered. Please see the node/drive manufacturer's documentation for details. See MEISqNodeConfigTrigger.
userFault	A user configurable trigger condition. A value of TRUE enables the trigger, FALSE disables the trigger.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet | mpiMotionAction

MEISqNodeConfigPacketError

Definition

```
typedef struct MEISqNodeConfigPacketError {
    long faultLimit;    /* 1 - 255 */
    long failLimit;     /* 1 - 255 */
} MEISqNodeConfigPacketError;
```

Description

MEISqNodeConfigPacketError specifies the limit conditions for SynqNet node packet rate errors.

faultLimit	Packet error rate limit to generate a fault. When the faultLimit is reached, the node will attempt to recover by switching the port used for data transmission. Valid range is 1 to 255. The default value is 12, which allows for a normal packet rate. The value saturates at 255.
failLimit	Packet error rate limit to generate a failure. When the failLimit is reached, the node will drop to the SYNQ_LOST state and disable its outputs. Valid range is 1 to 255. The default value is 12, which allows for a normal packet rate. The value saturates at 255.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet

MEISqNodeConfigTrigger

Definition

```
typedef struct MEISqNodeConfigTrigger {
    MPI_BOOL   enable;
    MPI_BOOL   invert;
} MEISqNodeConfigTrigger;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeConfigTrigger specifies trigger configurations.

enable	Enables or disables the trigger. A value of TRUE enables the trigger, FALSE disables the trigger.
invert	Normal or inverted trigger polarity. A value of FALSE indicates normal polarity, TRUE indicates inverted polarity.

See Also

MEISqNodeConfigIoAbort

MEISqNodeConfigUserFault

Definition

```
typedef struct MEISqNodeConfigUserFault {
    long            *addr;    /* firmware addr */
    unsigned long   mask;
    unsigned long   pattern;
} MEISqNodeConfigUserFault;
```

Description

MEISqNodeConfigUserFault specifies the trigger conditions for a user defined input. The trigger condition can be configured for any controller address. When the masked value at the specified addr matches the pattern, the user fault is active. The user fault triggers a SynqNet node IoAbort if the userFault flag in MEISqNodeConfigIoAbort{} is enabled. The user fault also triggers an action for all the motors associated with the node. The userFaultAction is specified in the MEIMotorConfig{} structure.

***addr**	A pointer to a controller address.
mask	A bit mask ANDed with the value at the controller address.
pattern	A bit pattern compared to the masked value at the controller address. When the masked value equals the pattern, the user trigger is TRUE.

See Also

MEISqNodeConfigIoAbort | MEIMotorConfig | meiSqNodeConfigGet | meiSqNodeConfigSet

MEISqNodeDataSize

Definition

```
typedef enum MEISqNodeDataSize {  /* read/write data width */
    MEISqNodeDataSize8BIT,
    MEISqNodeDataSize16BIT,
    MEISqNodeDataSize24BIT,
    MEISqNodeDataSize32BIT,
} MEISqNodeDataSize
```

Description

MEISqNodeDataSize is an enumeration of service command data lengths. The data length is in units of bits.

MEISqNodeDataSize8BIT	8 bit data length
MEISqNodeDataSize16BIT	16 bit data length
MEISqNodeDataSize24BIT	24 bit data length
MEISqNodeDataSize32BIT	32 bit data length

See Also

meiSqNodeCommand | MEISqNodeCmdHeader

MEISqNodeDownloadParams

Definition

```
typedef struct MEISqNodeDownloadParams {
    char                *filename;
    MEISqNodeChannel    channel;
    MEISqNodeCallback   callback;
} MEISqNodeDownloadParams;
```

Description

MEISqNodeDownloadParams specifies the parameters for downloading a binary image to a SynqNet node.

***filename**	A pointer to a file name. The file contains a header and binary code/data. Files are node/drive specific. Please see the Node Binary Files: Product Table or the drive manufacturer's documentation for the drive binary files.
channel	A communication interface to a node's logic device or drive processor. See MEISqNodeChannel.
callback	A pointer to a callback function, to monitor the download progress. See MEISqNodeCallback.

See Also

meiSqNodeDownload

MEISqNodeDriveInfo

Definition

```
typedef struct MEISqNodeDriveInfo {
    char    firmwareVersion[MEISqNodeDriveParamMAX_STRING_LENGTH];
} MEISqNodeDriveInfo;
```

Description

MEISqNodeDriveInfo contains information about a specified drive.

firmwareVersion	A string containing drive firmware version information that is retrieved from the Drive Processor on the Node.

See Also

meiSqNodeDownload

MEISqNodeDriveMonitor

Definition

```
typedef struct MEISqNodeDriveMonitor {
    MEISqNodeDriveMonitorDataType    type;
    MEISqNodeDriveMonitorData        data;
} MEISqNodeDriveMonitor;
```

Description

MEISqNodeDriveMonitor specifies the data to be placed in the monitor field by the drive.

type	The drive data is selected by its type. See MEISqNodeDriveMonitorDataType.
data	The location of the drive data. See MEISqNodeDriveMonitorData.

See Also

MEISqNodeMonitorValue | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorConfig

Definition

```
typedef struct MEISqNodeDriveMonitorConfig {
    MEISqNodeDriveMonitor    monitorA;
    MEISqNodeDriveMonitor    monitorB;
    MEISqNodeDriveMonitor    monitorC;
}MEISqNodeDriveMonitorConfig;
```

Description

MEISqNodeDriveMonitorConfig specifies the configuration for the drive monitor fields.

monitorA	configuration for drive monitor A
monitorB	configuration for drive monitor B
monitorC	configuration for drive monitor C

See Also

MEISqNodeDriveMonitor | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorData

Definition

```
typedef union {
    long    index;     /* the values for these
                       parameters are drive specific    */
    long    address;   /* and can be found in the
                       appropriate drive modules */
} MEISqNodeDriveMonitorData;
```

Description

MEISqNodeDriveMonitorData specifies the location of the monitor data. Drive data can be specified by either an index or an address. The location is drive specific. Please see the drive manufacturer's documentation.

index	A drive specific value to select a monitor data field from a table.
address	A drive specific memory address to select the monitor data.

See Also

meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorDataType

Definition

```
typedef enum MEISqNodeDriveMonitorDataType {
    MEISqNodeDriveMonitorDataTypeINDEX,
    MEISqNodeDriveMonitorDataTypeADDRESS,
    MEISqNodeDriveMonitorDataTypeFIXED_CONFIG,
} MEISqNodeDriveMonitorDataType;
```

Change History: Modified in the 03.04.00; added MEISqNodeDriveMonitorDataTypeFIXED_CONFIG.

Description

MEISqNodeDriveMonitorDataType is an enumeration of monitor data selection types.

MEISqNodeDriveMonitorDataTypeINDEX	Select monitor data using an index to a table.
MEISqNodeDriveMonitorDataTypeADDRESS	Select monitor data using an address.
MEISqNodeDriveMonitorDataTypeFIXED_CONFIG	Monitor data is not selectable. The monitor configuration is hard coded by the node/drive.

See Also

meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorInfo

Definition

```
typedef struct MEISqNodeDriveMonitorInfo{
   long                                monitorCount;
   MEISqNodeMonitorLocation            location[MEISqNodeMonitorValueIndexLAST];
   long                                configCount;
   const MEISqNodeMonitorConfigInfo    *configInfo; /* array of configuration
                                                       information of size
                                                       configCount, this
                                                       info is drive specific */
} MEISqNodeDriveMonitorInfo;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeDriveMonitorInfo is a structure that contains information about the monitor fields. This structure is useful for determining the number of supported monitor fields, their locations and how to decode them. It is also useful for determining the number of configurable monitor field data inputs and the reading the monitor data-specific information. The monitor field data is SynqNet drive-specific. Not all drives support monitors.

NOTE: SynqNet RMBs (Remote Motion Blocks) or I/O nodes do not support monitors.

monitorCount	the number of supported monitor fields.
location	an array of MEISqNodeMonitorLocation structures, with length equal to the monitorCount. Each MEISqNodeMonitorLocation structure contains information to decode each monitor field.
configCount	the number of configurable monitor field data inputs.
***configInfo**	a pointer to an array of MEISqNodeMonitorConfigInfo structures, with length equal to **configCount**. Each MEISqNodeMonitorConfigInfo structure contains information about the configurable monitor field data.

Sample Code

```
MEISqNodeDriveMonitorInfo monInfo;

returnValue =
   meiSqNodeDriveMonitorInfo(sqNode,
                             driveIndex,
                             &monInfo);

if(returnValue == MPIMessageOK && monInfo.monitorCount) {
   printf("Monitor Information:\n");
   printf("\rAvailable Monitors A");
   if(monInfo.monitorCount > 1) {
      printf(", B");
   }

   if(monInfo.monitorCount > 2) {
      printf(", C");
   }

   printf("\n");
}

if(monInfo.configCount) {
   printf("\nMonitor Configurations:\n");
   for(i = 0; i < monInfo.configCount; i++) {
       if(monInfo.configInfo[i].type == MEISqNodeDriveMonitorDataTypeINDEX)
          printf("\r\r Index %d", monInfo.configInfo[i].value);
       }
       else {
          printf("\r\r Drive address 0x%x", monInfo.configInfo[i].value);
       }
       printf(" %s\n",monInfo.configInfo[i].description);
   }
}
```

See Also

MEISqNodeMonitorConfigInfo | MEISqNodeMonitorLocation | MEISqNodeMonitorValueIndex | meiSqNodeDriveMonitorInfo

MEISqNodeDriveParamCallback

Definition

```
typedef void (*MEISqNodeDriveParamCallback)
             (MEISqNodeDriveParamCallbackType  type,
              const char                       *name,
              long                             number,
              const char                       *value);
```

Change History: Modified in the 03.03.00

Description

In the **MEISqNodeDriveParamCallback** structure, the function's pointer type defines a function that can be passed to the meiSqNodeDriveMapParamFileSet(...) function. The meiSqNodeDriveMapParamFileSet(...) function will call this type of function to report progress or warnings. A NULL value for the callback pointer will disable the callback feature.

type	the type of event that caused the callback function to be called.
name	name of the drive parameter.
number	drive parameter index.
value	the value of the drive parameter.

See Also

meiSqNodeDriveMapParamFileSet

MEISqNodeDriveParamCallbackType

Definition

```
typedef enum {
    MEISqNodeDriveParamCallbackTypeCHANGED,
    MEISqNodeDriveParamCallbackTypeSET_FAILED,
    MEISqNodeDriveParamCallbackTypeSKIPPING_READ_ONLY,
} MEISqNodeDriveParamCallbackType;
```

Change History: Modified in the 03.03.00

Description

The **MEISqNodeDriveParamCallbackType** enumeration is used by the MEISqNodeDriveParamCallback function to describe the type of event that caused the callback function to be called.

MEISqNodeDriveParamCallbackTypeCHANGED	This indicates that the new drive parameter value is different to the current parameter value.
MEISqNodeDriveParamCallbackTypeSET_FAILED	The drive parameter that was set has failed.
MEISqNodeDriveParamCallbackTypeSKIPPING_READ_ONLY	This indicates that the drive paramerter could not be set because it has read only attributes.

See Also

MEISqNodeDriveParamCallback

MEISqNodeFeedbackSecondary

Definition

```
typedef struct MEISqNodeFeedbackSecondary {
    long    motorIndex;
} MEISqNodeFeedbackSecondary;
```

Description

MEISqNodeFeedbackSecondary allows for configuration of the secondary feedback resources on a SynqNet node.

motorIndex	Indicates motorIndex on the node to which the secondary feedback resource is mapped. This value is MEISqNodeNOT_AVAILABLE if the secondary feedback resource does not exist on the node hardware

See Also

MEISqNodeConfig

MEISqNodeFilename

Definition

```
typedef struct MEISqNodeFileName{
    char  fileName[MEISqNodeFILENAME_MAX];
}MEISqNodeFileName;
```

Description

MEISqNodeFilename is used in methods that retrieve filenames from the MPI.

fileName	String containing the name of an SqNode image file.

See Also

MEISqNodeFpgaType

Definition

```
typedef enum MEISqNodeFpgaType {
    MEISqNodeFpgaTypeBOOT,
    MEISqNodeFpgaTypeRUN_TIME,
} MEISqNodeFpgaType
```

Description

MEISqNodeFpgaType is an enumeration of FPGA types.

MEISqNodeFpgaTypeBOOT	The FPGA is operating with a boot image. The boot image only supports basic SynqNet communication. Use meiSqNodeDownload(...) to download the runtime image to the SynqNet node.
MEISqNodeFpgaTypeRUN_TIME	The FPGA is operating with a runtime image.

See Also

meiSqNodeInfo | MEISqNodeInfoFpga | meiSqNodeDownload

MEISqNodeInfo

Definition

```
typedef struct MEISqNodeInfo {
    long                    motorCount;
    long                    driveCount;
    long                    motorOffset;
    long                    feedbackSecondaryCount;
    MEISqNodeInfoId         id;
    MEISqNodeInfoFpga       fpga;
    MEISqNodeInfoNetwork    network;
    MEISqNodeInfoIo         io;
} MEISqNodeInfo;
```

Description

MEISqNodeInfo contains static data stored for the SynqNet node. The motor objects are indexed sequentially across all the SynqNet nodes associated with each network. Each motor on a controller has a unique number.

motorCount	The number of motors that the SynqNet node supports.
driveCount	The number of drives interfaces that the SynqNet node supports.
motorOffset	The starting number for the first motor on the SynqNet node.
feedbackSecondaryCount	The number of auxillary feedbacks on the node.
id	A structure that contains identification data for the SynqNet node. See MEISqNodeInfoId.
fpga	A structure that contains identification data for the SynqNet node FPGA. See MEISqNodeInfoFpga.
network	A structure that contains network interface information for the SynqNet node. See MEISqNodeInfoNetwork.
io	A structure that returns how many of each type of node I/O this node supports.

See Also

meiSqNodeInfo

MEISqNodeInfoId

Definition

```
typedef struct MEISqNodeInfoId {
    unsigned  long   nodeType;   /* product/mfg code   */
    char             *nodeName;  /* product/mfg string */
    unsigned  long   option;     /* product option code*/
    unsigned  long   switchId;   /* rotary switch id   */
    unsigned  long   unique;     /* unique id code     */

    MPI_BOOL         exactMatch; /* TRUE/FALSE         */

    char             serialNumber[MEISqNodeID_CHAR_MAX];
    char             modelNumber[MEISqNodeID_CHAR_MAX];
    char             manufacturerData[MEISqNodeManufacturerDATA_CHAR_MAX];
} MEISqNodeInfoId;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeInfoId contains identification data for the SynqNet node.

All nodes by all manufacturers will have ***nodeType*** and ***unique*** numbers that should generate a unique identification for each node on the SynqNet network.. Although some node manufacturers may opt to leave the ***serialNumber*** and ***modelNumber*** fields blank, you can still identify and distinguish a node by comparing the ***nodeType*** and ***unique*** numbers. The ***nodeType*** number is also represented by a unique text string *nodeName*.

nodeType	A 32 bit value that identifies the node hardware. The upper 16 bits represent the manufacturer of the SynqNet node hardware. Each manufacturer has a unique value. The lower 16 bits represent the SynqNet node product type. The SynqNet node manufacturer determines a unique value to track a product series. Typically, the node type value is displayed in hex.
***nodeName**	A string that represents the SynqNet nodeType. The nodeName string matches the name of the SqNodeLib node specific header file.
option	The product option code within a product series.
switchId	If a node/drive have an physical address switch on its faceplate, switchId will contain the value to which the switch is set. If an ID switch is not supported by a node, this value will be set to -1 (0xFFFFFFFF).

unique	A 32 bit value that identifies the node. It is an unsigned long. The SynqNet node manufacturer determines this unique value to track a single product. This is useful to determine when individual nodes of the same type are switched or replaced on a SynqNet network. **NOTE**: It is possible for a manufacturer to use the same unique identification number for two nodes of different models. The combination of SqNode.Name (or nodeType) and SqNode.UniqueId will be unique for any given code.
exactMatch	A string that tells you if the node is running under a matched or unmatched classification. The value of meiSqNodeInfo.id.exactMatch is TRUE when all ID components have been matched to a supported configuration. The value is FALSE when running with a default (unmatched) configuration.
serialNumber	A string that represents the SynqNet node serial number. For a given node type, the serial number is unique. The SynqNet node manufacturer determines the serial number to track an individual unit.
modelNumber	A string that represents the SynqNet node model number. The SynqNet node manufacturer determines the model number.
manfacturerData	A string containing Manufacturer-specific data which is stored on the node at time of production.

See Also

meiSqNodeInfo | MEISqNodeInfoFpga

MEISqNodeInfoIo

Definition

```
typedef struct MEISqNodeInfoIo {
   long  digitalInCount;
   long  digitalOutCount;
   long  analogInCount;
   long  analogOutCount;
   long  segmentCount;
   long  maxWait;
} MEISqNodeInfoIo;
```

Change History: Modified in the 03.02.00

Description

MEISqNodeInfoIo lists the number of digital and analog inputs that are supported by a SynqNet node.

digitalInCount	The number of digital inputs on a SynqNet node.
digitalOutCount	The number of digital outputs on a SynqNet node.
analogInCount	The number of analog inputs on a SynqNet node.
analogOutCount	The number of analog outputs on a SynqNet node.
segmentCount	The total number of segments on a SynqNet node.
maxWait	This is the maximum amount of time between when the output bit is set in software and the hardware state takes effect. See Output Waits.

See Also

meiSqNodeInfo | meiSqNodeSegmentDigitalOutGet | meiSqNodeSegmentDigitalOutSet | meiSqNodeSegmentAnalogOutGet | meiSqNodeSegmentAnalogOutSet | What I/O does a nodes support?

MEISqNodeInfoFpga

Definition

```
typedef struct MEISqNodeInfoFpga {
    MEISqNodeFpgaType          type;
    unsigned           long    vendorDevice;
    unsigned           long    version;
    unsigned           long    branchVersion;
    MPI_BOOL                   defaultVersion; /* TRUE/FALSE */
} MEISqNodeInfoFpga;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeInfoFpga contains identification data for the SynqNet node FPGA.

type	The FPGA type. See MEISqNodeFpgaType.
vendorDevice	A 32 bit value that identifies the FPGA image. The upper 16 bits represent the manufacturer of the SynqNet node network interface device. Each manufacturer has a unique vendor value. The lower 16 bits represent the SynqNet node network interface component. The device is typically an FPGA (could be an ASIC). If the device is an FPGA, the vendorDevice information is stored in the FPGA binary image. Each device for a particular vendor has a unique device value. Typically, the vendorDevice value is displayed in hexadecimal format.
version	A 32-bit value that represents the revision of the device. The upper 16 bits (SqMac Version), represent the SynqNet network interface revision. The lower 16 bits (Node Version), represent the device revision. Typically, the version value is displayed in hexadecimal format. Ex: 0x02400344 SqMac Version: 0240 Node Version: 0344

branchVersion	A 32-bit value that identifies the branch from an existing version (MajorMinor) or from another Branch. The upper 16 bits (SqMac Branch Version), represent the SynqNet network interface branch revision. The lower 16 bits (Node Branch Version), represent the device branch revision. Ex: 0x01010102 SqMac Branch Version: 0101 Node Branch Version: 0102 **NOTE**: The FPGA branch version was added in FPGAs with SqMac version 0x0230 (and greater). If the SqMac version is 0x0230 (or greater), then the branch version will properly show the revision information. If the SqMac version is less than 0x0230, then the branch version will show 0xFF.
defaultVersion	Indicates if the default version of the SqNode FPGA image is loaded on this node. The defaultVersion defines the version of the SynqNet node FPGA image that was built and tested with the current version of the MPI.

See Also

meiSqNodeInfo | MEISqNodeInfoId | MPI/SynqNet FPGA Compatibility Check

MEISqNodeInfoNetwork

Definition

```
typedef struct MEISqNodeInfoNetwork {
    long    number;
    long    inPorts;
    long    outPorts;
} MEISqNodeInfoNetwork;
```

Description

MEISqNodeInfoNetwork structure contains information about the SynqNet node's network interface.

Remarks

The labeling convention for IN and OUT ports is for convenience. The hardware ports are identical. During SynqNet initialization, the node are discovered based on the OUT to IN port connections.

number	An index to a SynqNet network associated with a controller.
inPorts	The number of SynqNet IN port network interfaces.
outPorts	The number of SynqNet OUT port network interfaces.

See Also

meiSqNodeInfo

MEISqNodeMemory

Definition

```
typedef enum MEISqNodeMemory {
    MEISqNodeMemoryDATA,        /* node/drive processor RAM */
    MEISqNodeMemoryPROGRAM,     /* drive processor program memory */
    MEISqNodeMemoryIO,          /* drive I/O memory */
    MEISqNodeMemoryDRIVE,       /* direct command to drive */
} MEISqNodeMemory;
```

Description

MEISqNodeMemory is an enumeration of drive region types to access with a service command.

MEISqNodeMemoryDATA	node/drive processor data memory
MEISqNodeMemoryPROGRAM	drive processor program memory
MEISqNodeMemoryIO	drive I/O memory
MEISqNodeMemoryDRIVE	direct command to drive processor

See Also

MEISqNodeCmdHeader | meiSqNodeCommand

MEISqNodeMessage

Definition

```
typedef enum {
    MEISqNodeMessageINVALID,
    MEISqNodeMessageNODE_INVALID,
    MEISqNodeMessageSTATE_ERROR,
    MEISqNodeMessageCONFIG_NETWORK_MISMATCH,
    MEISqNodeMessageMAP_CONFIG_MISMATCH,
    MEISqNodeMessageNOT_IN_CONFIG_FILE,
    MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID,

    MEISqNodeMessageRESPONSE_TIMEOUT,
    MEISqNodeMessageREADY_TIMEOUT,
    MEISqNodeMessageSRVC_ERROR,
    MEISqNodeMessageSRVC_UNSUPPORTED,
    MEISqNodeMessageSRVC_CHANNEL_INVALID,

    MEISqNodeMessageCMD_NOT_SUPPORTED,
    MEISqNodeMessageDISCOVERY_FAILURE,
    MEISqNodeMessageDISPATCH_ERROR,
    MEISqNodeMessageINIT_FAILURE,
    MEISqNodeMessageINTERFACE_ERROR1,
    MEISqNodeMessageFILE_NODE_MISMATCH,
    MEISqNodeMessageFILE_INVALID,
    MEISqNodeMessageINVALID_HEADER,
    MEISqNodeMessageDOWNLOAD_FAIL,
    MEISqNodeMessageVERIFY_FAIL,
    MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED,
    MEISqNodeMessageVERIFY_NOT_SUPPORTED,
    MEISqNodeMessageBOOT_ROM_INVALID,
    MEISqNodeMessageINVALID_TABLE,
    MEISqNodeMessageINVALID_STR_LEN,
    MEISqNodeMessageFEEDBACK_MAP_INVAILD,
    MEISqNodeMessageNODE_FAILURE,
    MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT,

    MEISqNodeMessageIO_MODULE_INCOMPATIBILITY,
    MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED,
    MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED,
    MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED,
    MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED,
    MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED,

    MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT,
```

```
    MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MANY_SLICES,
    MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH,
    MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT,
    MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH,
    MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR,
    MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR,
    MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE,
    MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE,
    MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE,
    MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED,
    MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE,
    MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE,
    MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT,
    MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE,
    MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA,
    MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED,
    MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA,
    MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST,
    MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER,
    MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE,
    MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE,
    MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT,
    MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT,
    MEISqNodeMessageIO_SLICE_EEPROM_FORMAT,
    MEISqNodeMessageIO_SLICE_TOO_MUCH_IO,

    MEISqNodeMessageBOOT_FILE_NOT_FOUND,
    MEISqNodeMessagePARAM_READ_ONLY,
    MEISqNodeMessagePARAM_LOCKED,
    MEISqNodeMessageMONITOR_INDEX,
    MEISqNodeMessageMONITOR_ADDRESS,
    MEISqNodeMessageMONITOR_NA,

    MEISqNodeMessageCOMMUTATION_INIT_FAILURE,
    MEISqNodeMessagePOSITION_CLEAR_FAILURE,
    MEISqNodeMessageNODE_EEPROM_SET,
} MEISqNodeMessage;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00. Modified in the 03.02.00

Description

MEISqNodeMessage is an enumeration of SynqNet node error messages that can be returned by the MPI library.

MEISqNodeMessageINVALID

The SqNode type is out of range. This message code is returned by SynqNet node methods if the node type is not a member of the SQNodeLibNodeType enumeration.

MEISqNodeMessageNODE_INVALID

The SynqNet Node number is out of range. This message code is returned if the given node number is less than zero, or greater than or equal to MEISynqNetMaxNODE_COUNT.

MEISqNodeMessageSTATE_ERROR

Some methods can only be executed in SYNQ state. This message will be returned when the network is not in the expected network state (i.e ASYNQ state).

MEISqNodeMessageCONFIG_NETWORK_MISMATCH

The type of map file specified in meiSqNodeDriveMapParamFileSet(...) does not match the type of drive found on the network.

MEISqNodeMessageMAP_CONFIG_MISMATCH

The parameter name or number specified in meiSqNodeDriveMapParamFileSet(...) was not valid for the specified drive.

MEISqNodeMessageNOT_IN_CONFIG_FILE

The parameter name or number specified in meiSqNodeDriveMapParamFileSet(...) was not found.

MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID

A file with an incorrect format was used in meiSqNodeDriveMapParamFileSet(...).

MEISqNodeMessageRESPONSE_TIMEOUT

The drive/node did not respond to the service command issued in a reasonable amount of time.

MEISqNodeMessageREADY_TIMEOUT

The node/drive did not complete the hand shaking for the service command.

MEISqNodeMessageSRVC_ERROR

There was an error in carrying out the service command issued.

MEISqNodeMessageSRVC_UNSUPPORTED

The service command issued is either not supported or recognized by the drive.

MEISqNodeMessageSRVC_CHANNEL_INVALID

Invalid service channel specified. See MEISqNodeCmdHeader.

MEISqNodeMessageCMD_NOT_SUPPORTED

The service command is not supported by the node.

MEISqNodeMessageDISCOVERY_FAILURE

Unable to discover node rescources.

MEISqNodeMessageDISPATCH_ERROR

Is the default error code returned when a node specific routine has failed. Check the node FPGA version to verify whether or not it is correct.

MEISqNodeMessageINIT_FAILURE

A node specific initialization routine was unable to successfully complete its routine. Verify that the node FPGA is the default version for your MPI version. See MEISqNodeInfoFpga.

MEISqNodeMessageINTERFACE_ERROR1

This is an outdated node, which does not support the current discovery routine.

MEISqNodeMessageFILE_NODE_MISMATCH

Node type does not match the file provided for download.

MEISqNodeMessageFILE_INVALID

The file provided for download was not found or was corrupted.

MEISqNodeMessageINVALID_HEADER

The header information in the download image is invalid. Please verify firmware file to be correct and retry download. If firmware file is correct please contact firmware manufacturer.

MEISqNodeMessageDOWNLOAD_FAIL

Node firmware download failed. Verify that the firmware file is correct and retry the download.

NOTE: A network reset my be required.

MEISqNodeMessageVERIFY_FAIL

The node FPGA firmware does not match the FPGA image file.

MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED

The downloading of the node firmware (FPGA) image is not supported for this node.

MEISqNodeMessageVERIFY_NOT_SUPPORTED

The Node specified for verification does not support the upload of the FPGA image. Therefore, the image cannot be verified.

MEISqNodeMessageBOOT_ROM_INVALID

The SqNode Boot Rom identification or version is not recognized by the MPI.

MEISqNodeMessageINVALID_TABLE

Invalid resource table in node module. This is a fatal error within the MPI. Please verify MPI and node FPGA versions to be correct and then contact MEI's Technical Support.

MEISqNodeMessageINVALID_STR_LEN

An attempt to write information to the node has failed due to an invalid string length.

MEISqNodeMessageFEEDBACK_MAP_INVAILD

Returned from meiSqNodeConfigSet(...) when the given secondary encoder (*n*) is not mappable to the motor on the node specified by MEISqNodeFeedbackSecondary[n].motorIndex.

MEISqNodeMessageNODE_FAILURE

An attempt was made to access a SynqNet node that has a node failure event active. SynqNet Node Failure describes the details.

MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT

When initializing the SynqNet network one of the node requires a SynqNet packet larger that what can be supported.

MEISqNodeMessageIO_MODULE_INCOMPATIBILITY

Two modules attached to a SQID node are incompatible. This error message code is returned when initializing a SQID node. Different types of I/O module may be incompatible and will not work on the same SQID node.

MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED

The EEPROM on one of the modules attached to a SQID node has not been programmed.

MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED

The maximum number of I/O that can be supported by a SQID node has been exceeded.

MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED

When initializing an SQIO node, checks are performed to confirm that the node actually supports the correct number of I/O. This error is returned when one of these tests fails. An error will occur when the length of at least one of the inter module (SPI) buses did not match the length calculated from data held in the module EEPROMs. This error message can be returned by MPI functions that reset the SynqNet Network such as, mpiControlReset(...), mpiControlInit(...), meiSynqNetInit(...). This error can ONLY be generated by SQIO nodes. This fault can also be caused by a poor electrical connection between the SQID and the I/O modules. If the modules are firmly connected and the error persists then you will need to contact your supplier of the I/O Modules to correct the hardware.

MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED

During the initialization of the modules attached to a SQID node, the maximum current that can be drawn from the inter module 3.3V bus exceeds the allowable current.

MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED

During the initialization of the modules attached to a SQID node, the maximum current that can be drawn from the inter module 5V bus exceeds the allowable current.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT

An error was encountered while initalizing the Slice I/O node.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MAY_SLICES

Slice I/O nodes can only support 32 slices attached to the network adapter. This error is returned when more than 32 slices are detected.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH

A slice is attached to the node that is not supplied by MEI. You can only use slices supplied by MEI.

MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT

When initializing a Slice I/O node the initialization routine failed to complete within the expected time.

MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH

When attempting to restore the slice parameters to the attached slices of a Slice I/O node, the arrangement of slices did not match the expected arrangement. If you wish to clear the previous slice parameters, you can stop this error by calling meiSqNodeSegmentParamClear(...).

MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR

When attempting to access data on Slice I/O, a communication fault was detected. The message received from the slice was badly formed, errors include CRC and missing start/stop bits.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR

When attempting to access data on Slice I/O, a communication fault was detected. The slice returned too many characters when responding to this request.

MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE

When attempting to access data on Slice I/O, a communication fault was detected. The slice returned an error code.

MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE

When attempting to access data on Slice I/O, a communication fault was detected. An unknown fault was detected when accessing this slice.

MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the action requested on this data. For example, a read or write.

MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE

When attempting to access data on Slice I/O, a communication fault was detected. The slice is already in the requested mode.

MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT

When attempting to access data on Slice I/O, a communication fault was detected. The slice can not perform the requested action in its current state.

MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE

When attempting to access data on Slice I/O, a communication fault was detected. You cannot modify the data on this slice.

MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA

When attempting to access data on Slice I/O, a communication fault was detected. Not enough data was supplied to the slice for this operation.

MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA

When attempting to access data on Slice I/O, a communication fault was detected. Too much data was supplied to the slice for this operation.

MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the specified parameter.

MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE

When attempting to access data on Slice I/O, a communication fault was detected. The slice failed during a store operation.

MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE

When attempting to access data on Slice I/O, a communication fault was detected. The error code from the slice was not recognized.

MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT

When attempting to access data on Slice I/O, a communication fault was detected. The operation on the slice exceeded the timeout threshold.

MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT

When attempting to access data on Slice I/O, a communication fault was detected. The response from the slice was not formatted correctly.

MEISqNodeMessageIO_SLICE_SERVICE_EEPROM_FORMAT

The data held on the EEPROM on the network adaptor was not formatted correctly.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_IO

When starting the slice node, too many I/O slices were found. You should remove some slices to stop this error message from being generated.

MEISqNodeMessageBOOT_FILE_NOT_FOUND

The boot file "kollmorgen_ember.a00" was not found. When downloading drive images to Kollmorgen CD, DASA, and PicoDAD drives, a boot file is downloaded to the drive prior to the actual drive image. This boot file needs to be located in the same directory as the drive's image file that is provided for download.

MEISqNodeMessagePARAM_READ_ONLY

The drive parameter that the user is attempting to set is read only.

MEISqNodeMessagePARAM_LOCKED

The drive parameter that the user is attempting to set is not accessible. SelSFDParam must be set to 0, otherwise the SFD motor parameters will be used.

MEISqNodeMessageMONITOR_INDEX

Drive does not support the configuring of Monitors through indexing.

MEISqNodeMessageMONITOR_ADDRESS

Drive does not support the configuring of Monitors through addressing.

MEISqNodeMessageMONITOR_NA

The hardware support for SynqNet node monitors is not available. This message is returned by meiSqNodeDriveMonitorInfo(…), if the node/drive hardware does not support monitor fields. To avoid this problem, do not use the monitor methods if the SynqNet node hardware does not support it. The monitor field data is SynqNet drive-specific. All drives do not support monitors. SynqNet RMBs (Remote Motion Blocks) or I/O nodes do not support monitors.

MEISqNodeMessageCOMMUTATION_INIT_FAILURE

Motor commutation initialization failed. This message is returned from sgdsCommutationInit(…) or sgdzBsCommutationInit(…) if the commutation initialization fails.

MEISqNodeMessagePOSITION_CLEAR_FAILURE

SynqNet drive did not clear the position as requested. This message is returned from sgdsEncoderPositionClear(…) or sgdzBsEncoderPositionClear(…) if the SynqNet drive does not respond to the service command request to clear the feedback position.

MEISqNodeMessageNODE_EEPROM_SET

SynqNet node EERPOM was modified by the MPI. This message is returned from mpiControlInit(…) or meiSynqNetInit(…) if the Kollmorgen S300, S600 or S1800 drive's EEPROM was not properly configured at the factory. The MPI will attempt to discover the drive's configuration and set the appropriate option number in the node's EEPROM.

See Also

meiSqNodeDriveMapParamFileSet | meiSqNodeConfigSet

MEISqNodeMonitorConfigInfo

Definition

```
typedef struct MEISqNodeMonitorConfigInfo{
   MEISqNodeDriveMonitorDataType  type;
   long                           value;
   const char                     *description;
} MEISqNodeMonitorConfigInfo;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeMonitorConfigInfo is a structure that contains information about the configurable monitor field data. This structure is useful for determining the monitor selection type, its selection value, and a brief text description. The monitor field data is SynqNet drive-specific. Not all drives support monitors.

NOTE: SynqNet RMBs (Remote Motion Blocks) or I/O nodes do not support monitors.

type	Monitor field data is selected by an index (MEISqNodeDriveMonitorDataTypeINDEX) or a drive address (MEISqNodeDriveMonitorDataTypeADDRESS).
value	Monitor index or drive address to select the monitor field data.
***description**	A string to describe the monitor field data.

Sample Code

```
MEISqNodeDriveMonitorInfo monInfo;


returnValue =
   meiSqNodeDriveMonitorInfo(sqNode,
                             driveIndex,
                             &monInfo);

printf("\nMonitor Configurations:\n");
if(monInfo.configCount) {
   for(i = 0; i < monInfo.configCount; i++) {
       if(monInfo.configInfo[i].type == MEISqNodeDriveMonitorDataTypeINDEX) {
          printf("\r\r Index %d", monInfo.configInfo[i].value);
       }
       else {
          printf("\r\r Drive address 0x%x", monInfo.configInfo[i].value);
```

```
        }
        printf(" %s\n",monInfo.configInfo[i].description);
    }
}
else {
     printf("Monitors are not Configurable.\n");
     }
```

See Also

MEISqNodeDriveMonitorDataType | MEISqNodeDriveMonitorInfo | meiSqNodeDriveMonitorInfo | MEISqNodeMonitorValueIndex

MEISqNodeMonitorLocation

Definition

```
typedef struct MEISqNodeMonitorLocation {
   long            *addr;
   unsigned  long  mask;
   unsigned  long  rightShift;
} MEISqNodeMonitorLocation;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeMonitorLocation is a structure to decode the monitor field. It contains a controller address, bit mask, and bit shift information for the specific monitor field: MEISqNodeMonitorValueIndexA, MEISqNodeMonitorValueIndexB, or MEISqNodeMonitorValueIndexC.

addr	address for the drive monitor field
mask	bit mask to apply to the monitor field
rightShift	bit shift to apply to monitor field

Sample Code

Read and display the supported monitor field information.

```
MEIMotorConfig MEISqNodeDriveMonitorInfo monInfo;

returnValue =
meiSqNodeDriveMonitorInfo(sqNode,
                          driveIndex,
                          &monInfo);

if(returnValue == MPIMessageOK && monInfo.monitorCount) {
   printf("Monitor Information:\n");
   printf("\rAvailable Monitors A");
   if(monInfo.monitorCount > 1) {
   printf(", B");
   }
   if(monInfo.monitorCount > 2) {
```

```
   printf(", C");
   }
   printf("\n\nMonitor Locations:\n");
   for(i = 0; i < monInfo.monitorCount; i++) {
   printf(" %c Controller address 0x%x, mask 0x%x, right shift %d\n",
   'A'+ i, monInfo.location[i].addr, monInfo.location[i].mask,
           monInfo.location[i].rightShift);
  }
}
```

See Also

MEISqNodeMonitorConfigInfo | MEISqNodeDriveMonitorInfo | meiSqNodeDriveMonitorInfo | MEISqNodeMonitorValueIndex

MEISqNodeMonitorValue

Definition

```
typedef struct MEISqNodeMonitorValue {
    long    count;
    long    monitor[MEISqNodeMonitorValueIndexLAST];
} MEISqNodeMonitorValue;
```

Description

MEISqNodeMonitorValue contains the data for the monitor fields read by the meiSqNodeDriveMonitor (...) method.

count	The number of monitor fields read. This specifies the size of the monitor array.
monitor	An array of monitor data fields. Each field is indexed by the MEISqNodeMonitorValueIndex enumeration.

See Also

meiSqNodeDriveMonitor | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeMonitorValueIndex

Definition

```
typedef enum MEISqNodeMonitorValueIndex {
    MEISqNodeMonitorValueIndexA,
    MEISqNodeMonitorValueIndexB,
    MEISqNodeMonitorValueIndexC,
} MEISqNodeMonitorValueIndex;
```

Change History: Modified in the 03.04.00.

Description

MEISqNodeMonitorValueIndex is an enumeration of indices to node monitor values.

MEISqNodeMonitorValueIndexA	Index to node monitor value A.
MEISqNodeMonitorValueIndexB	Index to node monitor value B.
MEISqNodeMonitorValueIndexC	Index to node monitor value C.

See Also

meiSqNodeDriveMonitor | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeResponse

Definition

```
typedef struct MEISqNodeResponse {
    unsigned long    data;    /* response data */
} MEISqNodeResponse;
```

Description

MEISqNodeResponse contains the service command response data.

data	The response information from a service command. The data field is only valid for MEISqNodeCmdTypeREAD command types.

See Also

meiSqNodeCommand

MEISqNodeSegmentInfo

Definition

```
#define MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH 0x20
#define MEISqNodeSegmentInfoMODEL_NAME_LENGTH    0x20
#define MEISqNodeSegmentInfoMANUFACTURER_LENGTH  0x10

typedef struct MEISqNodeSegmentInfo {
   long  id;
   long  option;
   char  serialNumber[MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH];
   char  modelName[MEISqNodeSegmentInfoMODEL_NAME_LENGTH];
   long  digitalInCount;
   long  digitalOutCount;
   long  analogInCount;
   long  analogOutCount;
   long  version;
   long  paramCount;
   long  memoryCount;
   char  manufacturerData[MEISqNodeManufacturerDATA_CHAR_MAX];
} MEISqNodeSegmentInfo;
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfo contains data about the I/O that is supported by a segment (slice or module) attached to a SynqNet node.

id	This field contains a 32-bit number that uniquely identifies this kind of segment. For modules attached to a SQID node, the top 16 bits are the manufacturer code and the bottom 16 bits are to product code.
option	The option code for the segment. For slices attached to a Slice network adaptor this field is always zero.
serialNumber	The serial number of this segment. **NOTE**: Slice-I/O nodes do NOT report serial numbers.
modelName	A text string giving the model name of this module.

digitalInCount	The total number of digital inputs on this segment.
digitalOutCount	The total number of digital outputs on this segment.
analogInCount	The total number of analog inputs on this segment.
analogOutCount	The total number of analog outputs on this segment.
version	The version of the segment. For modules attached to a SQID node, this field is always zero.
paramCount	The total number segment parameters supported by this segment. For modules attached to a SQID node this field is always zero.
memoryCount	The total number of memory bytes available on this segment. For modules attached to a SQID node, this field is always zero.
manufacturerData	A series of characters programmed into the node during manufacturing. For slices attached to a Slice network adaptor, this field is always zero.

See Also

MEISqNodeConfigIoAbort

MEISqNodeSegmentUserData

Definition

```
typedef struct MEISqNodeSegmentUserData {
        char   data[MEISqNodeSegmentUserDATA_CHAR_MAX];
} MEISqNodeSegmentUserData;
```

Change History: Modified in the 03.02.00

Description

Modules attached to a SQID node have a small section of non-volatile memory that can be used for any purpose by the user. The **MEISqNodeSegmentUserData** structure holds a copy of this data.

data	Up to 16 bytes of data.

See Also

meiSqNodeSegmentUserDataGet | meiSqNodeSegmentUserDataSet | MPI Overview I/O: User Data

MEISqNodeStatus

Definition

```
typedef struct MEISqNodeStatus {
    MEISqNodeStatusPacketError   upStreamError;
    MEISqNodeStatusPacketError   downStreamError;
    MEISqNodeStatusCrcError      crcError;
    MPIEventMask                 eventMask;
    MPIEventStatusIoFaults       ioFaults;
} MEISqNodeStatus;
```

Change History: Modified in the 03.04.00.

Description

MEISqNodeStatus contains error counters and the ***eventMask*** for a SynqNet node.

upStreamError	The rate and count of bad synqNet messages received by the controller from the Node. See MEISqNodeStatusPacketError.
downStreamError	The rate and count of bad synqNet messages received by the Node from the controller. See MEISqNodeStatusPacketError.
crcError	Counters for the CRC errors. See MEISqNodeStatusCrcError.
eventMask	Array that defines the event mask bits. The array is defined as: `typedef MPIEventMaskELEMENT_TYPE MPIEEventMask [MPIEventMaskELEMENTS]` The bits are defined by the MPI/MEIEventType enumerations.
ioFaults	Flags indicating the source of any I/O faults. See MEISqNodeStatusIoFaults.

See Also

meiSqNodeStatus | meiSynqNetStatus | MEISqNodeConfig | MEISqNodeStatusIoFaults

MEISqNodeStatusCrcError

Definition

```
typedef struct MEISqNodeStatusCrcError {
    long    port[MEINetworkPortLAST];
} MEISqNodeStatusCrcError;
```

Description

MEISqNodeStatusCrcError contains CRC error counters for each network port. The CRC error counters are helpful for diagnosing data integrity problems. The counter increments for any CRC error on any packet received at that port (whether the packet is addressed to the node or not). The CRC error counters are cleared during network initialization.

port	An array of CRC error counters. Each network port has one CRC error counter. The valid range is 0 to 255. The value saturates at 255. A value of -1 indicates the controller was unable to read the data from the node due to a failed connection.

See Also

meiSqNodeStatus | MEINetworkPort

MEISqNodeStatusIoFaults

Definition

```
typedef struct MEISqNodeStatusIoFaults {
   MPI_BOOL  digitalIn;
   MPI_BOOL  digitalOut;
   MPI_BOOL  analogIn;
   MPI_BOOL  analogOut;
} MEISqNodeStatusIoFaults;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeStatusIoFaults contains flags that indicate the source of any I/O fault.

digitalIn	This flag indicates that there is a fault with using the digital inputs.
digitalOut	This flag indicates that there is a fault with using the digital outputs.
analogIn	This flag indicates that there is a fault with using the analog inputs.
analogOut	This flag indicates that there is a fault with using the analog outputs.

See Also

MEISqNodeStatus | meiSqNodeStatus

I/O Faults

MEISqNodeStatusPacketError

Definition

```
typedef struct MEISqNodeStatusPacketError {
    long    rate;
    long    count;
} MEISqNodeStatusPacketError;
```

Description

MEISqNodeStatusPacketError contains packet error counters and rate counters. Each SynqNet node has a packet error counter and a packet error rate counter. Packets addressed to a node are checked for integrity.

The packet error counters are used to monitor long-term data integrity. These counters do not trigger any fault or fail actions. The packet error counter is incremented once for each missing or invalid packet. Typically, an application will periodically read the packet error counters and store the values in a log.

The packet error rate counters are used to trigger fault recovery and/or failure shutdown. The packet error rate counter is incremented for each missing or invalid packet and is decremented for 16 consecutive valid packets. Thus, the packet error rate counters can detect large errors over short periods of time or small errors over long periods of time.

rate	The packet error rate counter. The valid range is 0 to 255. The value saturates at 255. A value of -1 indicates the controller was unable to read the data from the node due to a failed connection.
count	The packet error counter. The valid range is 0 to 255. The value saturates at 255. A value of -1 indicates the controller was unable to read the data from the node due to a failed connection.

See Also

meiSqNodeStatus | MEISqNodeConfigPacketError

MEISqNodeUserData

Definition

```
typedef struct MEISqNodeUserData {
    char    data[MEISqNodeUserDATA_CHAR_MAX];
}MEISqNodeUserData ;
```

Change History: Modified in the 03.02.00

Description

MEISqNodeUserData is used to store the user information that is located on the SqNode.

data	User information on the SqNode used for storing SqNode identification or any other useful data. Programmable string to be used by a customer to store identification-specific information. This data is not used by the MPI and is stored in the SqNode's EEPROM.

See Also

meiSqNodeUserDataGet | meiSqNodeUserDataSet

MEISqNodeConfigControlLatencyMIN_LIMIT

Definition

```
#define  MEISqNodeConfigControlLatencyMIN_LIMIT  (1000)
```

Change History: Added in the 03.04.00.

Description

MEISqNodeConfigControlLatencyMIN_LIMIT defines the minimum specifiable limit for the node's control latency.

See Also

MEISqNodeConfigControlLatencyMAX_LIMIT | MEISqNodeConfigControlLatency | meiSqNodeConfigSet | meiSqNodeConfigGet

MEISqNodeConfigControlLatencyMAX_LIMIT

Definition

```
#define  MEISqNodeConfigControlLatencyMAX_LIMIT  (1000)
```

Change History: Added in the 03.04.00.

Description

MEISqNodeConfigControlLatencyMAX_LIMIT defines

See Also

MEISqNodeConfigControlLatencyMIN_LIMIT

MEISqNodeDriveParamMAX_STRING_LENGTH

Definition

```
#define MEISqNodeDriveParamMAX_STRING_LENGTH  MEIDriveParamMAX_STRING_LENGTH
```

Change History: Modified in the 03.03.00

Description

MEISqNodeDriveParamMAX_STRING_LENGTH defines the maximum

See Also

MEISqNodeID_CHAR_MAX

Definition

```
#define   MEISqNodeID_CHAR_MAX  (30)
```

Description

MEISqNodeID_CHAR_MAX defines the maximum length (number of characters) in a sqNode identification string.

See Also

MEISqNodeFILENAME_MAX

Definition

```
#define  MEISqNodeeFILENAME_MAX  (18)
```

Description

MEISqNodeFILENAME_MAX defines the maximum size allowed for SqNode filenames.

See Also

MEISqNodeManufacturerDATA_CHAR_MAX

Definition

```
#define  MEISqNodeManufacturerDATA_CHAR_MAX  (0x10)
```

Description

MEISqNodeManufacturerDATA_CHAR_MAX defines the maximum number of characters stored in the Manufacturer's Data field.

See Also

MEISqNodeMaxFEEDBACK_SECONDARY

Definition

```
#define  MEISqNodeMaxFEEDBACK_SECONDARY  (MEISqNodeMaxMOTORS)
```

Description

MEISqNodeMaxFEEDBACK_SECONDARY defines the maximum number of secondary feedback devices per SynqNet node.

See Also

MEISqNodeConfig | MEISqNodeFeedbackSecondary

MEISqNodeMaxMOTORS

Definition

```
#define  MEISqNodeMaxMOTORS  (MEIXmpMotorsPerBlock)
```

Description

MEISqNodeMaxMOTORS defines the maximum number of motor objects supported on a single SynqNet node. (MEIXmpMotorsPerBlock = 8) This define should be used instead of the MEIXmpMotorsPerBlock definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

SqNode Objects

MEISqNodeNOT_AVAILABLE

Definition

```
#define  MEISqNodeNOT_AVAILABLE  (-1)
```

Description

MEISqNodeNOT_AVAILABLE defines a possible value for MEISqNodeConfig.feedbackSecondary[n].motorIndex.
If motorIndex = MEISqNodeNOT_AVAILABLE, then a secondary feedback device does not exist on the hardware.

See Also

MEISqNodeConfig | MEISqNodeFeedbackSecondary

MEISqNodeSEGMENT_MAX

Definition

```
#define  MEISqNodeSEGMENT_MAX (32)
```

Change History: Added in the 03.03.00

Description

MEISqNodeSEGMENT_MAX defines the maximum number of Slices or SQID Modules that can be supported.

See Also

Slice I/O | SQID

MEISqNodeSEGMENT_PARAMS_MAX

Definition

```
#define  MEISqNodeSEGMENT_PARAMS_MAX (10)
```

Change History: Added in the 03.03.00

Description

MEISqNodeSEGMENT_PARAMS_MAX defines the maximum number of parameters a Slice can support.

Sample Code

The following code shows how to read all the parameters from a slice, where you may not know how may parameters a slice supports.

```
MEISqNodeSegmentInfo segmentInfo;
meiSqNodeInfo( sqNode, 0, &segmentInfo );

char parameters[MEISqNodeSEGMENT_PARAMS_MAX];
meiSqNodeSegmentParamGet( sqNode0,
                          0,
                          0,
                          segmentInfo.paramCount,
                          parameters );
```

See Also

meiSqNodeSegmentParamSet | meiSqNodeSegmentParamGet

MEISqNodeSEGMENT_MEMORY_MAX

Definition

```
#define  MEISqNodeSEGMENT_MEMORY_MAX (30)
```

Change History: Added in the 03.03.00

Description

MEISqNodeSEGMENT_MEMORY_MAX is a macro that defines the maximum number of memory registers a Slice can support.

Sample Code

The following code shows how to read all the memory registers from a slice, where you may not know how may memory registers a slice supports.

```
MEISqNodeSegmentInfo segmentInfo;
meiSqNodeInfo( sqNode, 0, &segmentInfo );

char memory[MEISqNodeSEGMENT_MEMORY_MAX];
meiSqNodeSegmentMemoryGet( sqNode0,
                           0,
                           0,
                           segmentInfo.memoryCount,
                           memory );
```

See Also

meiSqNodeSegmentMemorySet | meiSqNodeSegmentMemoryGet

MEISqNodeSTATUS_NOT_AVAILABLE

Definition

```
#define  MEISqNodeSTATUS_NOT_AVAILABLE  (-1)
```

Description

With exception to MEISqNodeStatus.eventMaskValue, the **MEISqNodeSTATUS_NOT_AVAILABLE** value is assigned to all Node status variables when the Status is not available as a result of lost communication with the node.

See Also

MEISqNodeStatus

MEISqNodeUserDATA_CHAR_MAX

Definition

```
#define  MEISqNodeUserDATA_CHAR_MAX  (0x10)
```

Description

MEISqNodeUserDATA_CHAR_MAX defines the maximum number of characters in the User defined string, stored on the SqNode.

See Also

MEIFpgaSqNodeVersionMAX | MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEISqNodeSegmentInfoMANUFACTURER_LENGTH

Definition

```
#define MEISqNodeSegmentInfoMANUFACTURER_LENGTH          0x10
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfoMANUFACTURER_LENGTH defines the maximum number of bytes in the manufacturer data.

See Also

Overview of MPI I/O: What Information is Available About Each I/O Segment | meiSqNodeSegmentInfo

MEISqNodeSegmentInfoMODEL_NAME_LENGTH

Definition

```
#define  MEISqNodeSegmentInfoMODEL_NAME_LENGTH   0x20
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfoMODEL_NAME_LENGTH defines the maximum number of characters in the model name description.

See Also

Overview of MPI I/O: What Information is Available About Each I/O Segment | meiSqNodeSegmentInfo

MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH

Definition

```
#define  MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH   0x20
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH defines the maximum number of characters in the serial number.

See Also

Overview of MPI I/O: What Information is Available About Each I/O Segment | meiSqNodeSegmentInfo

MEISqNodeSegmentUserDATA_CHAR_MAX

Definition

```
#define MEISqNodeSegmentUserDATA_CHAR_MAX   (0x10)
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentUserDATA_CHAR_MAX defines the maximum number of user data bytes.

See Also

MEISqNodeSegmentUserData | MPI Overview I/O: User Data

MEIDriveMapParamMAX_STRING_LENGTH

Declaration

```
#define  MEIDriveMapParamMAX_STRING_LENGTH  (256)
```

Required Header: stdmei.h

Description

MEIDriveMapParamMAX_STRING_LENGTH macro defines the maximum length of a string that can be read from, or written to a drive parameter. The value is defined by a drive map constant.

See Also

MEIFPGARINCONREV

Definition

```
#define  MEIFPGARINCONREV  (0x0242)
```

Change History: Modified in the 03.04.00. Modified in the 03.02.00.

Description

MEIFPGARINCONREV defines the version of the SynqNet controller FPGA image that was built and tested with the current version of the MPI.

See Also

MEIFpgaSqMACVersionDEFAULT

Definition

```
#define MEIFpgaSqMACVersionDEFAULT    (0x0300)
```

Change History: Modified in the 03.04.00

Description

MEIFpgaSqMACVersionDEFAULT defines the version of the SqMAC FPGA image that was built and tested with the current version of the MPI. The sqMAC FPGA image is built into the SynqNet node FPGA image. This version applies to all node types.

See Also

MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqMACVersionMIN

Definition

```
#define MEIFpgaSqMACVersionMIN    (0x0300)
```

Change History: Modified in the 03.04.00

Description

MEIFpgaSqMACVersionMIN defines the minimum version of the SqMAC FPGA image that is compatible with the current version of the MPI. The sqMAC FPGA image is built into the SynqNet node FPGA image.

This version applies to all node types.

See Also

MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqMACVersionMAX

Definition

```
#define  MEIFpgaSqMACVersionMAX    (0x03FF)
```

Change History: Modified in the 03.04.00

Description

MEIFpgaSqMACVersionMAX defines the maximum version of the SqMAC FPGA image that is compatible with the current version of the MPI. The sqMAC FPGA image is built into the SynqNet node FPGA image.

This version applies to all node types.

See Also

MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqNodeVersionDEFAULT

Definition

```
#define MEIFpgaSqNodeVersionDEFAULT    (0x0400)
```

Change History: Modified in the 03.04.00. Modified in the 03.03.00. Modified in the 03.02.00.

Description

MEIFpgaSqNodeVersionDEFAULT defines the version of the SynqNet node FPGA image that was built and tested with the current version of the MPI. This is the recommended version to have loaded on all SynqNet nodes.

This version may not apply to all node types.

See Also

MEIFpgaSqNodeVersionMIN | MEIFpgaSqNodeVersionMAX | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqNodeVersionMIN

Definition

```
#define MEIFpgaSqNodeVersionMIN    (0x0400)
```

Change History: Modified in the 03.04.00.

Description

MEIFpgaSqNodeVersionMIN defines the minimum version of the SynqNet node FPGA image that is compatible with the current version of the MPI.

This version may not apply to all node types.

See Also

MEIFpgaSqNodeVersionMAX | MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqNodeVersionMAX

Definition

```
#define MEIFpgaSqNodeVersionMAX    (0x04FF)
```

Change History: Modified in the 03.04.00.

Description

MEIFpgaSqNodeVersionMAX defines the maximum version of the SynqNet node FPGA image that is compatible with the current version of the MPI.

This version may not apply to all node types.

See Also

MEIFpgaSqNodeVersionMIN | MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

SynqNet Objects

Introduction

A **SynqNet** object manages a single SynqNet network connected to a motion controller. It represents the physical network. It contains information about the network state, number of nodes, and netowrk status.

A SynqNet network can have one or more SqNode objects associated with it and each SqNode object can have one or more motors and/or drive interfaces. The SynqNet network provides read/write access to each node. During network initialization, the SynqNet nodes are discovered and mapped to the SynqNet object. The number of motors per SqNode object are determined and mapped to the controller's motor objects.

| Error Messages |

Methods

Create, Delete, Validate Methods

meiSynqNet**Create**
meiSynqNet**Delete**
meiSynqNet**Validate**

Configuration and Information Methods

meiSynqNet**CableNumToNodePort**
meiSynqNet**ConfigGet**
meiSynqNet**ConfigSet**
meiSynqNet**FlashConfigGet**
meiSynqNet**FlashConfigSet**
meiSynqNet**IdleCableListGet**
meiSynqNet**IdleCableStatus**
meiSynqNet**Info**
meiSynqNet**PacketFlashConfigGet**
meiSynqNet**PacketFlashConfigSet**
meiSynqNet**PortToCableNum**
meiSynqNet**PacketConfigGet**
meiSynqNet**PacketConfigSet**
meiSynqNet**Status**
meiSynqNet**Timing**

Action Methods

meiSynqNet**FlashTopologyClear**
meiSynqNet**FlashTopologySave**
meiSynqNet**Init**
meiSynqNet**NetworkObjectNext**
meiSynqNet**NodeRestart**
meiSynqNet**NodeShutdown**
meiSynqNet**Shutdown**

Event Methods

meiSynqNet**EventNotifyGet**
meiSynqNet**EventNotifySet**
meiSynqNet**EventReset**

Relational Methods

meiSynqNet**Control**
meiSynqNet**Number**

Data Types

MEISynqNet**CableList**
MEISynqNet**CableLength**
MEISynqNet**CableStatus**
MEISynqNet**Config**
MEISynqNet**FailedNodeMask**
MEISynqNet**Info**
MEISynqNet**Message**
MEISynqNet**RecoveryMode**
MEISynqNet**ResourceCommand**
MEISynqNet**ResourceIoBits**
MEISynqNet**ResourceMonitor**
MEISynqNet**PacketCfg**
MEISynqNet**PacketCfgEncoder**
MEISynqNet**PacketCfgIo**
MEISynqNet**PacketCfgMotor**
MEISynqNet**PacketCfgNode**
MEISynqNet**PacketCfgProbe**

MEISynqNet**ResourceCfgProbeDepth**

MEISynqNet**ShutdownNodeMask**

MEISynqNet**State**

MEISynqNet**Status**

MEISynqNet**StatusCrcError**

MEISynqNet**Timing**

MEISynqNet**Trace**

Constants

MEISynqNet**MaxCableHOP_COUNT**

MEISynqNet**MaxMotorFEEDBACK_PRIMARY_COUNT**

MEISynqNet**MaxMotorFEEDBACK_SECONDARY_COUNT**

MEISynqNet**MaxMotorCAPTURE_COUNT**

MEISynqNet**MaxMotorCOMPARE_COUNT**

MEISynqNet**MaxMotorENCODER_COUNT**

MEISynqNet**MaxMotorPULSE_ENGINE_COUNT**

MEISynqNet**MaxMotors**

MEISynqNet**MaxNodeMOTORS**

MEISynqNet**MaxNODE_COUNT**

MEISynqNet**NodeMaskELEMENTS**

MEISynqNet**PacketCfgIo**

MEISynqNet**PacketCfgMotor**

MEISynqNet**PacketCfgNode**

meiSynqNetCreate

Declaration

```
MEISynqNet meiSynqNetCreate(MPIControl  control,
                            long        number)
```

Required Header: stdmei.h

Description

meiSynqNetCreate creates a SynqNet object identified by ***number***, which is associated with a ***control*** object. SynqNetCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	An index to the SynqNet network. First network = 0 Second network = 1 Third network = 2, etc.

Return Values	
handle	to a SynqNet object. After creating a SynqNet object it must be validated using meiSynqNetValidate(...).
MPIHandleVOID	if the object could not be created

See Also

meiSynqNetDelete | meiSynqNetValidate

meiSynqNetDelete

Declaration

```
long meiSynqNetDelete(MEISynqNet  synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetDelete deletes a SynqNet object and invalidates its handle. SynqNetDelete is the equivalent of a C++ destructor. All objects that are created must be deleted in the reverse order to avoid memory leaks.

synqNet	a handle to a SynqNet object.

Return Values	
MPIMessageOK	

See Also

meiSynqNetCreate | meiSynqNetValidate

meiSynqNetValidate

Declaration

```
long meiSynqNetValidate(MEISynqNet  synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetValidate validates the SynqNet object and its handle. SynqNetValidate should be called immediately after an object is created.

synqNet	handle to a valid SynqNet object.

Return Values	
MPIMessageOK	
MEISynqNetMessageINTERFACE_NOT_FOUND	

See Also

meiSynqNetCreate | meiSynqNetDelete

meiSynqNetCableNumToNodePort

Declaration

```
long meiSynqNetCableNumToNodePort(MEISynqNet      synqNet,
                                  long            cableNumber,
                                  long            *nodeNumber,
                                  MEINetworkPort  *port);
```

Required Header: stdmei.h

Description

meiSqNodeConfigGet converts a cable number on a SynqNet network into a node number and port. It reads the node number and writes it into a long pointed to by ***nodeNumber*** and reads the port and writes it to the value pointed to by ***port***.

Network connections can be identified by a cable number OR a node number and port. For simple network topologies, it is easier to identify network connections by a cable number. For complex network topologies, it is easier to identify network connections by a node number and port.

synqNet	a handle to the SynqNet object
cableNumber	the number of the cable
***nodeNumber**	a pointer to a node number
***port**	a pointer to an enumerated port value

Return Values	
MPIMessageOK	

See Also

meiSynqNetNodePortToCableNum | MEISynqNetCableList

meiSynqNetConfigGet

Declaration

```
long meiSynqNetConfigGet(MEISynqNet          synqNet,
                         MEISynqNetConfig    *config);
```

Required Header: stdmei.h

Description

meiSynqNetConfigGet reads a SynqNet network (***synqNet***) configuration and writes it into the structure pointed to by ***config***.

synqNet	a handle to a SynqNet object
***config**	a pointer to a SynqNet config structure.

Return Values	
MPIMessageOK	

See Also

meiSynqNetInfo | meiSqNodeConfigGet | meiSynqNetConfigSet

meiSynqNetConfigSet

Declaration

```
long meiSynqNetConfigSet(MEISynqNet          synqNet,
                         MEISynqNetConfig    *config);
```

Required Header: stdmei.h

Description

meiSynqNetConfigSet writes a SynqNet network (***synqNet***) configuration from the structure pointed to by ***config***.

synqNet	a handle to a SynqNet object
***config**	a pointer to a SynqNet config structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessagePARAM_INVALID	
MPISynqNetMessageRING_ONLY	

See Also

meiSynqNetInfo | meiSqNodeConfigSet | meiSynqNetConfigGet

meiSynqNetFlashConfigGet

Declaration

```
long meiSynqNetFlashConfigGet(MEISynqNet          synqNet,
                              void                *flash,
                              MEISynqNetConfig    *config);
```

Required Header: stdmei.h

Description

meiSynqNetFlashConfigGet gets a SynqNet object's flash configuration and writes it in the structure pointed to by ***config***.

synqNet	a handle to a SynqNet object
***config**	pointer to a locally instantiated MEISynqNetConfig structure.
***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.

Return Values	
MPIMessageOK	

See Also

meiSynqNetFlashConfigSet | MEISynqNetConfig | MEIFlash

meiSynqNetFlashConfigSet

Declaration

```
long meiSynqNetFlashConfigSet(MEISynqNet          synqNet,
                              MEIFlash            *flash,
                              MEISynqNetConfig    *config);
```

Required Header: stdmei.h

Description

meiSynqNetFlashConfigSet gets a SynqNet object's flash configuration and writes it in the structure pointed to by ***config***.

NOTE: The network topology must first be saved before changing MEISynqNetConfig.cableLength[n] values in Flash memory. These values will also be cleared when network topology is cleared using meiSynqNetFlashTopologyClear(...).

synqNet	a handle to a SynqNet 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**	pointer to a locally instantiated MEISynqNetConfig structure that has been initialized by performing a call to meiSynqNetFlashConfigGet(...) or meiSynqNetConfigGet(...).

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessagePARAM_INVALID	
MEISynqNetMessageRING_ONLY	
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	

See Also

MEISynqNetConfig | MEIFlash | meiSynqNetFlashConfigGet | meiSynqNetConfigGet | meiSynqNetFlashTopologySave | meiSynqNetFlashTopologyClear

meiSynqNetIdleCableListGet

Declaration

```
long meiSynqNetIdleCableListGet(MEISynqNet          synqNet,
                                MEISynqNetCableList *idleCable);
```

Required Header: stdmei.h

Description

meiSynqNetIdleCableListGet reads a SynqNet network's list of idle cables and writes them into the structure pointed to by ***idleCable***. An idle cable has no data traffic. It is the redundant network connection available in ring topologies only. A single ring topology has only one idle cable.

After network initialization, the cable from the last node to the controller is idle by default. If a network fault occurs and the network is configured for fault recovery, the network traffic will be redirected around the faulty connection, through the idle cable, and the faulty connection will become the new idle cable. To test the idle cable, use the meiSynqNetIdleCableStatus(...) method.

synqNet	a handle to a SynqNet object
***idleCable**	a pointer to a SynqNet cable list structure. The cable list structure contains the number of idle cables and their identifying numbers.

Return Values	
MPIMessageOK	

See Also

meiSynqNetIdleCableStatus | SynqNet Topologies

meiSynqNetIdleCableStatus

Declaration

```
long meiSynqNetIdleCableStatus(MEISynqNet              synqNet,
                               long                    cableNumber,
                               MEISynqNetCableStatus   *cableStatus);
```

Required Header: stdmei.h

Description

meiSynqNetIdleCableStatus reads an idle cable's status and writes it into the structure pointed to by ***cableStatus***. Normally, the idle cable has no data traffic. meiSynqNetIdleCableStatus(...) sends a special test packet across the specified idle cable and then waits for a valid response, then it sends another test packet in the opposite direction and waits for valid response.

The idle cable number for a network can be found using meiSynqNetIdleCableListGet(...). SynqNetIdleCableStatus is not allowed for non-idle cables or when the network is recovering from a fault. During fault recovery (SynqNetState = SYNQ_RECOVERING), the network traffic is redirected around the faulty connection and the idle cable is reassigned. After recovery is complete (SynqNetState = SYNQ), the new idle cable can be tested with SynqNetIdleCableStatus. Use meiSynqNetStatus(...) to determine the SynqNet state.

synqNet	a handle to a SynqNet object
cableNumber	the number of the cable to be tested
***cableStatus**	a pointer to a SynqNet cable status enumerated value.

Return Values	
MPIMessageOK	

See Also

meiSynqNetIdleCableListGet | MEISynqNetState

meiSynqNetInfo

Declaration

```
long meiSynqNetInfo(MEISynqNet       synqNet,
                    MEISynqNetInfo   *info);
```

Required Header: stdmei.h

Description

meiSynqNetInfo reads static information about the network associated with the ***SynqNet*** object and writes it into the structure pointed to by ***info***. The SynqNet info structure contains read only data that was determined during network initialization.

synqNet	a handle to a SynqNet object
***info**	pointer to a SynqNet network information structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

Sample Code

```
/*
   Get the number of nodes on the SynqNet network
   Returns -1 if there is an error
*/
int NumberOfNodes(MEISynqNet synqNet)
{
        MEISynqNetInfo info;
        long returnValue;
        int numNodes = -1;

returnValue = meiSynqNetInfo(synqNet, &info);
if(returnValue == MPIMessageOK)
{
if(info.nodeCount >= 0 && info.nodeCount <= MEIXmpMaxSynqNetBlocks)
{
numNodes = info.nodeCount;
}
}
```

```
return numNodes;
}
```

See Also

meiSynqNetIdleCableListGet | MEISynqNetState

meiSynqNetPacketFlashConfigGet

Declaration

```
long  meiSynqNetPacketFlashConfigGet(MEISynqNet            synqNet,
                                     void                  *flash,
                                     MEISynqNetPacketCfg   *config);
```

Required Header: stdmei.h

Description

meiSynqNetPacketFlashConfigGet is currently unsupported and is reserved for future use.

synqNet	a handle to a SynqNet 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**	pointer to configuation structure defined by MEISynqNetPacketCfg.

Return Values	
MPIMessageUNSUPPORTED	

See Also

meiSynqNetPacketFlashConfigSet | MEISynqNetPacketCfgNode

meiSynqNetPacketFlashConfigSet

Declaration

```
long  meiSynqNetPacketFlashConfigSet(MEISynqNet            synqNet,
                                     void                  *flash,
                                     MEISynqNetPacketCfg   *config);
```

Required Header: stdmei.h

Description

meiSynqNetPacketFlashConfigSet is currently unsupported and is reserved for future use.

synqNet	a handle to a SynqNet 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**	pointer to configuation structure defined by MEISynqNetPacketCfg.

Return Values	
MPIMessageUNSUPPORTED	

See Also

meiSynqNetPacketFlashConfigGet | MEISynqNetPacketCfgNode

meiSynqNetNodePortToCableNum

Declaration

```
long meiSynqNetNodePortToCableNum(MEISynqNet      synqNet,
                                  long            nodeNumber,
                                  MEINetworkPort  port,
                                  long            *cableNumber);
```

Required Header: stdmei.h

Description

meiSynqNetNodePortToCableNum converts a node number and port on a SynqNet network into a cable number. It reads the cable number and writes it into a long pointed to by ***cableNumber***.

Network connections can be identified by a cable number OR a node number and port. For simple network topologies, it is easier to identify network connections by a cable number. For complex network topologies, it is easier to identify network connections by a node number and port.

synqNet	a handle to the SynqNet object
nodeNumber	the number of the node
port	an enumerated port value
***cableNumber**	a pointer to a cable number

Return Values	
MPIMessageOK	

See Also

MEISynqNetCableList

meiSynqNetPacketConfigGet

Declaration

```
/* WARNING: meiSynqNetPacketConfigSet(...) and
 * meiSynqNetPacketFlashConfigSet(...) are  low-level network
 * routines that must clear other controller object
configurations.
 * This method should be run before configuring most MPI objects.
 * Please refer to the online documentation for more information.
 */

long  meiSynqNetPacketConfigGet(MEISynqNet           synqNet,
                                MEISynqNetPacketCfg  *config);
```

Required Header: stdmei.h

Description

meiSynqNetPacketConfigGet reads the current network packet configuration for all nodes found on the network to the location pointed to by ***config***.

This method is useful for viewing the current network packed data being sent across the network. It is used in conjunction with meiSynqNetPacketConfigSet(...). This method is also useful for optimizing network traffic (bandwith).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

synqNet	a handle to a SynqNet object
***config**	pointer to configuation structure defined by MEISynqNetPacketCfg.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MPIMessageUNSUPPORTED	

See Also

meiSynqNetPacketConfigSet | MEISynqNetPacketCfgNode

meiSynqNetPacketConfigSet

Declaration

```
/* WARNING: meiSynqNetPacketConfigSet(...) and
 * meiSynqNetPacketFlashConfigSet(...) are  low-level network
 * routines that must clear other controller object
configurations.
 * This method should be run before configuring most MPI objects.
 * Please refer to the online documentation for more information.
 */

long  meiSynqNetPacketConfigSet(MEISynqNet           synqNet,
                                MEISynqNetPacketCfg  *config);
```

Required Header: stdmei.h

Description

meiSynqNetPacketConfigSet sets the network packet configuration to the configuration defined in the location pointed to by ***config***.

WARNING: meiSynqNetPacketConfigSet(...) is a low-level network routine that will clear other controller object configurations and reset the SynqNet network. This method should be executed in your application before configuring any other MPI objects. However, any control object configurations that force a network re-initialization must be performed before this routine is executed – please see mpiControlConfigSet(...) routine for more information.

This method is useful for optimizing network traffic (bandwith).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

synqNet	a handle to a SynqNet object
***config**	pointer to configuration structure defined by MEISynqNetPacketCfg.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MEISynqNetMessageINCOMPLETE_MOTOR	
MEISynqNetMessageINVALID_AUX_ENC_COUNT	
MEISynqNetMessageINVALID_MOTOR_COUNT	
MEISynqNetMessageINVALID_COMMAND_CFG	
MEISynqNetMessageINVALID_ENCODER_COUNT	
MEISynqNetMessageINVALID_CAPTURE_COUNT	
MEISynqNetMessageINVALID_COMPARE_COUNT	
MEISynqNetMessageINVALID_INPUT_COUNT	
MEISynqNetMessageINVALID_OUTPUT_COUNT	
MEISynqNetMessageINVALID_MONITOR_CFG	
MPIMessageUNSUPPORTED	

See Also

meiSynqNetPacketConfigGet | MEISynqNetPacketCfgNode | MEISynqNetPacketCfg | mpiControlConfigSet

meiSynqNetStatus

Declaration

```
long meiSynqNetStatus(MEISynqNet          synqNet,
                      MEISynqNetStatus    *status);
```

Required Header: stdmei.h

Description

meiSynqNetStatus reads status from the network associated with the SynqNet object and writes it into the structure pointed to by status. The SynqNet status structure contains network operation state and error counters. This data is updated every controller sample.

synqNet	a handle to a SynqNet object
***status**	pointer to a SynqNet status structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSqNodeStatus | meiSynqNetInfo

meiSynqNetTiming

Declaration

```
long meiSynqNetTiming(MEISynqNet         synqNet,
                      MEISynqNetTiming   *timing);
```

Required Header: stdmei.h

Description

meiSynqNetTiming returns the network timing values to the location pointed to by ***timing*** for the current operating network. This method can only be called in MEISynqNetStatus.state >= MEISynqNetStateSYNQ.

synqNet	handle to a valid SynqNet object.
***timing**	a pointer to a MEISynqNetTiming structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEISynqNetTiming | MEISynqNetState

meiSynqNetFlashTopologyClear

Declaration

```
long meiSynqNetFlashTopologyClear(MEISynqNet    synqNet,
                                  MEIFlash      flash);
```

Required Header: stdmei.h

Description

meiSynqNetFlashTopologyClear will clear the topology information that is saved in the controller's flash memory by meiSynqNetTopologySave(...).

Executing this method will disable the controller from comparing the discovered network topology with the topology saved in flash memory. But, during network initialization, the MPI Library still compares the topology stored in the controller's dynamic memory with the discovered topology. In both cases, the default service commands are sent to the nodes if the network initialization transitions to SYNQ (cyclic) mode.

WARNING: This is a low-level network configuration routine that will clear any SqNode or Motor configurations that use service commands.

The **meiSynqNetFlashTopologyClear** method will:

1. Shutdown the SynqNet network.
2. Reset SqNode and Motor configurations to their defaults. See MPI Object Configurations that use Service Commands.
3. Clear network topology information from Flash and Dynamic memory.
4. Initialize the SynqNet network.

synqNet	a handle to an MEISynqNet object whose network topology is to be cleared.
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.

Return Values	
MPIMessageOK	
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	

See Also

Save/Clear Topology to Flash | meiSynqNetFlashTopologySave

meiSynqNetFlashTopologySave

Declaration

```
long meiSynqNetFlashTopologySave(MEISynqNet    synqNet,
                                 MEIFlash      flash);
```

Required Header: stdmei.h

Description

meiSynqNetFlashTopologySave will save the current SynqNet topology to the controller's flash memory.

After a power-on or network shutdown/init, the controller/MPI Library will compare the discovered network topology with the topology saved in flash memory. If the topology information matches, the controller will automatically transition the network to SYNQ (cyclic) mode and configured service commands are sent to the nodes.

WARNING: This is a low-level network configuration routine that will reset the SynqNet network.

The **meiSynqNetFlashTopologySave** method will:

1. Shutdown the SynqNet network.
2. Set SqNode and Motor configurations to the values from Dynamic memory. See MPI Object Configurations that use Service Commands.
3. Initialize the SynqNet network.
4. Save network topology information to Flash memory.

synqNet	a handle to an MEISynqNet object whose network topology is to be saved.
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.

Return Values	
MPIMessageOK	
MPISynqNetMessageTOPOLOGY_SAVED	

See Also

Save/Clear Topology to Flash | meiSynqNetFlashTopologyClear

meiSynqNetInit

Declaration

```
long meiSynqNetInit(MEISynqNet    synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetInit initializes a SynqNet network. This method performs the same network initialization that is automatically done with mpiControlInit(...).

synqNet	a handle to a SynqNet object

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MEISynqNetMessageTOPOLOGY_MISMATCH	
MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH	
MEISynqNetMessageNODE_LATENCY_EXCEEDED	
MEISynqNetMessageNODE_FPGA_VERSION	
MEISynqNetMessageNODE_MAC_VERSION	
MEISynqNetMessageNODE_INIT_FAIL	

See Also

meiSynqNetInfo | meiSynqNetStatus

meiSynqNetNetworkObjectNext

Declaration

```
long meiSynqNetNetworkObjectNext(MEISynqNet           synqNet,
                                 MENetworkPort        port,
                                 MENetworkObjectInfo  *info);
```

Required Header: stdmei.h

Change History: Added in the 03.03.00

Description

meiSynqNetNetworkObjectNext gets the information for the neighboring network object (device) connected to the specified port and writes the information into the structure pointed to by ***info***.

NOTE: This info.type value may be MEINetworkObjectTypeNONE if there is nothing connected to the given port.

synqNet	a handle to a SynqNet object
port	specifies the node's IN or OUT port.
***info**	a pointer to the next object's info structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSqNodeNetworkObjectNext | MEINetworkObjectInfo

Version Utility

meiSynqNetNodeRestart

Declaration

```
long meiSynqNetNodeRestart(MEISynqNet    synqNet);
```

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

Description

meiSynqNetNodeRestart is used to restart all the nodes on the network that are not in cyclic mode and bring them back into Synq mode. The nodes being restarted must be positioned consecutively on the network and have a topology that matches the originally discovered topology. The rest of the network must already be in Synq mode.

synqNet	a pointer to a SynqNet object.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE	
MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING	
MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT	

Sample Code

The following code will restart any nodes that have been shutdown or failed.

```
returnValue = meiSynqNetNodeRestart (synqNet);
msgCHECK(returnValue);
```

See Also

meiSynqNetNodeShutdown | meiSynqNetStatus | MEISynqNetShutdownNodeMask | meiSynqNetInit

SynqNet HotReplace

meiSynqNetNodeShutdown

Declaration

```
long meiSynqNetNodeShutdown(MEISynqNet                    synqNet,
                            MEISynqNetShutdownNodeMask    nodeMask)
```

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

Description

meiSynqNetNodeShutdown is used to systematically shutdown a bank of nodes specified by the nodeMask that are to be repaired or replaced. It should be called prior to meiSynqNetNodeRestart(…), but if the node has already failed shutdown, it is not necessary. The nodes being shutdown should be consecutive to avoid stranding other nodes in the network. Recovery Mode should be enabled before shutting down the nodes in a ring topology.

synqNet	a handle to a SynqNet object.
nodeMask	a bit mask signifying the nodes that will be shutdown. Each bit represents a node (0x1 = node 0, 0x2 = node 1, etc.)

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	
MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE	
MEISynqNetMessageSHUTDOWN_NODES_STRANDED	
MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED	

Sample Code

The following code will attempt to restart nodes 9, 10, and 11.

```
#define   NODE_9    (9)
#define   NODE_10   (10)
#define   NODE_11   (11)

MEISynqNetShutdownNodeMask   nodeMask[0] =
   (1<<NODE_9) | (1<<NODE_10) | (1<<NODE_11) ;

returnValue = meiSynqNetNodeShutdown (synqNet, nodeMask);
msgCHECK(returnValue);

/* swap out and/or repair nodes 9, 10, 11 */

/* Restart nodes 9, 10, 11 */
returnValue = meiSynqNetNodeRestart(synqNet);
msgCHECK(returnValue);

/* check to see which nodes have been restarted */
returnValue = meiSynqNetStatus(synqNet,
                               &status);
msgCHECK(returnValue);
if((nodeMask[0] & (status.failedNodeMask[0] |
                   status.shutdownNodeMask[0])) {

   /* a node did not restart... time for error handling */
}
```

See Also

meiSynqNetShutdown | meiSynqNetNodeRestart | meiSynqNetStatus | MEISynqNetShutdownNodeMask

SynqNet HotReplace

meiSynqNetShutdown

Declaration

```
long  meiSynqNetShutdown(MEISynqNet     synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetShutdown disables a SynqNet network by shutting off cyclic data transmission from the controller to the nodes. This will cause the network state to transition to MEISynqNetStateDISCOVERY and nodes will go into a SynqLost state and disable their outputs.

synqNet	a handle to a SynqNet object

Return Values	
MPIMessageOK	

See Also

meiSynqNetInit | mpiControlReset | mpiControlInit | meiSynqNetStatus

meiSynqNetEventNotifyGet

Declaration

```
long meiSynqNetEventNotifyGet(MEISynqNet     synqNet,
                              MPIEventMask   *eventMask,
                              void           *external);
```

Required Header: stdmei.h

Description

meiSynqNetEventNotifyGet reads the event mask (that specifies the event types 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).

Use the event mask macros mpiEventMaskGET(...), mpiEventMaskBitGET(...), etc. to decode the eventMask.

The event notification data in external is in addition to the event notification data in eventMask. If either eventMask is NULL or external is NULL (not both), then the event notification data will not be copied to the NULL pointer.

Remarks

external either points to a structure of type MEIEventNotifyData or is NULL.

The MEIEventNotifyData structure is an array of controller addresses, whose contents are placed into the MEIEventStatusInfo structure (of all events generated by this object).

synqNet	a handle to a SynqNet object
***eventMask**	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
***external**	pointer to external

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEI/MPIEventType | MEIEventNotifyData | MEIEventStatusInfo

meiSynqNetEventNotifySet

Declaration

```
long meiSynqNetEventNotifySet(MEISynqNet    synqNet,
                              MPIEventMask  eventMask,
                              void          *external);
```

Required Header: stdmei.h

Description

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

Use the event mask macros meiEventMaskSYNQNET(...), mpiEventMaskSET(...), mpiEventMaskBitSET(...), mpiEventMaskCLEAR(...), etc. to create the eventMask.

The event notification data in ***external*** is in addition to the event notification data in ***eventMask***. If either ***eventMask*** is NULL or ***external*** is NULL (not both), then the event notification data will not be copied to the NULL pointer.

Remarks

external either points to a structure of type MEIEventNotifyData or is NULL.

The MEIEventNotifyData structure is an array of controller addresses, whose contents are placed into the MEIEventStatusInfo structure (of all events generated by this object).

synqNet	a handle to a SynqNet object
eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
***external**	pointer to external

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEI/MPIEventType | MEIEventNotifyData | MEIEventStatusInfo

meiSynqNetEventReset

Declaration

```
long meiSynqNetEventReset(MEISynqNet     synqNet,
                          MEIEventMask   eventMask);
```

Required Header: stdmei.h

Description

meiSynqNetEventReset resets the status/event bits that are specified in the eventMask and generated by the synqNet object. After a SynqNet event occurs, SynqNetEventReset should be called to reset the latched status/event bits so the specified event(s) can be generated again.

synqNet	a handle to a SynqNet object
eventMask	An array that defines the event mask bits. The array is defined as: `typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS]` The bits are defined by the MPI/MEIEventType enumerations.

Return Values	
MPIMessageOK	

See Also

meiSynqNetStatus | meiSqNodeEventReset | meiSqNodeStatus | mpiControlEventReset | mpiMotionEventReset | mpiMotorEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSqNodeEventReset | mpiAxisEventReset

Event Notification Methods

meiSynqNetControl

Declaration

```
MPIControl meiSynqNetControl(MEISynqNet  synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetControl returns a handle to the control object associated with the ***SynqNet*** object.

synqNet	a handle to a SynqNet object

Return Values	
MPIControl	a handle to a control object.
MPIHandleVOID	if synqNet is not valid.

See Also

meiSynqNetCreate | mpiControlCreate

meiSynqNetNumber

Declaration

```
long meiSynqNetNumber(MEISynqNet   synqNet,
                      long         *number);
```

Required Header: stdmei.h

Description

meiSynqNetNumber reads the index of a SynqNet network and writes it into the contents of a long pointed to by number. Each SynqNet network associated with a controller is indexed by an identification number (0, 1, 2, etc.).

synqNet	a handle to a SynqNet object.
***number**	a pointer to the index of a SynqNet network.

Return Values	
MPIMessageOK	

See Also

meiSynqNetInfo

MEISynqNetCableList

Definition

```
typedef struct MEISynqNetCableList {
    long    count;
    long    cableNumber[MEISynqNetCableHOP_COUNT];
} MEISynqNetCableList;
```

Description

MEISynqNetCableList contains the number of cables in the SynqNet network and their identifying numbers. The idle cables can be identified with the meiSynqNetIdleCableListGet(...) method.

A cable can also be identified by the node number and port. Use meiSynqNetCableNumToNodePort (...) to translate the cable number to a node number and a port.

count	The number of cables and number of valid elements in the cableNumber[] array.. Range is 0 to MEISynqNetCableHOP_COUNT.
cableNumber	The cables identified by their number. Cables are numbered in the order they are discovered during network initialization.

See Also

meiSynqNetIdleCableListGet | meiSynqNetIdleCableStatus | meiSynqNetCableNumToNodePort

MEISynqNetCableLength

Definition

```
typedef struct MEISynqNetCableLength {  /* lengths in meters */
    long minimum;
    long nominal;
    long maximum;
} MEISynqNetCableLength;
```

Description

MEISynqNetCableLength contains the nominal (average), minimum and maximum length in meters for a given network cable.

To disable the cable length checking feature, set the **minimum**, **nominal**, and **maximum** values to zero.

minimum	Minimum cable length in meters. A value less than zero or greater than the nominal value is invalid.
nominal	Nominal cable length in meters. A value less than zero is invalid.
maximum	Maximum cable length in meters. A value less than the nominal value is invalid.

See Also

MEISynqNetConfig | meiSynqNetConfigSet | meiSynqNetConfigSet | meiSynqNetInfo

MEISynqNetCableStatus

Definition

```
typedef enum MEISynqNetCableStatus {
   MEISynqNetCableStatusGOOD,
   MEISynqNetCableStatusBAD_UPSTREAM,
   MEISynqNetCableStatusBAD_DOWNSTREAM,
   MEISynqNetCableStatusBAD,
} MEISynqNetCableStatus;
```

Description

MEISynqNetCableStatus is an enumeration of a network connection's operating condition. Data transmission is verified in both directions. If the network hardware does not support separate upstream and downstream data path verification, it will report either GOOD or BAD.

MEISynqNetCableStatusGOOD	Network communication across the cable is working properly in both directions.
MEISynqNetCableStatusBAD_UPSTREAM	Network communication across the cable failed in the upstream direction (from node to controller).
MEISynqNetCableStatusBAD_DOWNSTREAM	Network communication across the cable failed in the downstream direction (from the controller to the node).
MEISynqNetCableStatusBAD	Network communication across the cable failed in both directions.

See Also

meiSynqNetIdleCableStatus | meiSynqNetIdleCableListGet

MEISynqNetConfig

Definition

```
typedef struct MEISynqNetConfig {
    MEISynqNetRecoveryMode   recoveryMode;
    MEISynqNetCableLength    cableLength[MEISynqNetCableHOP_COUNT];
} MEISynqNetConfig;
```

Description

MEISynqNetConfig contains configurations for the network's fault recovery mode and the network's cable lengths. The SynqNet configuration can be read with meiSynqNetConfigGet(...) and can be written with meiSynqNetConfigSet(...).

recoveryMode	A enumerated value representing the network's fault recovery response mode.
cableLength	An array of cable lengths. Range for cable lengths is 0 to 100 meters. This structure is provided to allow the user application to specify minimum, maximum, and nominal values to use network scheduling equations. Setting a minimum and maximum value will also enable cable length checking at network initialization time. If measured cable lengths fall outside the range defined by the minimum and maximum values, network initialization will fail with a return value of MEISynqNetMessageCABLE_LENGTH_MISMATCH. By default, the values in this structure are zero. Most applications will not need to modify these defaults. **NOTE**: The network topology must be saved before changing these values in Flash memory. These values will also be cleared when network topology is cleared using meiSynqNetFlashTopologyClear(...).

See Also

meiSynqNetConfigGet | meiSynqNetConfigSet | meiSynqNetInfo | meiSynqNetFlashConfigGet | meiSynqNetFlashConfigSet | meiSynqNetFlashTopologyClear

Cable Length

MEISynqNetFailedNodeMask

Definition

```
#define MEISynqNetNodeMaskELEMENTS  (1)

typedef long MEISynqNetFailedNodeMask  MEISynqNetShutdownNodeMask;
```

Description

MEISynqNetFailedNodeMask is an array of longs with a length of MEISynqNetNodeMaskELEMENTS. Each bit in the failed node mask represents a failed node (0x1 = node 0, 0x2 = node 1, 0x4 = node 2, 0x8 = node 3, etc.). A node failure occurs when the packet error rate counters exceed the packet error rate fail limit.

See Also

MEISynqNetStatus | meiSynqNetStatus | meiSqNodeStatus | MEISqNodeConfig

MEISynqNetInfo

Definition

```
typedef struct MEISynqNetInfo {
    MEINetworkType          networkType;
    long                    nodeCount;
    long                    nodeOffset;
    MEISynqNetCableLength   cableLength[MEISynqNetCableHOP_COUNT];
                            /* measured length */
} MEISynqNetInfo;
```

Description

MEISynqNetInfo contains static data that is determined during network initialization. It identifies the network type, number of nodes on the network, starting number (offset) of the first node, and an rough estimate of measured cable lengths.

networkType	contains currently discovered network topology type.
nodeCount	Contains the number of nodes currently discovered on the network.
nodeOffset	Starting node number for nodes on this network. Nodes are currently numbered sequentially across all networks on a controller.
cableLength	An array of measured network cable lengths. These values are estimated values. At network discovery, where network topology has not been saved to flash, the controller will send network packets to measure each cable length and automatically fill out these values for each cable found. These values are estimated values and are the values used in the network scheduling calculations.

Sample Code

```
/*
   Get the number of nodes on the SynqNet network
   Returns -1 if there is an error
*/
int NumberOfNodes(MEISynqNet synqNet)
{
        MEISynqNetInfo info;
        long returnValue;
        int numNodes = -1;

returnValue = meiSynqNetInfo(synqNet, &info);

  if(returnValue == MPIMessageOK)

  {

  if(info.nodeCount >= 0 && info.nodeCount <= MEIXmpMaxSynqNetBlocks)

  {

  numNodes = info.nodeCount;

  }

  }

 return numNodes;

  }
```

See Also

meiSynqNetInfo | MEISynqNetConfig

MEISynqNetMessage

Definition

```
typedef enum {
    MEISynqNetMessageSYNQNET_INVALID,
    MEISynqNetMessageMAX_NODE_ERROR,
    MEISynqNetMessageSTATE_ERROR,

    MEISynqNetMessageCOMM_ERROR,
    MEISynqNetMessageCOMM_ERROR_CRC,
    MEISynqNetMessageCOMM_ERROR_RX,
    MEISynqNetMessageCOMM_ERROR_RX_LEN,
    MEISynqNetMessageCOMM_ERROR_RX_FIFO,
    MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE,
    MEISynqNetMessageCOMM_ERROR_RX_CRC,

    MEISynqNetMessageINTERFACE_NOT_FOUND,
    MEISynqNetMessageTOPOLOGY_MISMATCH,
    MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH,

    MEISynqNetMessageRESET_REQ_TIMEOUT,
    MEISynqNetMessageRESET_ACK_TIMEOUT,
    MEISynqNetMessageDISCOVERY_TIMEOUT,
    MEISynqNetMessageNO_NODES_FOUND,
    MEISynqNetMessageNO_TIMING_DATA_AVAIL,

    MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW,
    MEISynqNetMessageINVALID_MOTOR_COUNT,
    MEISynqNetMessageINVALID_AUX_ENC_COUNT,
    MEISynqNetMessageINCOMPLETE_MOTOR,
    MEISynqNetMessageINVALID_COMMAND_CFG,
    MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT,
    MEISynqNetMessageINVALID_ENCODER_COUNT,
    MEISynqNetMessageINVALID_CAPTURE_COUNT,
    MEISynqNetMessageINVALID_COMPARE_COUNT,
    MEISynqNetMessageINVALID_INPUT_COUNT,
    MEISynqNetMessageINVALID_OUTPUT_COUNT,
    MEISynqNetMessageINVALID_MONITOR_CFG,
    MEISynqNetMessageINVALID_ANALOG_IN_COUNT,
    MEISynqNetMessageINVALID_DIGITAL_IN_COUNT,
    MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT,
    MEISynqNetMessageINVALID_ANALOG_OUT_COUNT

    MEISynqNetMessageLINK_NOT_IDLE,
    MEISynqNetMessageIDLE_LINK_UNKNOWN,
```

```
    MEISynqNetMessageRING_ONLY,
    MEISynqNetMessageRECOVERING,

    MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED,
    MEISynqNetMessageCABLE_LENGTH_TIMEOUT,
    MEISynqNetMessageCABLE_LENGTH_MISMATCH,
    MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL,
    MEISynqNetMessageCABLE_LENGTH_INVALID_MIN,
    MEISynqNetMessageCABLE_LENGTH_INVALID_MAX,

    MEISynqNetMessageNODE_FPGA_VERSION,
    MEISynqNetMessageMAX_MOTOR_ERROR,
    MEISynqNetMessagePLL_ERROR,
    MEISynqNetMessageNODE_INIT_FAIL,

    MEISynqNetMessageTOPOLOGY_CLEAR,
    MEISynqNetMessageTOPOLOGY_SAVED,
    MEISynqNetMessageTOPOLOGY_AMPS_ENABLED,

    MEISynqNetMessageNODE_MAC_VERSION,
    MEISynqNetMessageADC_SAMPLE_FAILURE,

    MEISynqNetMessageSCHEDULING_ERROR,

    MEISynqNetMessageINVALID_PROBE_CFG,
    MEISynqNetMessageINVALID_PROBE_DEPTH,
    MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE,
    MEISynqNetMessageNODE_LATENCY_EXCEEDED,
    MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE,
    MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING,
    MEISynqNetMessageHOT_RESTART_FAIL_TEST_PACKET,
    MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT,
    MEISynqNetMessageHOT_RESTART_NOT_ALL_NODES_RESTARTED,
    MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE,
    MEISynqNetMessageSHUTDOWN_NODES_STRANDED,
    MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED,

} MEISynqNetMessage;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00. Modified in the 03.02.00.

Description

MEISynqNetMessage is an enumeration of SynqNet error messages that can be returned by the MPI library.

MEISynqNetMessageSYNQNET_INVALID

The SynqNet number is out of range. This message code is returned by meiSynqNetCreate(...) if the SynqNet network number is less than zero or greater than or equal to MEIXmpMaxSynqNets.

MEISynqNetMessageMAX_NODE_ERROR

The SynqNet node number is out of range. This message code is returned by a SynqNet method if the specified node number is greater than or equal to MEIXmpMaxSynqNetBlocks.

MEISynqNetMessageSTATE_ERROR

The SynqNet network state is not valid. This message code is returned by any method that initializes a SynqNet network if the controller's network state is not a member of the MEIXmpSynqNetState or MEIXmpSynqNetInternalState enumeration. The most commonly used methods that initialize the SynqNet network are: mpiControlInit(...), mpiControlReset(...) and meiSynqNetInit(...). This message code indicates a failure in the controller's initialization sequence. To correct this problem, call mpiControlReset(...).

MEISynqNetMessageCOMM_ERROR

The SynqNet network communication failed. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_CRC

The SynqNet network communication failed due to excessive CRC errors. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX

The SynqNet network communication failed due to a Rincon receive error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_LEN

The SynqNet network communication failed due to a Rincon receive length error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_FIFO

The SynqNet network communication failed due to a Rincon receive buffer error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE

The SynqNet network communication failed due to a Rincon receive dribble error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_CRC

The SynqNet network communication failed due to a Rincon receive CRC error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageINTERFACE_NOT_FOUND

The controller does not support a SynqNet interface. This message code is returned by meiSynqNetValidate(...), meiSynqNetFlashTopologySave(...), meiSynqNetFlashTopologyClear(...), mpiControlInit(...), and mpiControlReset(...) if the controller does not have a SynqNet hardware interface. To correct this problem, use a controller that supports SynqNet.

MEISynqNetMessageTOPOLOGY_MISMATCH

The network topology does not match the expected network topology. This message code is returned by mpiControlInit(...) or meiSynqNetInit(...) if the discovered network topology does not match the controller's expected network topology (stored in dynamic memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into it's dynamic memory. This message code indicates the number of nodes, the node order, or types of nodes have changed since the initial network initialization. To correct this problem, either change the network topology to the original configuration or clear the controller's memory with mpiControlReset(...).

MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH

The network topology does not match the expected network topology. This message code is returned by mpiControlInit(...) or meiSynqNetInit(...) if the discovered network topology does not match the controller's expected network topology (stored in flash memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into its dynamic memory. Later, when meiSynqNetFlashTopologySave(...) is called, the topology information is stored into flash memory. This message indicates the number of nodes, the node order, or types of nodes have changed since the topology information was stored in flash. To correct this problem, either change the network topology to the saved configuration or clear the controller's flash topology with meiSynqNetFlashTopologyClear(...).

MEISynqNetMessageRESET_REQ_TIMEOUT

The network reset request packet exceeded the timeout. This message code is returned by mpiControlInit(...), mpiControlReset(...), or meiSynqNetInit(...) if the reset request packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageRESET_ACK_TIMEOUT

The network reset complete packet exceeded the timeout. This message code is returned by mpiControlInit(...), mpiControlInit(...), or meiSynqNetInit(...) if the reset complete packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageDISCOVERY_TIMEOUT

The network topology discovery exceeded the timeout. This message code is returned by mpiControlInit(...), mpiControlInit(...), or meiSynqNetInit(...) if the controller failed to discover the network topology in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageNO_NODES_FOUND

The controller did not find network nodes. This message code is returned by mpiControlInit(...), meiSynqNetPacketConfigSet(...), or meiSynqNetInit(...) if the controller failed to discover any nodes during network initialization. This message code indicates the first node has failed or the network connection from the controller to the first node is faulty. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageNO_TIMING_DATA_AVAIL

The corresponding SynqNet node module does not contain timing data, so the network cannot be initialized. Contact MEI or drive manufacturer for an updated node module.

MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW

The controller's SynqNet buffer size was exceeded. This message code is returned by mpiControlInit (...) or mpiControlReset(...) if the controller's SynqNet buffer could not be allocated due to overflow. This message code indicates the controller does not have enough memory to initialize the network topology. To correct the problem, either reduce the network nodes or use a different controller model.

MEISynqNetMessageINVALID_MOTOR_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more motors than supported by the node.

MEISynqNetMessageINVALID_AUX_ENC_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more secondary encoders than supported by the node.

MEISynqNetMessageINCOMPLETE_MOTOR

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a motor configuration that is missing feedback or command fields.

MEISynqNetMessageINVALID_COMMAND_CFG

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a different command configuration than is supported by the node.

MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains an illegal number of pulse engines.

MEISynqNetMessageINVALID_ENCODER_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more encoders for feedback than are supported by the node.

MEISynqNetMessageINVALID_CAPTURE_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger capture count than is supported by the node.

MEISynqNetMessageINVALID_COMPARE_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger compare count than is supported by the node.

MEISynqNetMessageINVALID_INPUT_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger ioInput count than is supported by the node.

MEISynqNetMessageINVALID_OUTPUT_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger ioOutput count than is supported by the node.

MEISynqNetMessageINVALID_MONITOR_CFG

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more monitor fields than the node can support.

MEISynqNetMessageINVALID_ANALOG_IN_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger analogIn count than is supported by the node.

MEISynqNetMessageINVALID_DIGITAL_IN_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger digitalIn count than is supported by the node.

MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger digitalOut count than is supported by the node.

MEISynqNetMessageINVALID_ANALOG_OUT_COUNT

This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger analogOut count than is supported by the node.

MEISynqNetMessageLINK_NOT_IDLE

This message is returned by meiSynqNetIdleCableStatus(...) when the cable number supplied is not the idle link. The status check can only be run on an idle cable. See meiSynqNetIdleCableListGet(...).

MEISynqNetMessageIDLE_LINK_UNKNOWN

This message is returned by meiSynqNetIdleCableListGet(...) when an idle cable number in a ring topology cannot be determined. This is due to one or more failed nodes on a ring topology. Be sure to check the status of the nodes.

MEISynqNetMessageRING_ONLY

This message is returned by meiSynqNetIdleCableListGet(...) and meiSynqNetIdleCableStatus(...) because idle cables exist on ring topologies only. It is also returned by meiSynqNetConfigSet(...) when attempting to enable SynqNet recovery mode, which is only supported for ring topologies.

MEISynqNetMessageRECOVERING

The network is recovering from a fault condition. This message is returned by meiSynqNetIdleCableStatus(...) during network fault recovery, because the network traffic is being redirected around the faulted cable and a new idle cable is in the process of being assigned. If you receive this message, wait for recovery to complete and then check the identity of the idle cable with meiSynqNetIdleCableListGet(...).

MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED

This message is returned when a cable on the network is too long.

MEISynqNetMessageCABLE_LENGTH_TIMEOUT

The cable length discovery failed to complete due to a timeout. This message is returned by mpiControlInit(…), meiSynqNetInit(…), or meiSynqNetNodeRestart(…) during SynqNet network initialization if the node fails to respond to a cable length measurement packet. This message indicates a hardware problem. To correct the problem, check your cabling and node hardware.

MEISynqNetMessageCABLE_LENGTH_MISMATCH

This message is returned at network initialization when topology has been saved to flash and a cable length is detected that lies outside the MEISynqNetConfig.cableLength[n].minimum and maximum. Check cable length (if necessary) and save new cable length values to flash.

MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL

The nominal cable length is too long.

MEISynqNetMessageCABLE_LENGTH_INVALID_MIN

The minimum cable length is too long or exceeds nominal value.

MEISynqNetMessageCABLE_LENGTH_INVALID_MAX

The maximum cable length is too long or is less than nominal value.

MEISynqNetMessageNODE_FPGA_VERSION

The node resource FPGA image is out of date. This is only a warning message. Your network will still reach cyclic (SYNQ) mode, but certain node resources may not be available or may not operate as expected. Only ignore this warning if you are aware of FPGA changes between this version and the version built with the MPI or use the sqNodeFlash.exe utility to load the most current FPGA version to the node.

MEISynqNetMessageMAX_MOTOR_ERROR

The number of motors on the network exceeds the number set by MEISynqNetMaxMOTORS.

MEISynqNetMessagePLL_ERROR

The node PLL is unable to lock with drive.

MEISynqNetMessageNODE_INIT_FAIL

A node specific initialization routine was unable to complete successfully. Verify node FPGA is the default version for your MPI version. See MEISqNodeInfoFpga.defaultVersion.

MEISynqNetMessageTOPOLOGY_CLEAR

An attempt to clear the saved network topology was made when no network topology has been saved.

MEISynqNetMessageTOPOLOGY_SAVED

An attempt to save network topology was made when the network topology has already been saved. Clear the network topology before attempting to save another topology to flash.

MEISynqNetMessageTOPOLOGY_AMPS_ENABLED

An meiSynqNetFlashTopologySave /Clear routine has been called while one or more motor amplifiers are enabled. Disables all motor amplifiers (mpiMotorAmpEnableSet(...)) before calling these low-level routines. To avoid this error message, disable the motor(s) amp enable before calling meiSynqNetFlashTopologySave/Clear.

MEISynqNetMessageNODE_MAC_VERSION

The node MAC FPGA image is out of date. This is an error message. Your network will probably never reach cyclic (SYNQ) mode. Use the sqNodeFlash.exe utility to load the most current FPGA version tothe node.

MEISynqNetMessageADC_SAMPLE_FAILURE

A check to verify that all analog inputs can be sampled during the given controller sample rate has failed. Either reduce the number of analog inputs on your network (see meiSynqNetPacketConfigGet / meiSynqNetPacketConfigSet) or decrease your controller sample rate (see mpiControlConfigGet / mpiControlConfigSet).

MEISynqNetMessageSCHEDULING_ERROR

This is a generic return value to indicate that a problem has occurred while calculating the network scheduling. For more specific information about the error, run your application with timing assignment tracing turned on, using the trace mask:

0x00100000 SynqNet: Display timing assignments

MEISynqNetMessageINVALID_PROBE_CFG

This message is returned by meiSynqNetPacketConfigSet(...), when the MEISynqNetPacketCfgProbe. count contains a larger count than is supported by the node.

MEISynqNetMessageINVALID_PROBE_DEPTH

This message is returned by meiSynqNetPacketConfigSet(...), when the MEISynqNetPacketCfgProbe. depth contains a larger count than is supported by the node.

MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE

The controller update period is not an integer multiple of all the SynqNet nodes drive periods. This message is returned from mpiControlConfigSet(…) if the specified controller sample rate is not an integer multiple of the node drive periods. For more details, see Sample Rate for more details.

MEISynqNetMessageNODE_LATENCY_EXCEEDED

The node latency exceeded the maximum control latency limit. This message is returned by mpiControlInit(…) or meiSynqNetInit(…) during SynqNet initialization if the node latency exceeds the maximum control latency configured by mpiSynqNetConfigSet(…). To avoid this problem, either leave the maximum control latency configuration at the default value or increase the maximum control latency limit.

MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE

The SynqNet network is not in the SYNQ state. This message is returned by meiSynqNetNodeRestart (…) if the SynqNet network is not in the SYNQ state when the nodes are restarted. To avoid this problem, make sure that the network state is in SYNQ mode before attempting to restart any shutdown nodes. Use meiSynqNetInit(…) to initialize a SynqNet network to the SYNQ state.

MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING

The SynqNet network is not ready for a node restart. This message is returned by meiSynqNetNodeRestart(…) if the SynqNet network fault recovery is in process. To avoid this problem, wait for fault recovery to complete before restarting a node.

MEISynqNetMessageHOT_RESTART_FAIL_TEST_PACKET

The SynqNet node restart did not complete due to a failed restart packet. This message is returned by meiSynqNetNodeRestart(…) if the restart packet failed. The problem is that the test packet is in use, so hot restart cannot take place. Test packets should not be in use when using Hot Restart because restart takes over the test packet's memory locations.

MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT

The SynqNet node restart did not complete due to failed node identification. This message is returned by meiSynqNetNodeRestart(…), if the a node did not respond to a service command to read the node's identification data. This message indicates a hardware problem. To correct the problem, check your cabling and node hardware.

MEISynqNetMessageHOT_RESTART_NOT_ALL_NODES_RESTARTED

One or more SynqNet nodes were not restarted. This message is returned by meiSynqNetNodeRestart(…), if any node could not be restarted. This message is for information purposes. Possible causes for nodes not being restarted are if they are not connected to the network or do not have power.

MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE

The specified nodes to shutdown are not sequential. This message is returned by meiSynqNetShutdown(…) if the shutdown **nodeMask** does not specify sequential nodes. This message protects the user from shutting down nodes without specifying them in the **nodeMask**. For example, suppose there are three nodes in a string (0, 1, 2) and the shutdown nodeMask = 0x5 (node 0 and 2). Node 1 would fail when nodes 0 and 2 are shutdown.

MEISynqNetMessageSHUTDOWN_NODES_STRANDED

The specified nodes to shutdown will cause additional nodes to fail. This message is returned by meiSynqNetShutdown(…) if the shutdown **nodeMask** will cause additional nodes not specified in the **nodeMask** to fail. This message protects the user from shutting down nodes without specifying them in the **nodeMask**. For example, suppose there are two nodes in a string (0, 1) and the shutdown nodeMask = 0x1 (node 0). Node 1 would fail when node 0 is shutdown.

MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED

The SynqNet network recovery mode is disabled. This message is returned by meiSynqNetShutdown(…) if the network is a ring topology and the recovery mode is disabled. To avoid this message, set the SynqNet network recovery configuration to Single-Shot or Auto-Arm.

See Also

MEISynqNetRecoveryMode

Definition

```
typedef enum MEISynqNetRecoveryMode {
    MEISynqNetRecoveryModeDISABLED,
    MEISynqNetRecoveryModeSINGLE_SHOT,
    MEISynqNetRecoveryModeAUTO_ARM,
} MEISynqNetRecoveryMode;
```

Description

MEISynqNetRecoveryMode is an enumeration of a network's fault recovery mode. Networks with ring topologies can be configured to recover from a fault condition. A fault condition occurs when a packet error rate counter exceeds its packet error rate fault limit. If fault recovery is enabled and a fault occurs, the node hardware and/or controller will detect the location of the fault and switch the direction of network traffic for nodes downstream from the faulted connection. During fault recovery, the network state is MEISynqNetStateSYNQ_RECOVERING. After the upstream and downstream packet error rate counters decrement to zero, the recovery is completed, and the network state returns to MEISynqNetStateSYNQ.

When fault recovery occurs, the controller will generate a MEIEventTypeSYNQNET_RECOVERY status/event. An application should notify the user about the fault and fix the broken cable/hardware as soon as possible. An application can determine the location of the fault using meiSynqNetIdleCableListGet(...). A network with a ring topology has one idle cable, which has no data traffic. The operation of the idle cable can be tested with meiSynqNetIdleCableStatus(...).

WARNING: If the idle cable is broken and a second fault occurs, then fault recovery will not be able to fully recover from the fault. SynqNet will try to recover, but some nodes will be stranded. Presently, SynqNet does not support an event to notify the application if an idle cable fails. In the meantime, an application can periodically poll the idle cable with meiSynqNetIdleCableStatus(...) to test the cable.

MEISynqNetRecoveryModeDISABLED	The network will not attempt to recover from a fault condition. This is the default mode for string topologies.
MEISynqNetRecoveryModeSINGLE_SHOT	The network will only attempt to recover from a fault condition one time. A second fault will be ignored.
MEISynqNetRecoveryModeAUTO_ARM	The network will attempt to recover from a fault condition. After the fault recovery is complete, the network will automatically be re-armed to respond to another fault condition. This is the default mode for ring topologies.

See Also

meiSynqNetConfigGet | meiSynqNetConfigSet | meiSynqNetStatus | meiSynqNetInfo | meiSynqNetIdleCableListGet | meiSynqNetIdleCableStatus

MEISynqNetResourceCommand

Definition

```
typedef enum MEISynqNetResourceCommand {
    MEISynqNetResourceCommandINVALID,
    MEISynqNetResourceCommandNONE,
    MEISynqNetResourceCommandDAC_ONE,
    MEISynqNetResourceCommandDAC_TWO,
    MEISynqNetResourceCommandDEMAND_A,
    MEISynqNetResourceCommandDEMAND_AB,
    MEISynqNetResourceCommandDEMAND_ABC,
} MEISynqNetResourceCommand;
```

Change History: Modified in the 03.03.00

Description

MEISynqNetResourceCommand enumeration is a member of the MEISynqNetPacketCfgMotor configuration structure. The values are used to configure the number of 32-bit command data fields sent to drive a motor. Valid values range from greater or equal to MEISynqNetResourceCommandFIRST and less than MEISynqNetResourceCommandLast, but may be further limited by the available resources on the node.

NOTE: A motor is considered incomplete if it has a command value of NONE.

MEISynqNetResourceCommandINVALID	Non-vaild command value was detected.
MEISynqNetResourceCommandNONE	No command data will be sent to this motor. NOTE: this disables the motor and will require the application to remove all other resources for this motor.
MEISynqNetResourceCommandDAC_ONE	Configures a 32bit data field of DAC data to besent to the motor.
MEISynqNetResourceCommandDAC_TWO	Configures two 32bit data fields of DAC data to besent to the motor.
MEISynqNetResourceCommandDEMAND_A	Configures one 16bit data field of Demand data to be sent to the motor.
MEISynqNetResourceCommandDEMAND_AB	Configures two 16bit data fields of Demand data to be sent to the motor.

MEISynqNetResourceCommandDEMAND_ABC	Configures three 16bit data fields of Demand data to be sent to the motor.

See Also

meiSynqNetPacketConfigGet | meiSynqNetPacketConfigSet | MEISynqNetPacketCfgMotor | MEISynqNetPacketCfg | MEISynqNetPacketCfgNode

MEISynqNetResourceIoBits

Definition

```
typedef enum MEISynqNetResourceIoBits {
    MEISynqNetResourceIoBitsINVALID,
    MEISynqNetResourceIoBitsNONE,
    MEISynqNetResourceIoBits16,
    MEISynqNetResourceIoBits48,
} MEISynqNetResourceIoBits;
```

Description

The **MEISynqNetResourceIoBits** enumeration is a member of the MEISynqNetPacketCfgMotor configuration structure. The values are used to configure the number of I/O data fields sent to a motor. Valid values range from greater or equal to MEISynqNetResourceIoBitsFIRST and less than MEISynqNetResourceIoBitsLAST, but may be further limited by the available resources on the node. This is a generic enumeration that is used to configure the resource count of different types of motor and node I/O.

NOTE: (for Motor I/O only) Enabled motors always contain one fixed 16bit data field of dedicated I/O. This is not application configurable.

MEISynqNetResourceIoBitsINVALID	Non-valid bit count was detected.
MEISynqNetResourceIoBitsNONE	No I/O bits will be sent/received.
MEISynqNetResourceIoBits16	One data field of 16 I/O bits will be sent/received.
MEISynqNetResourceIoBits48	One data field of 48 I/O bits will be sent/received.

See Also

meiSynqNetPacketConfigGet | meiSynqNetPacketConfigSet | MEISynqNetPacketCfgMotor | MEISynqNetPacketCfg | MEISynqNetPacketCfgNode

MEISynqNetResourceMonitor

Definition

```
typedef enum MEISynqNetResourceMonitor {
    MEISynqNetResourceMonitorINVALID,
    MEISynqNetResourceMonitorNONE,
    MEISynqNetResourceMonitorAB,
    MEISynqNetResourceMonitorABCD,
} MEISynqNetResourceMonitor;
```

Description

The **MEISynqNetResourceMonitor** enumeration is a member of the MEISynqNetPacketCfgMotor configuration structure. The values are used to configure the number of Monitor data fields received from a motor. Valid values range from greater or equal to MEISynqNetResourceMonitorFIRST and less than MEISynqNetResourceMonitorLAST, but may be further limited by the available resources on the node.

MEISynqNetResourceMonitorINVALID	Non-valid bit count was detected.
MEISynqNetResourceMonitorNONE	No monitor data fields will be sent/received for this motor.
MEISynqNetResourceMonitorAB	Two 16bit fields of monitor data will be sent/received for this motor.
MEISynqNetResourceMonitorABCD	Four 16bit fields of monitor data will be sent/received for this motor.

See Also

MEISynqNetPacketCfg

Definition

```
typedef struct MEISynqNetPacketCfg {
    MEISynqNetPacketCfgNode node[MEISynqNetMaxNODE_COUNT];
} MEISynqNetPacketCfg;
```

Description

MEISynqNetPacketCfg structure is used as a parameter to the meiSynqNetPacketConfigGet/Set methods. It contains the network packet configuration for all nodes found on the network. Only configurable packet data is represented in this structure.

NOTE: Fixed packet fields are NOT application configurable.

node	is an array of configurable packet structures for node data. Array indices 0 through MEISynqNetInfo.nodeCount are valid, with a maximum number of motors per node being defined by MEISynqNetMaxNODE_COUNT.

See Also

meiSynqNetPacketConfigSet | meiSynqNetPacketConfigGet | MEISynqNetPacketCfgNode | MEISynqNetPacketCfgMotor | MEISynqNetPacketCfgEncoder

MEISynqNetPacketCfgEncoder

Definition

```
typedef struct MEISynqNetPacketCfgEncoder {
    long feedbackCount;    /* per encoder */
    long captureCount;     /* per encoder */
    long compareCount;     /* per encoder */
} MEISynqNetPacketCfgEncoder;
```

Description

MEISynqNetPacketCfgEncoder structure is a member of the MEISynqNetPacketCfgMotor configuration structure. It contains the network packet configuration for a single encoder found on a particular motor. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

Setting all structure members to zero (0x0) values will effectively disable the encoder on the motor. Disabling an encoder will NOT affect encoder numbering, however, encoders with higher index numbers must be disabled before using encoders with lower index numbers.

NOTE: A motor is considered incomplete if it has no encoders enabled.

feedbackCount	configures the number of 32-bit feedback data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxEncoderFEEDBACK_COUNT , but may be further limited by the available resources on the node. **NOTE**: Setting this count to a value of zero (0x0) will affectively disable the encoder. All other encoder resources must also be set to NONE or zero (0x0).
captureCount	configures the number of 32-bit capture data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxMotorCAPTURE_COUNT, but may be further limited by the available resources on the node. If feedbackCount is zero (0x0), then this value must also be zero. **NOTE**: all capture/compare resources on a motor rely on 2 fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.

compareCount	configures the number of 32-bit compare data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxMotorCOMPARE_COUNT, but may be further limited by the available resources on the node. If feedbackCount is zero (0x0), then this value must also be zero. **NOTE**: all capture/compare resources on a motor rely on 2 fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.

See Also

meiSynqNetPacketConfigGet | meiSynqNetPacketConfigSet | MEISynqNetPacketCfgMotor | MEISynqNetPacketCfg | MEISynqNetPacketCfgNode

MEISynqNetPacketCfgIo

Definition

```
typedef struct MEISynqNetPacketCfgIo {
    long   digitalInCount;
    long   digitalOutCount;
    long   analogInCount;
    long   analogOutCount;
} MEISynqNetPacketCfgIo;
```

Description

MEISynqNetPacketCfgIo structure configures the network data packets that contain general purpose I/O.

digitalInCount	Selects the number of 32bit words of digital input data to receive over the SynqNet network.
digitalOutCount	Selects the number of 32bit words of digital output data to send over the SynqNet network.
analogInCount	Selects the number of 32bit words of analog input data to receive over the SynqNet network.
analogOutCount	Selects the number of 32bit words of analog output data to send over the SynqNet network.

See Also

meiSynqNetPacketConfigSet | meiSynqNetPacketConfigGet | MEISynqNetPacketCfgNode

MEISynqNetPacketCfgMotor

Definition

```
typedef struct MEISynqNetPacketCfgMotor {
    MEISynqNetResourceCommand   command;
    long                        pulseEngineCount;
    long                        feedbackPrimaryCount;
    long                        captureCount;
    long                        compareCount;
    MEISynqNetResourceIoBits    ioInput;
    MEISynqNetResourceIoBits    ioOutput;
    MEISynqNetResourceMonitor   monitor;
    MEISynqNetPacketCfgProbe    probe;
} MEISynqNetPacketCfgMotor;
```

Description

MEISynqNetPacketCfgMotor structure is a member of the MEISynqNetPacketCfgNode configuration structure. It contains the network packet configuration for a single motor found on a particular node. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

NOTE: Setting all structure members to a NONE or Zero (0x0) value will effectively disable the motor on the network. This will cause the controller to renumber the subsequent motors on the network.

command	Selects the command data type and count. Please see MEISynqNetResourceCommand.
pulseEngineCount	Configures the number of pulse engine to be enable for a motor. Valid numbers are zero thru MEISynqNetMaxMotorPULSE_ENGINE_COUNT, but may be further limited by the available resources on the node. Each pulse engine requires a 64-bit upstream data field and a 32-bit downstream data field.
feedbackPrimaryCount	Configures the number of 32-bit feedback data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxEncoderFEEDBACK_COUNT, but may be further limited by the available resources on the node.

captureCount	Configures the number of 32-bit capture data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxMotorCAPTURE_COUNT, but may be further limited by the available resources on the node. If feedbackCount is zero (0x0), then this value must also be zero. **NOTE**: All capture/compare resources on a motor rely on two fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.
compareCount	Configures the number of 32-bit compare data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxMotorCOMPARE_COUNT, but may be further limited by the available resources on the node. **NOTE**: all capture/compare resources on a motor rely on two fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.
ioInput	Selects the number of input bit data received from this motor. Please see MEISynqNetResourceIoBits.
ioOutput	Selects the number of output bit data sent to this motor. Please see MEISynqNetResourceIoBits.
monitor	Selects the number of montior data fields received from this motor. Please see MEISynqNetResourceMonitor.
probe	A structure that specifies the probe count and register depth.

See Also

meiSynqNetPacketConfigGet | meiSynqNetPacketConfigSet | MEISynqNetPacketCfg | MEISynqNetPacketCfgNode | MEISynqNetResourceCommand | MEISynqNetResourceIoBits | MEISynqNetResourceMonitor | MEISynqNetPacketCfgProbe

MEISynqNetPacketCfgNode

Definition

```
typedef struct MEISynqNetPacketCfgNode {
     MEISynqNetPacketCfgMotor   motor[MEISynqNetMaxNodeMOTORS]
     MEISynqNetPacketCfgIo      io;
     long                       feedbackSecondaryCount;
} MEISynqNetPacketCfgNode;
```

Description

MEISynqNetPacketCfgNode is a member of the MEISynqNetPacketCfg configuration structure. It contains the network packet configuration for all motors found on a particular node. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

motor	An array of configurable packet structures for motor data. Array indices 0 through MEISqNodeInfo.motorCount are valid, with a maximum number of motors per node being defined by MEISynqNetMaxNodeMOTORS.
io	Configures the network data packets general purpose node I/O.
feedbackSecondaryCount	Configures the number of 32-bit secondary feedback data fields to be sent for a node. Valid numbers are zero thru MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT, but may be further limited by the available resources on the node.

See Also

meiSynqNetPacketConfigGet | meiSynqNetPacketConfigSet | MEISynqNetPacketCfg

MEISynqNetPacketCfgProbe

Definition

```
typedef struct MEISynqNetPacketCfgProbe {
     long                                count;
     MEISynqNetResourceProbeDepth        depth[MEISynqNetMaxMotorPROBE_COUNT];
} MEISynqNetPacketCfgProbe;
```

Description

MEISynqNetPacketCfgProbe specifies the network packet configuration for all the Probes found on a particular motor. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

count	the number of Probe engines per motor. Each Probe engine requires cyclic packet data for the status fields.
depth	an array of enumerated values representing the Probe data depth. Each probe engine can support up to 16 register fields for the probe data. The length of the array is specified by the Probe count.

See Also

MEISynqNetPacketCfgMotor | MEISynqNetResourceProbeDepth | MPIProbeStatus

MEISynqNetResourceCfgProbeDepth

Definition

```
typedef enum MEISynqNetResourceProbeDepth {
    MEISynqNetResourceProbeDepthNONE,
    MEISynqNetResourceProbeDepthTWO,
    MEISynqNetResourceProbeDepthFOUR,
    MEISynqNetResourceProbeDepthEIGHT,
    MEISynqNetResourceProbeDepthSIXTEEN,
} MEISynqNetResourceProbeDepth;
```

Description

MEISynqNetResourceCfgProbeDepth is an enumeration of the possible Probe register depths. Each Probe register is 16 bits. The packet data size is 32 bits. Each Probe engine can support up to 16 register fields for the Probe data.

MEISynqNetResourceProbeDepthNONE	zero Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthTWO	2 Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthFOUR	4 Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthEIGHT	8 Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthSIXTEEN	16 Probe data registers will be transmitted upstream

See Also

MEISynqNetShutdownNodeMask

Definition

```
typedef MEISynqNetFailedNodeMask  MEISynqNetShutdownNodeMask;
```

Change History: Added in the 03.04.00.

Description

MEISynqNetShutdownNodeMask is an array of longs with length of MEISynqNetNodeMaskELEMENTS. Each bit the shutdown node mask represents a node (0x1 = node 0, 0x2 = node 1, 0x4 = node 2, 0x8 = node 3, etc.) to be shutdown.

See Also

MEISynqNetStatus | meiSynqNetStatus | meiSynqNetNodeShutdown

MEISynqNetState

Definition

```
typedef enum MEISynqNetState {
    MEISynqNetStateINVALID,
    MEISynqNetStateDISCOVERY,
    MEISynqNetStateASYNQ,
    MEISynqNetStateSYNQ,
    MEISynqNetStateSYNQ_RECOVERING,
} MEISynqNetState;
```

Description

MEISynqNetState is an enumeration of the SynqNet network states. Each state shows the present operation mode for the network. Network data traffic occurs in two modes: asynchronous and synchronous.

During **asynchronous** communication, the MPI or controller sends one packet to one node and the node responds with one packet. There are no timing restrictions. Asynchronous communication is used during network initialization, reset, discovery, and node binary download.

During **synchronous** communication, the controller sends packets to the nodes and the nodes respond with packets. All data traffic is scheduled in fixed timeslots to avoid collisions. Packet data updates are cyclic and synchronized to the controller's sample rate. The network state can be read with meiSynqNetStatus(...).

MEISynqNetStateDISCOVERY	This is the initial state before the network is initialized. During this phase, the controller checks the network integrity, determines the network topology, resets the nodes, identifies, initializes, and addresses the nodes. Data packets are sent/received asynchronously.
MEISynqNetStateASYNQ	This state is used for off-line operations, like node binary download. Data packets are sent/received asynchronously.
MEISynqNetStateSYNQ	This is the normal operating state. Data packets are sent/received synchronously.
MEISynqNetStateSYNQ_RECOVERING	During this state, a network fault condition was detected and the network is being reconfigured to route data packets around the fault. After the reconfiguration is complete and the packet error rate counters have decremented to zero, the network will return to the SYNQ state.

See Also

MEISynqNetStatus | meiSynqNetStatus

MEISynqNetStatus

Definition

```
typedef struct MEISynqNetStatus {
    MEISynqNetState             state;
    MPI_BOOL                    topologySaved;  /* TRUE/FALSE */
    MEISynqNetStatusCrcError    crcError;
    MPIEventMask                eventMask;
    MEISynqNetFailedNodeMask    failedNodeMask;
    MEISynqNetShutdownNodeMask  shutdownNodeMask;
} MEISynqNetStatus;
```

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

Description

MEISynqNetStatus contains network state information, CRC counters, eventMask, and the failedNodeMask for a SynqNet network. The network status can be read with the meiSynqNetStatus (...) method.

state	Present operation mode for the network.
topologySaved	Contains the status of the network topology. FALSE means the topology is dynamically discovered at initialization. TRUE means that the topology is verified against an expected topology at initialization. See Saving Current Topology To Flash for more information.
crcError	CRC error counters for the controller's network ports. The CRC value increments when received data is corrupt. Range is from 0 to 255. Counter saturates at 255.

eventMask	An array that defines the status for the event mask bits. The array is defined as: `typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS]` The bits are defined by the MPI/MEIEventType enumerations.

failedNodeMask	Array that defines the failed node mask bits. Each bit represents a failed node (0x1 = node 0, 0x2 = node 2, 0x4 = node 3, etc.). The array is defined as:
	
	`#define MEISynqNetNodeMaskELEMENTS (1)`
	`typedef long MEISynqNetFailedNodeMask [MEISynqNetNodeMaskELEMENTS];`

shutdownNodeMask	Bit mask representing the nodes that have been shutdown by the user.

See Also

meiSynqNetStatus | meiSqNodeStatus | MEISqNodeStatus | meiSynqNetNodeShutdown | meiSynqNetNodeShutdownNodeMask

MEISynqNetStatusCrcError

Definition

```
typedef struct MEISynqNetStatusCrcError {
    long port[MEINetworkPortLAST];
} MEISynqNetStatusCrcError;
```

Description

MEISynqNetStatusCrcError contains the CRC values for each network port on the controller. This structure is read via the meiSynqNetStatus(...) method.

port	an array of CRC values, one for each network port.

See Also

MEINetworkPort | MEISynqNetStatus | meiSynqNetStatus | CRC Error Counters

MEISynqNetTiming

Definition

```
typedef struct MEISynqNetTiming {
    double controllerFreq;      /* kHz */
    double controllerPeriod;    /* uS  */

    long   txTime;              /*  %  */
    double calculationLimit;    /* uS  */
    double calculationTime;     /* uS  */
    double calculationSlack;    /* uS  */

    long   bandwidthUsage;      /*  %  */

    struct {
        double synq;            /* uS  - synq packet + spacing         */
        double demand;          /* uS  - sum demand packets + spacing  */
        double control;         /* uS  - sum control packets + spacing */

        double total;           /* uS  */
    } downstream;

    struct {
        double feedback;        /* uS  - sum feedback packets + spacing */
        double status;          /* uS  - sum status packets + spacing   */

        double total;           /* uS  */
    } upstream;

    struct {
        double updateFreq;      /* uS  */
        double updatePeriod;    /* uS  */

        double demandLatency;   /* uS  */
        double feedbackLatency; /* uS  */
        double latencyOverhead; /* uS  */
        double controlLatency;  /* uS  */

        double discoveryLatencyTolerance; /* +/- uS per network discovery */
    } node[MEISynqNetMaxNODE_COUNT];
} MEISynqNetTiming;
```

Change History: Modified in the 03.04.00.

Description

MEISynqNetTiming contains static data that is determined during network initialization. It identifies timing statistics about the current network. Most values are just reported based on network timing calculations. The values that can be configured by the MPI are stated below as configurable.

controllerFreq	The sample rate calculation of the SynqNet controller board, in kHz. It is also the rate at which SynqNet packets are sent. The controllerFreq must be configured so that the driveUpdateFreq is an integer multiple of the controllerFreq. This value is configurable. See MPIControlConfig.
controllerPeriod	The sample period calculation of the SynqNet controller board, in uS. Derived from controllerFreq.
txTime	The scheduled time for SynqNet packets to be sent, in % (of the controllerPeriod). Defaults to 75%. Must be set to a value greater than the foreground calculation time, and less than 100%. This value is configurable. See MEIControlConfig.
calculationLimit	The maximum allowed foreground calculation time, in uS. Derived from txTime * controllerPeriod.
calculationTime	The foreground calculation time of the controller, in uS.
calculationSlack	The available slack time between the foreground calculation time, and the scheduled txTime, in uS.
bandwidthUsage	The amount of SynqNet bandwidth used, in %, for this SynqNet configuration. The actual packet payload configured is divided by the maximum available SynqNet bandwidth, for both upstream and downstream packets. The greater of the two is reported.
downstream	Total downstream (controller to nodes) packet payload, in uS. Includes spacing between packets. Breaks down into the following three packet types.
downstream.synq	Downstream SYNQ packet payload, in uS. Includes spacing between packets.
downstream.demand	Downstream DEMAND packet payload, in uS. Includes spacing between packets.
downstream.control	Downstream CONTROL packet payload, in uS. Includes spacing between packets.

upstream	Total upstream (nodes to controller) packet payload, in uS. Includes spacing between packets. Breaks down into the following two packet types.
upstream.feedback	Upstream FEEDBACK packet payload, in uS. Includes spacing between packets.
upstream.status	Upstream STATUS packet payload, in uS. Includes spacing between packets.
node.updateFreq	The cyclic communication rate of a drive processor on a SynqNet node, in kHz. Typically fixed to 16 kHz, but may vary depending on drive type. May be configurable on some drives. This rate often matches the drive PWM rate. Analog drives that do not have drive processors and thus have no scheduled updates, do not have update frequencies and will report 0.
node.updatePeriod	The cyclic communication period of a drive processor on a SynqNet node, in uS. Derived from driveUpdateFreq. Analog drives that do not have drive processors and thus have no scheduled updates, do not have update frequencies and will report 0.
node.demandLatency	The time to send SynqNet demand data downstream from controller to nodes, in uS. Does not include drive demand delays.
node.feedbackLatency	The time to send SynqNet feedback data upstream from nodes to controller, in uS. Rounded up to make the total SynqNet latency an integer multiple of the driveUpdatePeriod. Does not include drive feedback delays.
node.latencyOverhead	
node.controlLatency	The overall SynqNet system control latency, in uS. Always rounded up to an integer multiple of driveUpdatePeriod. Control latency begins when position feedback is sampled on the node, includes upstream packet delays, controller foreground calculation, downstream packet delays, and ends when demands are strobed on the nodes. Note drive feedback and demand delays are NOT included. Control latency can be broken down into the four components listed below: feedbackLatency, calculationTime, calculationSlack, and demandLatency.
node.discoveryLatencyTolerance	

See Also

SynqNet Timing Values

MEISynqNetTrace

Definition

```
typedef enum {
    MEISynqNetTraceDYNA_ALLOC   = MEISynqNetTraceFIRST << 0,
    MEISynqNetTraceDYNA_FREE    = MEISynqNetTraceFIRST << 1,
    MEISynqNetTraceINIT         = MEISynqNetTraceFIRST << 2,
    MEISynqNetTraceRESET        = MEISynqNetTraceFIRST << 3,
    MEISynqNetTraceTIMING       = MEISynqNetTraceFIRST << 4,
    MEISynqNetTraceSERVICE_CMD  = MEISynqNetTraceFIRST << 5,
    MEISynqNetTraceFLOWCTRL     = MEISynqNetTraceFIRST << 6,
    MEISynqNetTraceMAP          = MEISynqNetTraceFIRST << 7,
    MEISynqNetTraceUNMAP        = MEISynqNetTraceFIRST << 8,

} MEISynqNetTrace;
```

Description

MEISynqNetTrace is an enumeration of SynqNet trace bits that can be used to enable/disable library trace statement output for the SynqNet object.

MEISynqNetTraceDYNA_ALLOC	Enables trace statements for controller external memory allocation during SynqNet initialization.
MEISynqNetTraceDYNA_FREE	Enables trace statements for controller external memory de-allocation.
MEISynqNetTraceINIT	Enables trace statements for SynqNet network initialization.
MEISynqNetTraceRESET	Enables trace statements for SynqNet network reset.
MEISynqNetTraceTIMING	Enables trace statements for SynqNet network timing calculations.
MEISynqNetTraceSERVICE_CMD	Enables trace statements for SynqNet service command transactions between the controller and SynqNet nodes.
MEISynqNetTraceFLOWCTRL	This trace bit enables trace statements for the SynqNet service command handshake between the controller and SynqNet nodes.
MEISynqNetTraceMAP	This trace bit enables trace statements for the mapping of firmware pointers to the dynamic controller memory.

MEISynqNetTraceUNMAP	This trace bit enables trace statements for the unmapping of firmware pointers from the dynamic controller memory.

See Also

Trace.exe utility

MEISynqNetMaxCableHOP_COUNT

Definition

```
#define MEISynqNetCableHOP_COUNT (MEISynqNetMaxNODE_COUNT + 1)
```

Description

MEISynqNetMaxCableHOP_COUNT is the maximum number of cables for a SynqNet network. Presently, string and ring topologies are supported. The maximum number of cables is based on the maximum number of nodes, plus one return cable for ring topologies.

See Also

MEISynqNetCableList | MEISynqNetConfig

MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT

Definition

```
#define MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT  (1)
```

Description

MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT defines the maximum number of primary feedback resources per motor.

NOTE: Feedback count may be further limited by the available resources on the node.

See Also

MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT

Definition

```
#define MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT
   (MEISqNodeMaxFEEDBACK_SECONDARY)
```

Description

MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT defines the maximum number of secondary feedback resources per motor.

NOTE: Feedback count may be further limited by the available resources on the node.

See Also

MEISynqNetMaxMotorCAPTURE_COUNT

Definition

```
#define MEISynqNetMaxMotorCAPTURE_COUNT (MEIXmpMaxCapturesPerMotor) /*  2 */
```

Description

MEISynqNetMaxMotorCAPTURE_COUNT defines the maximum number of capture resources per motor.

NOTE: Capture count may be further limited by the available resources on the node.

This define should be used instead of the MEIXmpMaxCapturesPerMotor definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

MEISynqNetMaxMotorCOMPARE_COUNT

Definition

```
#define MEISynqNetMaxMotorCOMPARE_COUNT (MEIXmpMaxCapturesPerMotor) /*  2 */
```

Description

MEISynqNetMaxMotorCOMPARE_COUNT defines the maximum number of compare resources per motor.

NOTE: Compare count may be further limited by the available resources on the node.

This define should be used instead of the MEIXmpMaxComparesPerMotor definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

MEISynqNetMaxMotorENCODER_COUNT

Definition

```
#define MEISynqNetMaxMotorENCODER_COUNT (MEIXmpMotorEncoders) /*  2 */
```

Description

MEISynqNetMaxMotorENCODER_COUNT defines the maximum number of encoder resources per motor.

NOTE: The encoder count may be further limited by the available resources on the node.

This define should be used instead of the MEIXmpMotorEncoders definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

MEISynqNetMaxMotorPULSE_ENGINE_COUNT

Definition

```
#define MEISynqNetMaxMotorPULSE_ENGINE_COUNT  (1)
```

Description

MEISynqNetMaxMotorPULSE_ENGINE_COUNT defines the number of pulse engines per motor.

See Also

MEISynqNetMaxMOTORS

Definition

```
#define MEISynqNetMaxMOTORS (MEIXmpMAX_Motors) /* 32 */
```

Description

MEISynqNetMaxMOTORS defines the maximum number of motors supported on a single SynqNet network.

See Also

MEISynqNetMaxNodeMOTORS

Definition

```
#define MEISynqNetMaxNodeMOTORS (MEISqNodeMaxMOTORS)
```

Description

MEISynqNetMaxNodeMOTORS defines the maximum number of motor objects supported on a single SynqNet node.

See Also

MEISynqNetMaxNODE_COUNT

Definition

```
#define MEISynqNetMaxNODE_COUNT (MEIXmpMaxSynqNetBlocks) /* 32 */
```

Description

MEISynqNetMaxNODE_COUNT defines the maximum number of nodes supported on a single SynqNet network. This define should be used instead of the MEIXmpMaxSynqNetBlocks definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

MEISynqNetNodeMaskELEMENTS

Definition

```
#define MEISynqNetNodeMaskELEMENTS (1)
typedef long MEISynqNetFailedNodeMask  [MEISynqNetNodeMaskELEMENTS];
```

Description

MEISynqNetNodeMaskELEMENTS defines the number of data elements in an MEISynqNetFailedNodeMask.

See Also

MEISynqNetFailedNodeMask

Trace Objects

Introduction

Use the **Trace** module to selectively produce trace output on a global and/or per-object basis for your application. You can specify the types of trace output when an application is linked, or dynamically (by using a debugger).

NOTE: You can also define your own trace function, using meiPlatformTraceFunction(...) For example, you could define your own function to send traces to a circular memory buffer.

The format of the trace output is determined by **printf(...)**-like trace macros located in MPI library source. The trace macros are of the form **meiTrace#(mask, format, arg ...)**, where ***format*** and the ***args*** determine the trace output, and where # indicates the total number of arguments following the ***format*** argument (because macros cannot take variable numbers of arguments).

The placement and content of the **meiTrace(...)** macros in the MPI library source is the responsibility of whomever maintains the library. Because trace can be added as desired, it is often useful to leave trace statements in the library source code rather than remove them, as is similarly done with debug printf(...) statements. It is also useful to define per-object trace output types so that the volume of trace output is set to a manageable level.

The Trace module interface is declared in the **XMP\include\trace.h** header file. In order for your application to use Trace functions, you must build your application with the MEI_TRACE conditional-compile symbol defined.

NOTE: **Debug and DebugSingle are the only MPI library configurations that will produce trace output.**

To install trace, simply install the DLL for either the Debug or DebugSingle configuration. The Debug and DebugSingle configurations of the MPI library are built with the MEI_TRACE compile-time symbol defined.

By default, trace output is sent to standard error. However, to send trace output to a file, your application can call the meiTraceFile(char *fileName) function.

To obtain the current global trace mask, call meiTraceGet.
To modify the global trace mask, call meiTraceSet.

To obtain an object's trace mask, call meiObjectTraceGet (defined in stdmei.h).
To modify an object's trace mask, call meiObjectTraceSet.

See Also:
Trace Masks
Global Trace Outputs
Per-Object Trace Outputs

Methods

Configuration and Information Methods

meiTrace**Eol**	Set the end-of-line character to be used by Trace
meiTrace**File**	Send trace output to a file
meiTrace**Function**	sets function used to display a trace buffer
meiTrace**Get**	Get global trace mask
meiTrace**MaskBits**	Convert the trace mask into an array of trace bits.
meiTrace**Msg**	Convert the message identification value into a string.
meiTrace**MsgFunction**	Set a module's trace message function.
meiTrace**Set**	Set global trace mask

Data Types

MEITrace
MEITrace**Function**
MEITrace**Mask**

Constants

MEITrace**MaskGLOBAL**

meiTraceFile

Declaration

```
long meiTraceFile(const char *fileName)
```

Required Header: stdmei.h

Description

meiTraceFile causes trace output to be sent to the file ***fileName***. By default, trace output goes to standard output. Note that if ***fileName*** is Null, trace output still goes to standard output.

WARNING!
Be careful, you can easily run out of disk space. To save disk space, use a circular file type instead of regular file type.

Return Values	
MPIMessageOK	if *TraceFile* successfully causes trace output to be sent to the file

See Also

meiTraceGet

Declaration

```
MEITraceMask meiTraceGet(void)
```

Required Header: stdmei.h

Description

meiTraceGet returns the current global trace mask for the application.

Returns
The global trace mask

See Also

meiTraceSet

meiTraceSet

Declaration

```
MEITraceMask meiTraceSet(MEITraceMask mask)
```

Required Header: stdmei.h

Description

meiTraceSet sets the global trace mask to ***mask***.

If "traceMask" is	Then
MEITraceALL	all global categories of trace will be enabled
MEITraceNONE	all categories of trace will be disabled

Returns
The value of the previous *global trace mask*

See Also

meiTraceGet | MEITrace

meiTraceEol

Declaration

```
char meiTraceEol(char eol)
```

Required Header: stdmei.h

Description

meiTraceEol function simply calls **meiPlatformTraceEol(...)**, which sets the end-of-line character that will be used by meiPlatformTrace(...). By default, meiPlatformTrace(...) will append a newline character ('\n') to the messages that it displays. The **meiPlatformTraceEol(...)** function allows your application to set the default end-of-line character.

Returns
The previous end-of-line character used by meiPlatformTrace(...)

See Also

meiPlatformTrace | meiPlatformTraceEol

meiTraceFunction

Declaration

```
MEITraceFunction meiTraceFunction(MEITraceFunction traceFunction)
```

Required Header: stdmei.h

Description

meiTraceFunction sets the function used to display a trace buffer.

Front end to meiPlatformTraceFunction(). If traceFunction is NULL (default), then trace functions is fprintf(MEIPlatformTraceSTREAM) (default stdout).

Return Values	
handle	to previous Trace function
NULL	otherwise

See Also

meiTraceMaskBits

Declaration

```
long meiTraceMaskBits(long    mask,
                      long    *bitCount,
                      long    *bit);
```

Required Header: stdmei.h

Description

meiTraceMaskBits converts the trace mask into an array of trace bits and the length of the array.

mask	A bit mask of enumerated trace bits.
***bitCount**	A pointer to a long containing the number of trace bits enabled in the mask. This value is also the length of the bit array.
***bit**	A pointer to an array of longs containing the enumerated trace bits. Each array member contains one trace bit enumerated value.

See Also

MEITrace | meiTraceGet | meiTraceSet

meiTraceMsg

Declaration

```
const char *meiTraceMsg(long    messageId,
                        char    *messageText);
```

Required Header: stdmei.h

Description

meiTraceMsg converts the message identification value into a string pointed to by ***messageText***.

messageId	a message identification value.
***messageText**	a pointer to a character string containing the text for the messageId.

See Also

meiTraceMsgFunction

meiTraceMsgFunction

Declaration

```
long meiTraceMsgFunction(MPIModuleId            moduleId,
                         MEITraceMsgFunction    function);
```

Required Header: stdmei.h

Description

meiTraceMsgFunction sets a module's trace message function.

moduleId	an enumerated module identification value
function	a pointer to a trace message function.

See Also

MEITrace

Definition

```
typedef enum {
    MEITraceNONE = 0,
    MEITraceFIRST = 0x0001,
    MEITraceFUNCTION_ENTRY   = (int) MEITraceFIRST << 0,
    MEITraceFUNCTION_RETURN  = (int) MEITraceFIRST << 1,
    MEITraceMEMORY_ALLOC     = (int) MEITraceFIRST << 2,
    MEITraceMEMORY_FREE      = (int) MEITraceFIRST << 3,
    MEITraceMEMORY_GET       = (int) MEITraceFIRST << 4,
    MEITraceMEMORY_SET       = (int) MEITraceFIRST << 5,
    MEITraceVALIDATE         = (int) MEITraceFIRST << 6,
    MEITraceLOCK_GIVE        = (int) MEITraceFIRST << 7,
    MEITraceLOCK_TAKE        = (int) MEITraceFIRST << 8,
    MEITraceEVENT            = (int) MEITraceFIRST << 9,

    MEITraceALL = (int) ((MEITraceLAST << 1) - 1)
} MEITrace;
```

Description

MEITrace is an enumeration of generic trace bits that can be used to enable/disable library trace statement output for objects throughout the MPI.

MEITraceFUNCTION_ENTRY	Trace the entry into all methods.
MEITraceFUNCTION_RETURN	Trace the return from all methods.
MEITraceMEMORY_ALLOC	Enables trace statements for all host memory allocations.
MEITraceMEMORY_FREE	Enables trace statements for all host memory de-allocations.
MEITraceMEMORY_GET	Enables trace statements for all controller memory reads.
MEITraceMEMORY_SET	Enables trace statements for all controller memory writes.
MEITraceVALIDATE	Enables trace statements for all function parameter validations.
MEITraceLOCK_GIVE	Enables trace statements for all IPC lock releases.
MEITraceLOCK_TAKE	Enables trace statements for all IPC lock takes.
MEITraceEVENT	Enables trace statements for all MPI Events.

See Also

Trace Object | Trace.exe utility

MEITraceFunction

Definition

```
typedef long (*MEITraceFunction) (const char *buffer);
```

Description

Definition for a trace function interface. **MEITraceFunction** can be used to define a custom trace output routine. MEITraceFunction function must take a pointer to a buffer as a parameter and must return a long.

See Also

meiTraceFunction

MEITraceMask

Definition

```
typedef   unsigned long MEITraceMask;
```

Description

MEITraceMask is a bit mask used to enable/disable library trace statement output.

See Also

meiTraceGet | meiTraceSet | MEITraceMaskGLOBAL

MEITraceMaskGLOBAL

Definition

```
extern  MEITraceMask  MEITraceMaskGLOBAL;
```

Description

MEITraceMaskGLOBAL is a non-object specific MPI Trace mask variable used for library wide Trace bits.

See Also

MEITraceMask

Trace Masks

Every MPI object contains an MEITraceMask and every process contains a single global MEITraceMask. An MEITraceMask consists of bits, where each bit corresponds to a single trace category. A trace category is a specific type of debug information that you want to be displayed by the MPI library. A trace category can be either global (applying to all MPI objects) or object-specific (applying only to a specific MPI object).

Trace Category	Is
global	declared by the MEITrace{...} enum in trace.h.
object-specific (for MPI objects)	declared in stdmei.h.
object-specific (for MEI objects)	declared in the object header file (for MEI objects). Note that the trace mask bits for object-specific trace categories overlap.

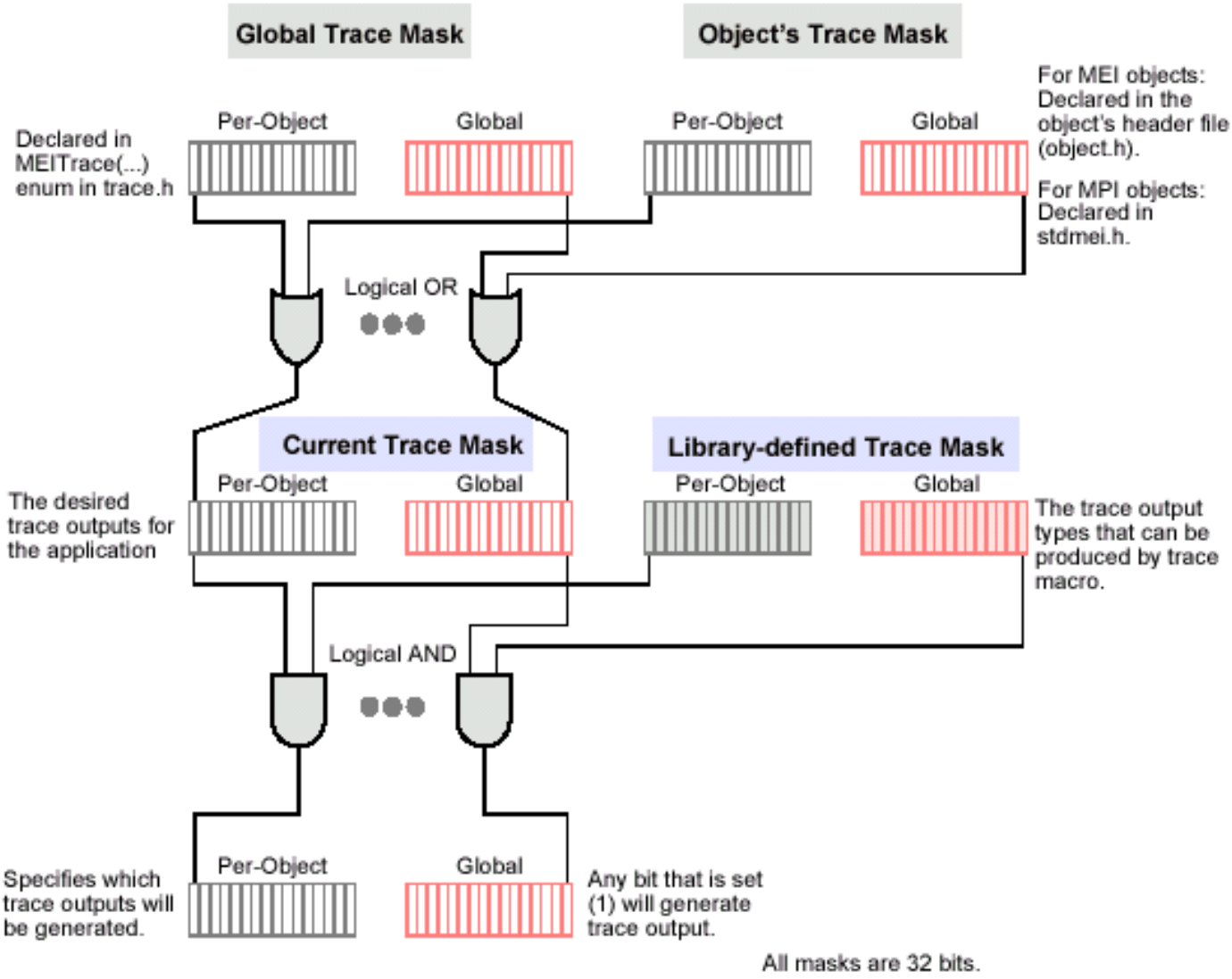
An object will produce trace output for a trace category when the logical **OR** of the **global trace mask** and the **object's trace mask** has the bit set that corresponds to the trace category.

If the global trace mask has all of its bits set, then all objects will display trace output for all trace categories.

If an object's trace mask has **all** of its bits set, then that object will display trace output for all trace categories, but a different object of the same type might produce less or no trace output depending on the setting of its trace mask. The setting of the global and object trace masks is under the control of your application.

The trace mask is derived in 2 steps:

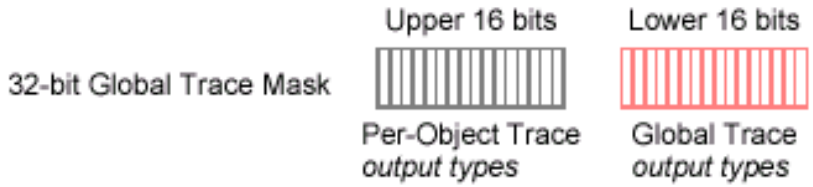
1. The **global trace mask** is logically **OR**ed with the **object trace mask**. This yields the **current trace mask**, representing the desired trace output types as specified by the application.
2. The **current trace mask** (from step1) is logically **AND**ed with a **library-defined trace mask** (that describes the trace output types for which the trace macro should produce output). If the result of the AND is non-zero, trace output will be produced using the ***format*** and the ***args*** [from **meiTrace#(mask, format, arg ...**



Return to Trace Object's page

Global Trace Outputs

There is a global 32-bit trace mask: the low 16 bits are the global trace output types, while the upper 16 bits are the per-object trace output types. Each object has a similar trace mask. The upper 16 bits of the global trace mask are not defined, but can be used to set the per-object output types for all objects. To enable all trace output types for all objects, set the global trace mask to all 1s (i.e., -1).



The **MEITrace{...}** enum (declared in trace.h) specifies the global types of trace output, i.e., the types of trace output that can be produced by any object or module. **The MEITrace{...}** enum defines constants that you use together as a bit mask. You specify the desired trace output as a combination (logical OR) of **MEITrace{...}** constants.

There are 16 possible types of global trace output, with 12 global trace outputs defined.

Output Type	Displays
MEITraceFUNCTION_ENTRY	Function name & calling parameters upon entry to function
MEITraceFUNCTION_RETURN	Function name, calling parameters & return value upon exit from function
MEITraceMEMORY_ALLOC	The Address & byte count when memory is dynamically allocated
MEITraceMEMORY_FREE	The Address & byte count when dynamically allocated memory is freed
MEITraceMEMORY_GET	Source address, destination address, byte count when reading XMP firmware memory
MEITraceMEMORY_SET	Source address, destination address, byte count when writing XMP firmware memory
MEITraceVALIDATE	Results of object validation
MEITraceLOCK_GIVE	When a resource lock is released
MEITraceLOCK_TAKE	When a resource lock is waited for & obtained
MEITraceEVENT	When an XMP event is received
MEITraceALL	All global trace outputs (lower 16 bits)

MEIModuleTraceALL	All per-object trace outputs (upper 16 bits)

Return to Trace Object's page

Per-Object Trace Outputs

There are 16 possible types of per-object trace output. Each object can declare up to 16 of its own trace output types. MPI modules declare per-object trace output types in stdmei.h. MEI modules declare per-object trace output types in the module header file.

Output Type	Displays
MEIMotionTraceSTATUS	Status of the Motion Supervisor
MEINotifyTrcaeTHREAD	When a thread goes to sleep or wakes up
MEISequenceTraceLOAD	When a batch of new commands are sent to the XMP Program Sequencer
MEIConfigTracePROGRESS	Displays "." as it executes (used by config utility)
MEIRecorderTraceRECORD_GET	When the Recorder gets records from the XMP
MEIRecorderTraceSTATUS	The number of data records available in the XMP

NOTE: The first 5 output types overlap in the mask.

Return to Trace Object's page

Xmp Module

Introduction

The **Xmp** module provides the low-level interface between the controller and the MPI library. It defines the shared memory access between the controller's processor and the host CPU. It also contains several hardware constants and maximum values for resources. ALL data transactions between the MPI and controller are defined by this module.

The MPI provides a layer between the application code and the controller. It protects the application from *xmp.h* changes, hides controller complexity, handles semaphore locking, performs data validation and range checking, plus many other features. Normally, an application does not need the Xmp module. An application should ALWAYS use the MPI to access the controller. In some cases, the MPI may not have methods/structures to support a controller feature. For example, a custom feature may require support for direct access to the MPI, but it has not been yet been developed. In these cases, an application can use xmp.h and mpiControlMemoryGet/Set to directly access the controller.

WARNING!
The *xmp.h* file is version dependent. Make sure to ONLY use the xmp.h that was included with the MPI and controller firmware software package. Using mismatched xmp.h defines can cause unexpected and potentially dangerous behavior!

Be aware that the xmp.h is always changing. It is an internal file, used by the controller firmware and MPI. It is optimized for memory allocation and performance for the controller's processor. As new features are developed and improved, the xmp.h is modified. If you use xmp.h defines in your application code, make sure to check for changes when upgrading to new software releases.

Data Types

MEIXmp**CommMode**

MEIXmp**SwitchType**

MEIXmpCommMode

Declaration

```
typedef enum {
    MEIXmpCommModeNONE,
    MEIXmpCommModeCLOSED_LOOP,
    MEIXmpCommModeOPEN_LOOP,
    MEIXmpCommModeSIMULATE,
} MEIXmpCommMode;
```

Required Header: xmp.h

Description

MEIXmpCommMode is an enumeration that contains the different options for controller based sinusoidal commutation. It is important to understand that drive-based sinusoidal commutation is not covered in this enumeration. The value used for drive-based sinusoidal commutation is MEIXmpCommModeNONE.

Ways to tell if you are using drive or controller based sinusoidal commutation:

Drive-based Sinusoidal Commutation - only use MEIXmpCommModeNONE

- You are using a SynqNet drive.
- You use drive based commutation settings.
- You are using drive based commutation initialization.

Controller-based Sinusoidal Commutation

- You are not using a SynqNet drive.
- You are using a torque or current mode drive.
- You have two +/- 10 V current command inputs to your drive.
- You are using a SynqNet node that can provide two +/- 10 V outputs per motor (MEI-RMB-10V2, for example).
- You are using sinusoidal commutation.
- You drive does not provide sinusoidal commutation.
- You are not using brushed motors.
- Your drive is configured to accept two +/- 10V inputs for sinusoidal commutation.

Sinusoidal commutation requires initialization of some sort. If you are using Controller based sinusoidal commutation, please see Sinusoidal Commutation. There are three types of sinusoidal commutation initialization routines supported by the XMP/ZMP controller:

- Step (scstep.c)
- Hall (schall.c)
- Dither (scdither.c)

MEIXmpCommModeNONE	Default mode. This mode is used for: • All drives that perform their own commutation. • Any control loop that only requires a single command signal.
MEIXmpCommModeCLOSED_LOOP	Used during controller calculated sinusoidal commutation. This mode uses two command signals to command a two, three, or four phase sinusoidal commutation mode. This mode is most often used with an MEI RMB connected to a drive that has two ±10V inputs for torque command.
MEIXmpCommModeOPEN_LOOP	Used during sinusoidal commutation initialization before going to CLOSED_LOOP. This mode is only used when hall sensors are not available to go directly to CLOSED_LOOP mode.
MEIXmpCommModeSIMULATE	MEI internal mode to simulate a motor. Not used for actual motor control.

See Also

MEIXmpSwitchType

Declaration

```
typedef enum {
    MEIXmpSwitchTypeNONE,
    MEIXmpSwitchTypeMOTION_ONLY,
    MEIXmpSwitchTypeWINDOW,
    MEIXmpSwitchTypeUSER,
} MEIXmpSwitchType;
```

Required Header: xmp.h

Description

MEIXmpSwitchType is an enumeration for gain scheduling that determines the gain scheduling mode. Only MEIXmpSwitchTypeNONE and MEIXmpSwitchTypeMOTION_ONLY are available in standard firmware types.

Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV). To change the gain scheduling type from *none* (uses only the gains in gain table index 0), use MEIFilterConfig. GainSwitchType is set with mpiFilterConfigSet(...).

When setting filter gain parameters using mpiFilterGainGet(...) and mpiFilterGainSet(...), use the gain index value to write to a gain index of your choosing.

MEIXmpSwitchTypeNONE	Default value in factory default firmware. This mode uses mpiFilterGainIndexSet() and mpiFilterGainIndexGet() to manipulate the gain index manually, if desired.
MEIXmpSwitchTypeMOTION_ONLY	Switch gains based on controller's switching algorithm.

MEIFilterGainIndex (go to MEIFilterGainIndex)

MEIFilterGainIndexNO_MOTION	When command velocity = 0
MEIFilterGainIndexACCEL	When command acceleration > 0
MEIFilterGainIndexDECEL	When command acceleration < 0
MEIFilterGainIndexVELOCITY	When command velocity > 0 and command acceleration = 0 The firmware automatically takes care of this switching. Be aware when checking the gain index, that the firmware can change the gain index in real time.

Description

Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV). To change the gain scheduling type from NONE (uses only the gains in gain table index 0), use MEIFilterConfig.GainSwitchType, which is set with mpiFilterConfigSet(...).

Use mpiFilterConfigSet(...) to change MEIFilterConfig.GainSwitchType to one of the MEIXmpSwitchType enumerations to change the gain scheduling mode.

See Also

MPIFilterConfig | mpiFilterConfigGet | mpiFilterConfigSet | MEIFilterGainIndex | mpiFilterGainIndexSet | mpiFilterGainIndexGet | mpiFilterGainGet | mpiFilterGainSet

meiDeprecated Objects

Introduction

In an effort to provide the most comprehensive library of functions, the MPI library is constantly being updated and improved. In every software release, there may be a variety of additions, revisions, and deletions to the header files. However, in order to maintain backwards compatibility, selected data types and functions have been preserved in meiDeprecated.h and are still supported. Although the data types and methods listed below are still supported, their use in future applications is discouraged.

Methods

Data Types

MEICanDigitalIO
MPIControlIoWords
MPIControlIo
MEIControlVersion
MEIMotorDedicatedIn
MEIMotorInput
MEIMotorDedicatedOut
MEIMotorDedicatedOVERTRAVEL_POS
MEIMotorDedicatedOVERTRAVEL_NEG
MEIMotorDedicatedInINDEX
MEIMotorGeneralIo
MEIMotorInput
MEIMotorIoMask
MPIMotorIo
MEIFilterDataType
MEISynqNetMessageNODE_UNAVAILABLE
MEISynqNetMessageRESPONSE_TIMEOUT
MEISynqNetMessageREADY_TIMEOUT
MEISynqNetMessageSRVC_ERROR
MEISynqNetMessageSRVC_UNSUPPORTED
MEISynqNetMaxMotorENCODER_COUNT

Constants

MEIMotorDedicatedOVERTRAVEL_POS
MEIMotorDedicatedOVERTRAVEL_NEG

meiCanFlashNodeConfigGet (Deprecated)

NOTE:
meiCanFlashNodeConfigGet was moved to meiDeprecated.h in the 03.01.00 MPI software release.

Declaration

```
long  meiCanFlashNodeConfigGet(MEICan            can,
                               void*             flash,
                               long              node,
                               MEICanNodeConfig* nodeConfig);
```

Required Header: stdmei.h

Description

meiCanFlashNodeConfigGet returns a copy of the current flash configuration of the CAN controller.

can	a handle to the Can object
flash	normally NULL
node	the node number of the CANOpen node
nodeConfig	a pointer to the CAN node configuration structure that will be filled in by this function

Return Values	
MPIMessageOK	

See Also

meiCanNodeFlashConfigSet

meiCanFlashNodeConfigSet (Deprecated)

NOTE:
meiCanFlashNodeConfigSet was moved to meiDeprecated.h in the 03.01.00 MPI software release.

Declaration

```
long  meiCanFlashNodeConfigSet(MEICan            can,
                               void*             flash,
                               long              node,
                               MEICanNodeConfig* nodeConfig);
```

Required Header: stdmei.h

Description

meiCanFlashNodeConfigSet updates the current flash configuration for the node.

can	a handle to the Can object
flash	normally NULL
node	the node number of the CANOpen node
nodeConfig	a pointer to the CAN node configuration structure containing the new configuration

Return Values	
MPIMessageOK	

See Also

meiCanNodeFlashConfigGet

meiCanNodeDigitalInputGet (Deprecated)

NOTE:
meiCanNodeDigitalInputGet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long meiCanNodeDigitalInputGet(MEICan    can,
                               long      node,
                               long      bit,
                               long*     data);
```

Required Header: stdmei.h

Description

meiCanNodeDigitalInputGet gets the current state of the digital input bit on the specified CAN node.

(Not to be confused with meiCanNodeDigitalInputsGet.)

can	Handle to the CAN object.
node	The node number of the CANOpen node.
bit	Which bit on this node. This value should be between 0 and 63.
data	A pointer to where the current digital bit is returned. The value returned will be either 0 or 1.

Return Values	
MPIMessageOK	

Sample Code

The following sample code shows how to interrogate the current state of a single digital input bit on a controller. The variable **data** will contain either one or zero depending on the electrical signal being applied to the input pin on the CANOpen node. See meiCanCreate on how to create the CANHandle.

```
long data;
long Result;
Result = meiCanNodeDigitalInputGet(CANHandle,
                                   3, /*node*/
                                   0, /*bit*/
                                   &data );
```

See Also

meiCanCreate

meiCanNodeDigitalInputsGet (Deprecated)

NOTE:
meiCanNodeDigitalInputsGet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long meiCanNodeDigitalInputsGet(MEICan          can,
                                long            node,
                                MEICanDigitalIO* data);
```

Required Header: stdmei.h

Description

meiCanNodeDigitalInputsGet gets the current state of all the digital input bits on the specified CAN node.

(Not to be confused with meiCanNodeDigitalInputGet.)

can	handle to the CAN object
node	the node number of the CANOpen node.
data	a pointer to where the current digital bits are returned.

Return Values	
MPIMessageOK	

See Also

meiCanNodeDigitalInputGet

meiCanNodeDigitalOutputGet (Deprecated)

NOTE:
meiCanNodeDigitalOutputGet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long meiCanNodeDigitalOutputGet(MEICan    can,
                                long      node,
                                long      bit,
                                long*     data);
```

Required Header: stdmei.h

Description

meiCanNodeDigitalOutputGet gets the current state of the digital output bit on the specified CAN Node.

(Not to be confused with meiCanNodeDigitalOutputsGet.)

can	handle to the CAN object
node	the node number of the CANOpen node.
bit	which bit on this node.
data	a pointer to where the current digital bit is returned.

Return Values	
MPIMessageOK	

See Also

meiCanNodeDigitalOutputSet | meiCanNodeDigitalOutputsGet | meiCanNodeDigitalOutputsSet

meiCanNodeDigitalOutputsGet (Deprecated)

NOTE:
meiCanNodeDigitalOutputsGet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long meiCanNodeDigitalOutputsGet(MEICan            can,
                                 long              node,
                                 MEICanDigitalIO*  data);
```

Required Header: stdmei.h

Description

meiCanNodeDigitalOutputsGet gets the current state of all the digital output bits on the specified CAN node.

(Not to be confused with meiCanNodeDigitalOutputGet.)

can	handle to the CAN object
node	the node number of the CANOpen node.
data	a pointer to where the current digital bit is returned.

Return Values	
MPIMessageOK	

See Also

meiCanNodeDigitalOutputGet | meiCanNodeDigitalOutputSet | meiCanNodeDigitalOutputsSet

meiCanNodeDigitalOutputSet (Deprecated)

NOTE:
meiCanNodeDigitalOutputSet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long meiCanNodeDigitalOutputSet(MEICan    can,
                                long      node,
                                long      bit,
                                long      data);
```

Required Header: stdmei.h

Description

meiCanNodeDigitalOutputSet changes the state of the digital output bit on the specified CAN Node.

(Not to be confused with meiCanNodeDigitalOutputsSet.)

can	Handle to the CAN object
node	The node number of the CANOpen node.
bit	Which bit on this node. This value should be between 0 and 63.
data	The new state of the digital bit. The value returned will be either 0 or 1.

Return Values	
MPIMessageOK	

See Also

meiCanNodeDigitalOutputGet | meiCanNodeDigitalOutputsGet | meiCanNodeDigitalOutputsSet

meiCanNodeDigitalOutputsSet (Deprecated)

NOTE:
meiCanNodeDigitalOutputsSet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long meiCanNodeDigitalOutputsSet(MEICan             can,
                                 long               node,
                                 MEICanDigitalIO*   data);
```

Required Header: stdmei.h

Description

meiCanNodeDigitalOutputsSet changes the current state of all the digital output bits on the specified CAN node.

(Not to be confused with meiCanNodeDigitalOutputSet.)

can	handle to the CAN object
node	the node number of the CANOpen node.
data	the new data of the digital bits.

Return Values	
MPIMessageOK	

See Also

meiCanNodeDigitalOutputGet | meiCanNodeDigitalOutputSet | meiCanNodeDigitalOutputsGet

meiControlEnabledLmbGet (Deprecated)

NOTE:
meiControlEnabledLmbGet was moved to meiDeprecated.h in the 20030420 MPI software release.

Declaration

```
long  meiControlEnabledLmbGet  (MPIControl   control,
                                 long        *count);
```

Required Header: stdmei.h

Description

meiControlEnabledLmbGet is for backwards compatability only (always returns zero).

control	a handle to the Control object.
***count**	a pointer to the location at which to write the number of enabled Local Motion Blocks (LMBs).

Return Values	
MPIMessageOK	

See Also

meiControlEnabledLmbSet

meiControlEnabledLmbSet (Deprecated)

NOTE:
meiControlEnabledLmbSet was moved to meiDeprecated.h in the 20030420 MPI software release.

Declaration

```
/* for backwards compatability only (does nothing) */
long  meiControlEnabledLmbSet  (MPIControl   control,
                                long         *count);
```

Required Header: stdmei.h

Description

meiControlEnabledLmbSet is for backwards compatability only. It does nothing.

control	a handle to the Control object.
***count**	a pointer to the location that stores the number of Local Motion Blocks (LMBs) to enable.

Return Values	
MPIMessageOK	

See Also

meiControlEnabledLmbGet

mpiControlIoGet (Deprecated)

NOTE:
mpiControlIoGet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long mpiControlIoGet(MPIControl       control,
                     MPIControlIo     *io);
```

Required Header: stdmei.h

Description

mpiControlIoGet reads the states of a controller's digital inputs and writes them into a structure pointed to by ***io***. Some controller models have local digital I/O. Please see the controller hardware documentation for details.

control	a handle to a Control object
***io**	a pointer to a structure containing the digital input and output values.

Return Values	
MPIMessageOK	

See Also

mpiControlIoSet | MPIControlInput | MPIControlOutput

mpiControlIoSet (Deprecated)

NOTE:
mpiControlIoSet was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long mpiControlIoSet(MPIControl       control,
                     MPIControlIo     *io);
```

Required Header: stdmei.h

Description

mpiControlIoSet writes the states of a controller's digital I/O using data from a structure pointed to by ***io***. Some controller models have local digital I/O. Please see the controller hardware documentation for details.

control	a handle to a Control object
***io**	a pointer to a structure containing the digital input and output values.

Return Values	
MPIMessageOK	

See Also

mpiControlIoGet | MPIControlInput | MPIControlOutput

meiControlIoBitGet (Deprecated)

NOTE:
meiControlIoBitGet was moved to meiDeprecated.h in the 03.03.00 MPI software release.
Please see Transitioning to the New Motor I/O Functions.

Declaration

```
long  meiControlIoBitGet(MPIControl          control,
                         MEIControlIoBit     bit,
                         MPI_BOOL            *value);
```

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

Description

meiControlIoBitGet reads the state of a controller digital input bit and writes it into a long pointed to by ***value***. Some controller models have local digital I/O. Please see the controller hardware documentation for details.

control	a handle to the Control object
bit	an enumerated bit number
***value**	a pointer to a long. The value contains the state of the bit.

Return Values	
MPIMessageOK	

See Also

meiControlIoBitSet | MEIControlIoBit

meiControlIoBitSet (Deprecated)

NOTE:
meiControlIoBitSet was moved to meiDeprecated.h in the 03.03.00 MPI software release.
Please see Transitioning to the New Motor I/O Functions.

Declaration

```
long  meiControlIoBitSet(MPIControl          control,
                         MEIControlIoBit     bit,
                         MPI_BOOL            *value);
```

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

Description

meiControlIoBitSet writes the state of a controller digital output bit using data from a value pointed to by a long. Some controller models have local digital I/O. Please see the controller hardware documentation for details.

control	a handle to the Control object
bit	an enumerated bit number
***value**	a pointer to a long. The value contains the state of the bit.

Return Values	
MPIMessageOK	

See Also

meiControlIoBitGet | MEIControlIoBit

meiControlVersionGet (Deprecated)

NOTE:
meiControlVersionGet was moved to meiDeprecated.h in the 03.01.00 MPI software release.

Declaration

```
long  meiControlVersionGet(MPIControl          control,
                           MEIControlVersion   *version);
```

Required Header: stdmei.h

Description

meiControlVersionGet writes the the version numbers of the XMP firmware, hardware, and the MPI library to the structure pointed to by ***version***.

Return Values	
MPIMessageOK	

See Also

meiControlVersionSet

meiControlVersionSet (Deprecated)

NOTE:
meiControlVersionSet was moved to meiDeprecated.h in the 03.01.00 MPI software release.

Declaration

```
long  meiControlVersionSet(MPIControl          control,
                           MEIControlVersion   *version);
```

Required Header: stdmei.h

Description

meiControlVersionSet sets the version numbers of the XMP firmware, hardware, and the MPI library using data from the structure pointed to by version.

Normally, the MPI library is compatible only with the XMP firmware for which the library is specifically built; i.e., only when

```
version -> mpi.firmware.version == version -> xmp.firmware.version
```

However, there are times when it is desirable to have the MPI library ignore incompatible firmware and continue to operate. As an example, the flash utility instructs the MPI library to ignore firmware incompatibility when new firmware is being loaded. Of course, this new firmware should also be compatible with the MPI library. In such cases, the version -> xmp.firmware structure will be copied into ***control***.

Return Values	
MPIMessageOK	

See Also

meiControlVersionGet

mpiMotorIoGet (Deprecated)

NOTE:
mpiMotorIoGet was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Declaration

```
long  mpiMotorIoGet(MPIMotor    motor,
                    MPIMotorIo  *io);
```

Required Header: stdmei.h

Description

mpiMotorIoGet gets a Motor's (***motor***) dedicated I/O bits and writes them into the structure pointed to by ***io***.

NOTE: When using I/O on SynqNet nodes, use the motor I/O masks in the drive's header file for cleaner code. Most SynqNet nodes have a header file that defines things that are specific to that drive. The header files are found in C:\mei\XMP\sqNodeLib\include. An example is shown below that reads the hall sensors from the Trust TA802.

```
/*
        Shows the hall sensors.
        Make sure you include trust_ta800.h. Ex:
        #include "C:\mei\XMP\sqNodeLib\include\trust_ta800.h"

        The hall sensor masks are found in the enum TA800MotorIoMask.
*/

void showHalls(MPIMotor motor)
{
    MPIMotorIo io;
      long returnValue;
      long a, b, c;

      while (meiPlatformKey(MPIWaitPOLL) <= 0)
      {
         returnValue = mpiMotorIoGet(motor, &io);
         msgCHECK(returnValue);

         a = ((io.input & TA800MotorIoMaskHALL_A) == TA800MotorIoMaskHALL_A);
         b = ((io.input & TA800MotorIoMaskHALL_B) == TA800MotorIoMaskHALL_B);
         c = ((io.input & TA800MotorIoMaskHALL_C) == TA800MotorIoMaskHALL_C);

         /* Prints a 1 or 0 indicating the hall state */
```

```
         printf("Hall A %d B %d C %d\t\t\r", a, b, c);

         meiPlatformSleep(100);
      }
}
```

Sample Code

```
/* Poll io inputs for a motor and print to the screen */
void readIo(MPIMotor motor)
{
    MPIMotorIo io;
    long returnValue;

    while (meiPlatformKey(MPIWaitPOLL) <= 0)
    {
        returnValue = mpiMotorIoGet(motor, &io);
        msgCHECK(returnValue);

        printf("\rIO %x", io.input);

        meiPlatformSleep(100); /* Wait 100 mSec */
    }
}
```

Return Values	
MPIMessageOK	

See Also

mpiMotorIoSet

mpiMotorIoSet (Deprecated)

NOTE:
mpiMotorIoSet was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Declaration

```
long  mpiMotorIoSet(MPIMotor    motor,
                    MPIMotorIo  *io);
```

Required Header: stdmei.h

Description

mpiMotorIoSet sets a Motor's (***motor***) dedicated I/O bits using data from the structure pointed to by ***io***.

NOTE: When using I/O on SynqNet nodes, use the motor I/O masks in the drive's header file for clearer code. Most SynqNet nodes have a header file that defines things that are specific to that drive. The header files are found in C:\mei\XMP\sqNodeLib\include. An example is shown below that reads the hall sensors from the Trust TA802.

```
/*
        Shows the hall sensors.
        Make sure you include trust_ta800.h. Ex:
        #include "C:\mei\XMP\sqNodeLib\include\trust_ta800.h"

        The hall sensor masks are found in the enum TA800MotorIoMask.
*/

void showHalls(MPIMotor motor)
{
    MPIMotorIo io;
      long returnValue;
      long a, b, c;

      while (meiPlatformKey(MPIWaitPOLL) <= 0)
      {
         returnValue = mpiMotorIoGet(motor, &io);
         msgCHECK(returnValue);

         a = ((io.input & TA800MotorIoMaskHALL_A) == TA800MotorIoMaskHALL_A);
         b = ((io.input & TA800MotorIoMaskHALL_B) == TA800MotorIoMaskHALL_B);
         c = ((io.input & TA800MotorIoMaskHALL_C) == TA800MotorIoMaskHALL_C);

         /* Prints a 1 or 0 indicating the hall state */
         printf("Hall A %d B %d C %d\t\t\r", a, b, c);
```

```
            meiPlatformSleep(100);
        }
}
```

Return Values	
MPIMessageOK	

See Also

mpiMotorIoGet

mpiMotorFeedbackGet (Deprecated)

NOTE:
mpiMotorFeedbackGet was moved to meiDeprecated.h in the 20030402 MPI software release.

Declaration

```
long  mpiMotorFeedbackGet(MPIMotor   motor,  /* use mpiMotorFeedback(...) */
                          double     feedback);
```

Required Header: stdmei.h

Description

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

Return Values	
MPIMessageOK	

See Also

meiPlatformCreate (Deprecated)

NOTE:
meiPlatformCreate was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
MEIPlatform  meiPlatformCreate(MPIControl  control);
```

Required Header: stdmei.h

Description

meiPlatformCreate is for internal purposes only.

Customers should NOT use meiPlatformCreate in motion applications.

meiPlatformDelete (Deprecated)

NOTE:
meiPlatformDelete was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Declaration

```
long  meiPlatformDelete(MEIPlatform   platform);
```

Required Header: stdmei.h

Description

meiPlatformDelete is for internal purposes only.

Customers should NOT use meiPlatformCreate in motion applications.

meiSqNodeAnalogInput (Deprecated)

NOTE:
meiSqNodeAnalogInput was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Declaration

```
long  meiSqNodeAnalogInput(MEISqNode   node,
                           long        channel,
                           double      *state);
```

Required Header: stdmei.h

Description

meiSqNodeAnalogInput gets the current state of an analog input on a SynqNet node.

node	a handle to a SynqNet node object.
channel	the index of the analog input channel (with respect to the node).
***state**	a pointer to where the current state of the input is written by this function.

Return Values	
MPIMessageOK	

See Also

MEICanDigitalIO (Deprecated)

NOTE:
MEICanDigitalIO was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Definition

```
typedef struct MEICanDigitalIO {
    unsigned   long    data[2];
} MEICanDigitalIO;
```

Description

MEICanDigitalIO holds the state of all the digital inputs or outputs on a CANOpen node. The maximum number of inputs or outputs on a single node supports is 64.

data	Data associated with the command.

See Also

MPIControlIoWords (Deprecated)

NOTE:
MPIControlIoWords was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Definition

```
#define  MPIControlIoWords   (1)
```

Description

MPIControlIoWords defines the number of 32 bit Input and Output words on the controller.

See Also

MPIControlIo | MEIControlIoBit | MEIControlInput | MEIControlOutput

MPIControlIo (Deprecated)

NOTE:
MPIControlIo was moved to meiDeprecated.h in the 03.03.00 MPI software release.

Definition

```
typedef struct MPIControlIo {
    unsigned long   input[MPIControlIoWords];
    unsigned long   output[MPIControlIoWords]
} MPIControlIo;
```

Description

MPIControlIo contains the controller's local digital input and output states. The digital inputs can be read and the digital outputs can be read or written through this structure.

input	An array of digital input values. Each bit mask is defined by the MEIControlInput enumeration.
output	An array of digital output values. Each bit mask is defined by the MEIControlOutput enumeration.

See Also

MEIControlOutput | MEIControlInput | mpiControlIoGet | mpiControlIoSet

MEIControlVersion (Deprecated)

NOTE:
MEIControlVersion was moved to meiDeprecated.h in the 03.01.00 MPI software release. It was replaced by MEIControlInfo.

Definition

```
/* replaced by MEIControlInfo */
typedef struct MEIControlVersion {
    struct {   /* control.c */
        char   *version;   /* MEIControlVersionMPI (YYYYMMDD) */


        struct {   /* xmp.h */
            long    version;    /* MEIXmpVERSION */
            long    option;     /* MEIXmpOPTION */
        } firmware;
    } mpi;

    struct {
        long    version; /* hardware version */

        struct {   /* MEIXmpData.SystemData{} */
            long    version;    /* MEIXmpVERSION_EXTRACT(SoftwareID) */
            char    revision;   /* ('A' - 1) + MEIXmpREVISION_EXTRACT(SoftwareID) */
            long    subRevision;    /* MEIXmpSUB_REV_EXTRACT(Option) */
            long    developmentId;  /* MEIXmpDEVELOPMENT_ID_EXTRACT(Option) */
            long    option;         /* MEIXmpOPTION_EXTRACT(Option) */
            long    userVersion;
            long    branchId;
     } firmware;

     struct {
        struct {
            long    version;
            long    option;
          } PLD;
          struct {
             char number[10];
             char rev[5];
          } model;
          struct {
             long    version;
          } FPGA;
       } board;
    } xmp;
    struct {
       char version[10];
    } driver;

} MEIControlVersion;
```

Description

MEIControlVersion is a structure that specifies the version information for the MPI and the controller's firmware, FPGAs, and the bus interface.

mpi	A structure that contains the version information of the MPI.
mpi.version	A string representing the version of the MPI. The version of the MPI is broken down by date, branch, and revision (MPIVersion.branch.revision). For ex: 20021220.1.2 means MPI version 20021220, branch 1, revision 2.
mpi.firmware	The firmware version information that the current version of the MPI will work with. A new field has been added to the XMP's firmware to identify and differentiate between intermediate branch software revisions. The branch value is represented as a hex number between 0x00000000 and 0xFFFFFFFF. Each digit represents an instance of a branch (0x1 to 0xF). A single digit represents a single branch from a specific version, two digits represent a branch of a branch, three digits represent a branch of a branch of a branch, etc.
xmp	A structure that contains the version information of the XMP controller.
xmp.firmware	The XMP's firmware version information.
xmp.motionBlock[]	An array of structures that contain version information about the motion blocks on the XMP.
xmp.board.PLD.version	A string that contains the controller's PLD version.
xmp.board.PLD.option	A string that contains the controller's PLD option.
xmp.board	An array of structures that contain version information about the XMP controller boards.
xmp.board.model.number	A string that contains the hardware model number for the controller.
xmp.board.model.rev	A string that contains the hardware revision for the controller.
xmp.board.FPGA.version	A string that contains the controller's FPGA version.
driver.version	A string containing device driver version information. This values is "N/A" if the driver version is not available from your operating system.

See Also

MPIControl | MEIControlInfo

MEIMotorDedicatedIn (Deprecated)

NOTE:
MEIMotorDedicatedIn was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Definition

```
typedef enum {
    MEIMotorDedicatedInAMP_FAULT       =  MEIXmpMotorDedicatedFlagsMaskAMP_FAULT,  /
* bit 0  */
    MEIMotorDedicatedInBRAKE_APPLIED   =  MEIXmpMotorDedicatedFlagsMaskBRAKE_ON,   /
* bit 1  */
    MEIMotorDedicatedInHOME            =  MEIXmpMotorDedicatedFlagsMaskHOME,       /
* bit 2  */
    MEIMotorDedicatedInLIMIT_HW_POS    =  MEIXmpMotorDedicatedFlagsMaskPOS_LIMIT,  /
* bit 3  */
    MEIMotorDedicatedInLIMIT_HW_NEG    =  MEIXmpMotorDedicatedFlagsMaskNEG_LIMIT,  /
* bit 4  */
    MEIMotorDedicatedInINDEX_PRIMARY   =  MEIXmpMotorDedicatedFlagsMaskENC_INDEX0, /
* bit 5  */
    MEIMotorDedicatedInFEEDBACK_FAULT  =  MEIXmpMotorDedicatedFlagsMaskENC_FAULT,  /
* bit 6  */
    MEIMotorDedicatedInCAPTURED        =  MEIXmpMotorDedicatedFlagsMaskCAPTURED,   /
* bit 7  */
    MEIMotorDedicatedInHALL_A          =  MEIXmpMotorDedicatedFlagsMaskHALL_A,     /
* bit 8  */
    MEIMotorDedicatedInHALL_B          =  MEIXmpMotorDedicatedFlagsMaskHALL_B,     /
* bit 9  */
    MEIMotorDedicatedInHALL_C          =  MEIXmpMotorDedicatedFlagsMaskHALL_C,     /
* bit 10 */
    MEIMotorDedicatedInAMP_ACTIVE      =  MEIXmpMotorDedicatedFlagsMaskAMP_ACTIVE, /
* bit 11 */
    MEIMotorDedicatedInINDEX_SECONDARY =
MEIXmpMotorDedicatedFlagsMaskENC_INDEX1,     /* bit 12 */
    MEIMotorDedicatedInWARNING         =
MEIXmpMotorDedicatedFlagsMaskWARNING,        /* bit 13 */
    MEIMotorDedicatedInDRIVE_STATUS_9  =
MEIXmpMotorDedicatedFlagsMaskDRIVE_STATUS9,  /* bit 14 */
    MEIMotorDedicatedInDRIVE_STATUS_10 =
MEIXmpMotorDedicatedFlagsMaskDRIVE_STATUS10, /* bit 15 */
} MEIMotorDedicatedIn;
```

Description

MEIMotorDedicatedIn is an enumeration of bit masks for the motor's dedicated inputs. The support for dedicated inputs is node/drive specific. See the node/drive manufacturer's documentation for details.

MEIMotorDedicatedInAMP_FAULT	Generated by the masked motor fault bits. Active when one or more masked motor faults bits are active. See MEIMotorFaultConfig.
MEIMotorDedicatedInBRAKE_APPLIED	Mechanical brake state.
MEIMotorDedicatedInHOME	Position calibration sensor.
MEIMotorDedicatedInLIMIT_HW_POS	Hardware limit for the positive direction.
MEIMotorDedicatedInLIMIT_HW_NEG	Hardware limit for the negative direction.
MEIMotorDedicatedInINDEX_PRIMARY	Primary encoder index input signal.
MEIMotorDedicatedInFEEDBACK_FAULT	Position feedback status. TRUE when position feedback fails, FALSE when operating properly.
MEIMotorDedicatedInCAPTURED	Currently not supported.
MEIMotorDedicatedInHALL_A	Reflects the state of Hall Sensor A
MEIMotorDedicatedInHALL_B	Reflects the state of Hall Sensor B
MEIMotorDedicatedInHALL_C	Reflects the state of Hall Sensor C
MEIMotorDedicatedInAMP_ACTIVE	A bit set by the drive that indicates the amplifier's state. 1 = Amplifier is closing the current loop and the motor winding are energized. 0 = Amplifier is not closing the current loop and the motor windings are not energized. Support for this bit varies depending on the drive type.
MEIMotorDedicatedInINDEX_SECONDARY	Secondary encoder index input signal.
MEIMotorDedicatedInWARNING	Drive warning state. 1 = drive warning status bit is active and warning message is available 0 = drive warning status bit is not active. Support for this bit varies depending on the drive type.
MEIMotorDedicatedInDRIVE_STATUS_9	State of bit 9 in the SynqNet drive specific status register.
MEIMotorDedicatedInDRIVE_STATUS_10	State of bit 10 in the SynqNet drive specific status register.

See Also

mpiMotorIoGet

MEIMotorInput (Deprecated)

NOTE:
MEIMototInput was moved to meiDeprecated.h in the 03.02.00 MPI software release. It was replaced by MEIMotorDedicatedIn.

Definition

```
typedef enum {
    MEIMotorInputOVERTRAVEL_POS = MEIMotorDedicatedInLIMIT_HW_POS,
    MEIMotorInputOVERTRAVEL_NEG = MEIMotorDedicatedInLIMIT_HW_NEG,
    MEIMotorInputHOME           = MEIMotorDedicatedInHOME,
    MEIMotorInputAMP_FAULT      = MEIMotorDedicatedInAMP_FAULT,
    MEIMotorInputINDEX          = MEIMotorDedicatedInINDEX_PRIMARY,
} MEIMotorInput;
```

Description

MEIMotorInputOVERTRAVEL_POS	See MEIMotorDedicatedInLIMIT_HW_POS.
MEIMotorInputOVERTRAVEL_NEG	See MEIMotorDedicatedInLIMIT_HW_NEG.
MEIMotorInputHOME	See MEIMotorDedicatedInHOME.
MEIMotorInputAMP_FAULT	See MEIMotorDedicatedInAMP_FAULT.
MEIMotorInputINDEX	See MEIMotorDedicatedInINDEX_PRIMARY.

See Also

MEIMotorDedicatedIn

MEIMotorDedicatedOut (Deprecated)

NOTE:
MEIMotorDedicatedOut was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Definition

```
typedef enum {
    MEIMotorDedicatedOutAMP_ENABLE     =
MEIXmpMotorDedicatedFlagsMaskAMP_ENABLE,    /* bit 0 */
    MEIMotorDedicatedOutBRAKE_RELEASE  =
MEIXmpMotorDedicatedFlagsMaskBRAKE_RELEASE, /* bit 1 */
} MEIMotorDedicatedOut;
```

Description

MEIMotorDedicatedOut is an enumeration of bit masks for the motor's dedicated outputs. The support for dedicated outputs is node/drive specific. See the node/drive manufacturer's documentation for details.

MEIMotorDedicatedOutAMP_ENABLE	Enable/disable drive or amplifier. Drive is enabled when TRUE, disabled when FALSE.
MEIMotorDedicatedOutBRAKE_RELEASE	Enable/disable mechanical brake. Brake is released (motor shaft is free) when TRUE, engaged when FALSE.

See Also

mpiMotorIoGet | mpiMotorIoSet

MEIMotorGeneralIo (Deprecated)

NOTE:
MEIMotorGeneralIo was moved to meiDeprecated.h in the 03.02.00 MPI software release. See MPIMotorGeneralIo.

Definition

```
typedef enum MEIMotorGeneralIo {
  MEIMotorGeneralIoINVALID = -1
  MEIMotorGeneralIo0,
  MEIMotorGeneralIo1,
  MEIMotorGeneralIo2,
  MEIMotorGeneralIo3,
  MEIMotorGeneralIo4,
  MEIMotorGeneralIo5,
  MEIMotorGeneralIo6,
  MEIMotorGeneralIo7,
  MEIMotorGeneralIo8,
  MEIMotorGeneralIo9,
  MEIMotorGeneralIo10,
  MEIMotorGeneralIo11,
  MEIMotorGeneralIo12,
  MEIMotorGeneralIo13,
  MEIMotorGeneralIo14,
  MEIMotorGeneralIo15,

} MEIMotorGeneralIo;
```

Description

MEIMotorGeneralIo enumeration gives labels for each of the general purpose outputs that a motor can support.

See Also

MEIMotorIoMask (Deprecated)

NOTE:
MEIMotorIoMask was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Definition

```
typedef enum {
    MEIMotorIoMask0  =  MEIXmpMotorIOMaskConfigurable0,  /* bit 16 */
    MEIMotorIoMask1  =  MEIXmpMotorIOMaskConfigurable1,  /* bit 17 */
    MEIMotorIoMask2  =  MEIXmpMotorIOMaskConfigurable2,  /* bit 18 */
    MEIMotorIoMask3  =  MEIXmpMotorIOMaskConfigurable3,  /* bit 19 */
    MEIMotorIoMask4  =  MEIXmpMotorIOMaskConfigurable4,  /* bit 20 */
    MEIMotorIoMask5  =  MEIXmpMotorIOMaskConfigurable5,  /* bit 21 */
    MEIMotorIoMask6  =  MEIXmpMotorIOMaskConfigurable6,  /* bit 22 */
    MEIMotorIoMask7  =  MEIXmpMotorIOMaskConfigurable7,  /* bit 23 */
    MEIMotorIoMask8  =  MEIXmpMotorIOMaskConfigurable8,  /* bit 24 */
    MEIMotorIoMask9  =  MEIXmpMotorIOMaskConfigurable9,  /* bit 25 */
    MEIMotorIoMask10 =  MEIXmpMotorIOMaskConfigurable10, /* bit 26 */
    MEIMotorIoMask11 =  MEIXmpMotorIOMaskConfigurable11, /* bit 27 */
    MEIMotorIoMask12 =  MEIXmpMotorIOMaskConfigurable12, /* bit 28 */
    MEIMotorIoMask13 =  MEIXmpMotorIOMaskConfigurable13, /* bit 29 */
    MEIMotorIoMask14 =  MEIXmpMotorIOMaskConfigurable14, /* bit 30 */
    MEIMotorIoMask15 =  MEIXmpMotorIOMaskConfigurable15, /* bit 31 */
} MEIMotorIoMask;
```

Description

MEIMotorIoMask is an enumeration of bit masks for the motor's configurable I/O. The support for configurable I/O is node/drive specific. See the node/drive manufacturer's documentation for details.

MEIMotorIoMask0	motor I/O mask for bit number 0
MEIMotorIoMask1	motor I/O mask for bit number 1
MEIMotorIoMask2	motor I/O mask for bit number 2
MEIMotorIoMask3	motor I/O mask for bit number 3
MEIMotorIoMask4	motor I/O mask for bit number 4
MEIMotorIoMask5	motor I/O mask for bit number 5
MEIMotorIoMask6	motor I/O mask for bit number 6
MEIMotorIoMask7	motor I/O mask for bit number 7
MEIMotorIoMask8	motor I/O mask for bit number 8
MEIMotorIoMask9	motor I/O mask for bit number 9
MEIMotorIoMask10	motor I/O mask for bit number 10
MEIMotorIoMask11	motor I/O mask for bit number 11

MEIMotorIoMask12	motor I/O mask for bit number 12
MEIMotorIoMask13	motor I/O mask for bit number 13
MEIMotorIoMask14	motor I/O mask for bit number 14
MEIMotorIoMask15	motor I/O mask for bit number 15

See Also

mpiMotorIoGet | mpiMotorIoSet | MEIMotorIoType

MPIMotorIo (Deprecated)

NOTE:
MPIMotorIo was moved to meiDeprecated.h in the 03.02.00 MPI software release.

Definition

```
typedef struct MPIMotorIo {
    unsigned long       input;
    unsigned long       output;
} MPIMotorIo;
```

Description

MPIMotorIo is a structure used to read the Motor input and set Motor output values.

input	The **input** value reflects the 32 bits of general and dedicated input values. The dedicated input values (bits 0-15) are specified by MEIMotorDedicatedIn. The general purpose input values (bits 16-31) can be specified by MEIMotorIoMask.
output	The **output** value reflects the 32 bits of general and dedicated output values. The dedicated output values (bits 0-15) are specified by MEIMotorDedicatedOut. The general purpose output values (bits 16-31) can be specified by MEIMotorIoMask.

See Also

MEIMotorDedicatedIn | MEIMotorDedicatedOut | MEIMotorIoMask

MEIFilterDataType (Deprecated)

NOTE:
MEIFilterDataType was moved to meiDeprecated.h in the 20030516 MPI software release.

Definition

```
/*
 * MEIFilterDataType changed to simply MEIDataType
 * so that other MPI methods can use the data type enum
 */
#define MEIFilterDataTypeINVALID        (MEIDataTypeINVALID)
#define MEIFilterDataTypeCHAR           (MEIDataTypeCHAR)
#define MEIFilterDataTypeSHORT          (MEIDataTypeSHORT)
#define MEIFilterDataTypeUSHORT         (MEIDataTypeUSHORT)
#define MEIFilterDataTypeLONG           (MEIDataTypeLONG)
#define MEIFilterDataTypeULONG          (MEIDataTypeULONG)
#define MEIFilterDataTypeFLOAT          (MEIDataTypeFLOAT)
#define MEIFilterDataTypeDOUBLE         (MEIDataTypeDOUBLE)
#define MEIFilterDataTypeLAST           (MEIDataTypeLAST)
#define MEIFilterDataTypeFIRST          (MEIDataTypeFIRST)
```

Description

MEIFilterDataType is an enumeration of data types for the filter coefficients.

MEIFilterDataTypeCHAR	character filter data type
MEIFilterDataTypeSHORT	short integer filter data type
MEIFilterDataTypeUSHORT	unsigned short integer filter data type
MEIFilterDataTypeLONG	long integer filter data type
MEIFilterDataTypeULONG	unsigned long filter data type
MEIFilterDataTypeFLOAT	floating point filter data type
MEIFilterDataTypeDOUBLE	double precision floating point filter data type

See Also

MEIDataType

MEISynqNetMessage (Deprecated)

NOTE:
MEISynqNetMessage was moved to meiDeprecated.h in the 20030715 MPI software release.

Definition

```
#define MEISynqNetMessageNODE_UNAVAILABLE (MEISqNodeMessageNODE_INVALID)
#define MEISynqNetMessageRESPONSE_TIMEOUT (MEISqNodeMessageRESPONSE_TIMEOUT)
#define MEISynqNetMessageREADY_TIMEOUT    (MEISqNodeMessageREADY_TIMEOUT)
#define MEISynqNetMessageSRVC_ERROR       (MEISqNodeMessageSRVC_ERROR)
#define MEISynqNetMessageSRVC_UNSUPPORTED (MEISqNodeMessageSRVC_UNSUPPORTED)
```

Description

MEISynqNetMessage is an enumeration of SynqNet error messages that can be returned by the MPI library.

MEISynqNetMessageNODE_UNAVAILABLE	The node number is not available on the network. This message code is returned by MPI methods that fail a service command transaction due to the specified node number is greater than or equal to the total number of nodes discovered during network initialization. To correct this problem, check the discovered node count with meiSynqNetInfo(…). If the node count is not what you expected check your network wiring, node condition, and re-initialize the network with mpiControlReset(…).
MEISynqNetMessageRESPONSE_TIMEOUT	The node failed to respond to a service command within the timeout. This message code is returned by MPI methods that fail a service command transaction because the node failed to respond within the allotted amount of time. To correct this problem, check your node hardware. There are 32 possible message codes for this error. Each message code specifies a different node, from node number 0 to 31.
MEISynqNetMessageREADY_TIMEOUT	The node failed to be ready for a service command within the timeout. This message code is returned by MPI methods that fail a service command transaction because the node is not ready to accept service commands. To correct this problem, check your node hardware. There are 32 possible message codes for this error. Each message code specifies a different node, from node number 0 to 31.
MEISynqNetMessageSRVC_ERROR	The service command failed. This message code is returned by MPI methods that fail a service command transaction. To correct this problem, check your node hardware. There are 32 possible message codes for this error. Each message code specifies a different node, from node number 0 to 31.
MEISynqNetMessageSRVC_UNSUPPORTED	The node does not support the service command. This message code is returned by MPI methods that fail a service command transaction because the node does not support it.

See Also

meiSynqNetInfo | mpiControlReset

MEISynqNetMaxMotorENCODER_COUNT (Deprecated)

NOTE:
MEISynqNetMaxMotorENCODER_COUNT was moved to meiDeprecated.h in the 20030722 MPI software release.

Definition

```
#define MEISynqNetMaxMotorENCODER_COUNT (MEIXmpMotorEncoders) /*  2 */
```

Description

MEISynqNetMaxMotorENCODER_COUNT defines the maximum number of encoder resources per motor.

NOTE: The encoder count may be further limited by the available resources on the node.

This define should be used instead of the MEIXmpMotorEncoders definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h.

See Also

MEIMotorDedicatedInOVERTRAVEL_POS (Deprecated)

NOTE:
MEIMotorDedicatedInOVERTRAVEL_POS was moved to meiDeprecated.h in the 20030421 MPI software release.

Definition

```
/* Backwards compatible dedicated I/O defines */
#define MEIMotorDedicatedInOVERTRAVEL_POS MEIMotorDedicatedInLIMIT_HW_POS
```

Description

MEIMotorDedicatedInOVERTRAVEL_POS is equivalent to MPIMotorDedicatedInLIMIT_HW_POS.

See MPIMotorDedicatedIn for a description.

See Also

MPIMotorDedicatedIn

MEIMotorDedicatedInOVERTRAVEL_NEG (Deprecated)

NOTE:
MEIMotorDedicatedInOVERTRAVEL_NEG was moved to meiDeprecated.h in the 20030421 MPI software release.

Definition

```
/* Backwards compatible dedicated I/O defines */
#define MEIMotorDedicatedInOVERTRAVEL_NEG MEIMotorDedicatedInLIMIT_HW_NEG
```

Description

MEIMotorDedicatedInOVERTRAVEL_NEG is equivalent to MPIMotorDedicatedInLIMIT_HW_NEG.

See MPIMotorDedicatedIn for a description.

See Also

MPIMotorDedicatedIn

Table of Error Messages / Return Values

The table below provides an exhaustive list of the error messages / return values that are returned by the message.exe utility. For a complete description of the error message and tips on troubleshooting, click on the links under the Define column.

String (sort alphabetically by String)	**Define** (sort alphabetically by Define)
OK	MPIMessageOK
Argument invalid	MPIMessageARG_INVALID
Parameter invalid	MPIMessagePARAM_INVALID
Handle invalid	MPIMessageHANDLE_INVALID
Out of memory	MPIMessageNO_MEMORY
Object freed	MPIMessageOBJECT_FREED
Object not enabled	MPIMessageOBJECT_NOT_ENABLED
Object not found	MPIMessageOBJECT_NOT_FOUND
Object on list	MPIMessageOBJECT_ON_LIST
Object in use	MPIMessageOBJECT_IN_USE
Timeout	MPIMessageTIMEOUT
Unsupported	MPIMessageUNSUPPORTED
Fatal error	MPIMessageFATAL_ERROR
File close error	MPIMessageFILE_CLOSE_ERROR
File open error	MPIMessageFILE_OPEN_ERROR
File read error	MPIMessageFILE_READ_ERROR
File write error	MPIMessageFILE_WRITE_ERROR
Firmware File Mismatch	MPIMessageFILE_MISMATCH
Axis	MPIAxisMessageFIRST
Axis: axis invalid	MPIAxisMessageAXIS_INVALID
Axis: unable to set command position	MPIAxisMessageCOMMAND_NOT_SET

Axis: not mapped to motion supervisor	MPIAxisMessageNOT_MAPPED_TO_MS
Can	MEICanMessageFIRST
Can: can invalid	MEICanMessageCAN_INVALID
Can: firmware invalid	MEICanMessageFIRMWARE_INVALID
Can: firmware version mismatch	MEICanMessageFIRMWARE_VERSION
Can: firmware not initialized	MEICanMessageNOT_INITIALIZED
Can: IO not supported by node	MEICanMessageIO_NOT_SUPPORTED
Can: file format incorrect	MEICanMessageFILE_FORMAT_ERROR
Can: user aborted operation	MEICanMessageUSER_ABORT
Can: DPR Command protocol	MEICanMessageCOMMAND_PROTOCOL
Can: interface not found	MEICanMessageINTERFACE_NOT_FOUND
Can: node dead	MEICanMessageNODE_DEAD
Can: SDO Timeout	MEICanMessageSDO_TIMEOUT
Can: SDO abort	MEICanMessageSDO_ABORT
Can: SDO protocol	MEICanMessageSDO_PROTOCOL
Can: transmit buffer overflow	MEICanMessageTX_OVERFLOW
Can: RTR transmit buffer overflow	MEICanMessageRTR_TX_OVERFLOW
Can: Receive buffer empty	MEICanMessageRX_BUFFER_EMPTY
Can: Bus off	MEICanMessageBUS_OFF
Can: Signature invalid	MEICanMessageSIGNATURE_INVALID
Capture	MEICaptureMessageFIRST
Capture: invalid motor number	MPICaptureMessageMOTOR_INVALID
Capture: invalid capture type	MPICaptureMessageCAPTURE_TYPE_INVALID
Capture: invalid capture number	MPICaptureMessageCAPTURE_INVALID
Capture: invalid encoder index parameter	MPICaptureMessageENCODER_INVALID
Capture: vaild capture edge required	MEICaptureMessageINVALID_EDGE

Capture: global capture cannot be enabled and be a trigger source at the same time	MEICaptureMessageGLOBAL_CONFIG_ERR
Capture: global config already enabled on another capture on this block	MEICaptureMessageGLOBAL_ALREADY_ENABLED
Capture: capture not enabled	MEICaptureMessageCAPTURE_NOT_ENABLED
Capture: capture is in an invalid state	MEICaptureMessageCAPTURE_STATE_INVALID
Capture: capture does not map to existing hardware	MEICaptureMessageNOT_MAPPED
Capture: capture not supported on primary encoder	MEICaptureMessageUNSUPPORTED_PRIMARY
Capture: capture not supported on secondary encoder	MEICaptureMessageUNSUPPORTED_SECONDARY
Capture: secondary index can not be used for multiple source capturing	MEICaptureMessageSECONDARY_INDEX_INVALID
Capture: capture currently armed	MEICaptureMessageCAPTURE_ARMED
Client	MEIClientMessageFIRST
Client: client invalid	MEIClientMessageCLIENT_INVALID
Client: method invalid	MEIClientMessageMETHOD_INVALID
Client: header invalid	MEIClientMessageHEADER_INVALID
Command	MPICommandMessageFIRST
Command: command invalid	MEICommandMessageCOMMAND_INVALID
Command: type invalid	MPICommandMessageTYPE_INVALID
Command: param invalid	MPICommandMessagePARAM_INVALID
Compensator	MPICompensatorMessageFIRST
Compensator: compensator object invalid	MPICompensatorMessageCOMPENSATOR_INVALID
Compensator: compensator object not configured	MPICompensatorMessageNOT_CONFIGURED
Compensator: object not enabled	MPICompensatorMessageNOT_ENABLED
Compensator: axis not enabled	MPICompensatorMessageAXIS_NOT_ENABLED

Compensator: table size is too small. adjust MPIControlConfig. compensatorPostionCount	MPICompensatorMessageTABLE_SIZE_ERROR
Compensator: position delta invalid	MPICompensatorMessagePOSITION_DELTA_INVALID
Compensator: table dimension not supported	MPICompensatorMessageDIMENSION_NOT_SUPPORTED
Control	MPIControlMessageFIRST
Control: application is not compatible with MPI DLL	MPIControlMessageLIBRARY_VERSION
Control: address invalid	MPIControlMessageADDRESS_INVALID
Control: control invalid	MPIControlMessageCONTROL_INVALID
Control: control number exceeds maximum limit	MPIControlMessageCONTROL_NUMBER_INVALID
Control: type invalid	MPIControlMessageTYPE_INVALID
Control: interrupts disabled	MPIControlMessageINTERRUPTS_DISABLED
Control: out of external memory	MPIControlMessageEXTERNAL_MEMORY_OVERFLOW
Control: adcCount exceeds configuration limit	MPIControlMessageADC_COUNT_INVALID
Control: axisCount exceeds configuration limit	MPIControlMessageAXIS_COUNT_INVALID
Control: invalid axisFrameCount	MPIControlMessageAXIS_FRAME_COUNT_INVALID
Control: captureCount exceeds configuration limit	MPIControlMessageCAPTURE_COUNT_INVALID
Control: compareCount exceeds configuration limit	MPIControlMessageCOMPARE_COUNT_INVALID
Control: cmdDacCount exceeds configuration limit	MPIControlMessageCMDDAC_COUNT_INVALID
Control: auxDacCount exceeds configuration limit	MPIControlMessageAUXDAC_COUNT_INVALID
Control: filterCount exceeds configuration limit	MPIControlMessageFILTER_COUNT_INVALID
Control: motionCount exceeds configuration limit	MPIControlMessageMOTION_COUNT_INVALID
Control: motorCount exceeds configuration limit	MPIControlMessageMOTOR_COUNT_INVALID

Control: sample rate value must be greater than or equal to 1000	MPIControlMessageSAMPLE_RATE_TO_LOW
Control: sample rate value must be less than or equal to 100000	MPIControlMessageSAMPLE_RATE_TO_HIGH
Control: recorderCount exceeds configuration limit	MPIControlMessageRECORDER_COUNT_INVALID
Control: compensatorCount exceeds configuration limit	MPIControlMessageCOMPENSATOR_COUNT_INVALID
Control: cannot configure while axes are running	MPIControlMessageAXIS_RUNNING
Control: cannot configure while recorders are running	MPIControlMessageRECORDER_RUNNING
Control: Application compiled with invalid packing alignment	MPIControlMessagePACK_ALIGNMENT
Control: firmware invalid	MEIControlMessageFIRMWARE_INVALID
Control: No firmware found (factory default)	MEIControlMessageFIRMWARE_VERSION_NONE
Control: firmware version mismatch	MEIControlMessageFIRMWARE_VERSION
Control: Too many FPGA hardware socket types	MEIControlMessageFPGA_SOCKETS
Control: Bad FPGA hardware socket data	MEIControlMessageBAD_FPGA_SOCKET_DATA
Control: FPGA hardware socket does not exist	MEIControlMessageNO_FPGA_SOCKET
Control: Invalid Motion Block count	MEIControlMessageINVALID_BLOCK_COUNT
Control: Too many SynqNet objects	MEIControlMessageSYNQNET_OBJECTS
Control: SynqNet state is invalid	MEIControlMessageSYNQNET_STATE
Control: I/O bit selected is unavailable	MEIControlMessageIO_BIT_INVALID
driveMap	MEIDriveMapMessageFIRST
driveMap: Could not open drive map file	MEIDriveMapMessageMAP_FILE_OPEN_ERROR
driveMap: Format error in drive map file	MEIDriveMapMessageMAP_FILE_FORMAT_INVALID

driveMap: Node type not found in drives map files	MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP
driveMap: Drive firmware version not found in drives map file (see drive vendor for .dm file update)	MEIDriveMapMessageVERSION_NOT_FOUND_IN_MAP
driveMap: Drive parameter is read-only	MEIDriveMapMessageDRIVE_PARAM_READ_ONLY
Element	MEIElementMessageFIRST
Element: element invalid	MEIElementMessageELEMENT_INVALID
Element: Velocity must be positive	MPIPathMessageILLEGAL_VELOCITY
Element: Acceleration must be positive	MPIPathMessageILLEGAL_ACCELERATION
Element: timeSlice must be positive	MPIPathMessageILLEGAL_TIMESLICE
Element: blending cannot be used with path interpolation method	MPIPathMessageINVALID_BLENDING
Event	MPIEventMessageFIRST
Event: event invalid	MPIEventMessageEVENT_INVALID
EventMgr	MPIEventMgrMessageFIRST
EventMgr: eventMgr invalid	MPIEventMgrMessageEVENTMGR_INVALID
Filter	MPIFilterMessageFIRST
Filter: filter invalid	MPIFilterMessageFILTER_INVALID
Filter: filter algorithm invalid	MPIFilterMessageINVALID_ALGORITHM
Filter: DRate value out of range (0-7)	MPIFilterMessageINVALID_DRATE
Filter: Divide by zero in conversion	MPIFilterMessageCONVERSION_DIV_BY_0
Filter: Specified postfilter section not enabled	MPIFilterMessageSECTION_NOT_ENABLED
Filter: Invalid filter form	MPIFilterMessageINVALID_FILTER_FORM
Filter: KA1 value out of range (0-.999)	MPIFilterMessageINVALID_KA1
Flash	MEIFlashMessageFIRST

Flash: flash invalid	MEIFlashMessageFLASH_INVALID
Flash: flash verify error	MEIFlashMessageFLASH_VERIFY_ERROR
Flash: flash write error	MEIFlashMessageFLASH_WRITE_ERROR
Flash: flash file path is too long	MEIFlashMessagePATH
Flash: write to flash failed. Network topology has not been saved to flash - use meiSynqNetTopologySave()	MEIFlashMessageNETWORK_TOPOLOGY_ERROR
List	MEIListMessageFIRST
List: list invalid	MEIListMessageLIST_INVALID
List: element not found	MEIListMessageELEMENT_NOT_FOUND
List: element invalid	MEIListMessageELEMENT_INVALID
Map	MEIMapMessageFIRST
Map: name invalid	MEIMapMessageNAME_INVALID
Map: name not found	MEIMapMessageNAME_NOT_FOUND
Map: index invalid	MEIMapMessageINDEX_INVALID
Map: file invalid	MEIMapMessageFILE_INVALID
Motion	MPIMotionMessageFIRST
Motion: motion invalid	MPIMotionMessageMOTION_INVALID
Motion: axis not found	MPIMotionMessageAXIS_NOT_FOUND
Motion: axis count invalid	MPIMotionMessageAXIS_COUNT
Motion: axis frame count invalid	MPIMotionMessageAXIS_FRAME_COUNT
Motion: type invalid	MPIMotionMessageTYPE_INVALID
Motion: attribute invalid	MPIMotionMessageATTRIBUTE_INVALID
Motion: MPIStateIDLE	MPIMotionMessageIDLE
Motion: MPIStateMOVING	MPIMotionMessageMOVING
Motion: MPIStateSTOPPING	MPIMotionMessageSTOPPING
Motion: MPIStateSTOPPED	MPIMotionMessageSTOPPED
Motion: MPIStateSTOPPING_ERROR	MPIMotionMessageSTOPPING_ERROR

Motion: MPIStateERROR	MPIMotionMessageERROR
Motion: auto-start	MPIMotionMessageAUTO_START
Motion: illegal delay	MPIMotionMessageILLEGAL_DELAY
Motion: profile error	MPIMotionMessagePROFILE_ERROR
Motion: profile not supported	MPIMotionMessagePROFILE_ERROR_NOT_SUPPORTED
Motion: path error	MPIMotionMessagePATH_ERROR
Motion: frame buffer low	MPIMotionMessageFRAMES_LOW
Motion: frame buffer empty	MPIMotionMessageFRAMES_EMPTY
Motion: frame buffer too small	MPIMotionMessageFRAME_BUFFER_TOO_SMALL
Motion: RESERVED0	MEIMotionMessageRESERVED0
Motion: RESERVED1	MEIMotionMessageRESERVED1
Motion: RESERVED2	MEIMotionMessageRESERVED2
Motion: No Axis mapped	MEIMotionMessageNO_AXES_MAPPED
Motion: Bad Path Data	MEIMotionMessageBAD_PATH_DATA
Motor	MPIMotorMessageFIRST
Motor: motor invalid	MPIMotorMessageMOTOR_INVALID
Motor: motor type invalid	MPIMotorMessageTYPE_INVALID
Motor: motor not enabled	MEIMotorMessageMOTOR_NOT_ENABLED
Motor: secondary encoder not available	MEIMotorMessageSECONDARY_ENCODER_NA
Motor: hardware not found	MEIMotorMessageHARDWARE_NOT_FOUND
Motor: cannot set disable action to CMD_EQ_ACT when motor type is STEPPER	MEIMotorMessageSTEPPER_INVALID
Motor: cannot set motor type to STEPPER when disable action is CMD_EQ_ACT	MEIMotorMessageDISABLE_ACTION_INVALID
Motor: stepper Pulse Width out of range (.00000006 < pulseWidth < .00100006)	MEIMotorMessagePULSE_WIDTH_INVALID
Motor: unable to invert feedback for specified encoder type	MEIMotorMessageFEEDBACK_REVERSAL_NA

Motor: unable to disable the filter for specified encoder type	MEIMotorMessageFILTER_DISABLE_NA
Motor: Motor phase finding failure	MEIMotorMessagePHASE_FINDING_FAILED
Motor: specified demand mode not supported by drive	MEIMotorMessageDEMAND_MODE_UNSUPPORTED
Motor: unable to switch demand mode while the amplifier is enabled	MEIMotorMessageDEMAND_MODE_NOT_SET
Notify	MPINotifyMessageFIRST
Notify: notify invalid	MPINotifyMessageNOTIFY_INVALID
Notify: wait in progress	MPINotifyMessageWAIT_IN_PROGRESS
Packet	MEIPacketMessageFIRST
Packet: packet invalid	MEIPacketMessagePACKET_INVALID
Packet: address invalid	MEIPacketMessageADDRESS_INVALID
Packet: communication error	MEIPacketMessageCOMM_ERROR
Packet: connection closed	MEIPacketMessageCONNECTION_CLOSED
Packet: receive error	MEIPacketMessageRECV_ERROR
Packet: send error	MEIPacketMessageSEND_ERROR
Path	MPIPathMessageFIRST
Path: Path invalid	MPIPathMessagePATH_INVALID
Path: Dimension Out of Range	MPIPathMessageILLEGAL_DIMENSION
Path: Illegal Element	MPIPathMessageILLEGAL_ELEMENT
Path: Arc Dimension Out of Range	MPIPathMessageARC_ILLEGAL_DIMENSION
Path: Helix Dimension Out of Range	MPIPathMessageHELIX_ILLEGAL_DIMENSION
Path: Illegal Radius	MPIPathMessageILLEGAL_RADIUS
Path: Path too long	MPIPathMessagePATH_TOO_LONG
Probe	MPIProbeMessageFIRST
Probe: invalid node number	MPIProbeMessageNODE_INVALID
Probe: invalid probe type	MPIProbeMessagePROBE_TYPE_INVALID

Probe: invalid probe number	MPIProbeMessagePROBE_INVALID
Recorder	MPIRecorderMessageFIRST
Recorder: recorder invalid	MPIRecorderMessageRECORDER_INVALID
Recorder: already started	MPIRecorderMessageSTARTED
Recorder: already stopped	MPIRecorderMessageSTOPPED
Recorder: not configured	MPIRecorderMessageNOT_CONFIGURED
Recorder: no recorders available	MPIRecorderMessageNO_RECORDERS_AVAIL
Recorder: not enabled	MPIRecorderMessageNOT_ENABLED
Recorder: cannot configure while running	MPIRecorderMessageRUNNING
Sequence	MPISequenceMessageFIRST
Sequence: sequence invalid	MPISequenceMessageSEQUENCE_INVALID
Sequence: command count invalid	MPISequenceMessageCOMMAND_COUNT
Sequence: command not found	MPISequenceMessageCOMMAND_NOT_FOUND
Sequence: MPISequenceStateSTARTED	MPISequenceMessageSTARTED
Sequence: MPISequenceStateSTOPPED	MPISequenceMessageSTOPPED
Server	MEIServerMessageFIRST
Server: server invalid	MEIServerMessageSERVER_INVALID
Server: method invalid	MEIServerMessageMETHOD_INVALID
Server: header invalid	MEIServerMessageHEADER_INVALID
SqNode	MEISqNodeMessageFIRST
SqNode: invalid	MEISqNodeMessageINVALID
SqNode: Node invalid	MEISqNodeMessageNODE_INVALID
SqNode: network state error	MEISqNodeMessageSTATE_ERROR
SqNode: Config file and network are different	MEISqNodeMessageCONFIG_NETWORK_MISMATCH
SqNode: Map and config files are different	MEISqNodeMessageMAP_CONFIG_MISMATCH

SqNode: Not in Config File	MEISqNodeMessageNOT_IN_CONFIG_FILE
SqNode: Config file format invalid	MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID
SqNode: service cmd response timeout	MEISqNodeMessageRESPONSE_TIMEOUT
SqNode: node busy : service cmd ready timeout	MEISqNodeMessageREADY_TIMEOUT
SqNode: service cmd error	MEISqNodeMessageSRVC_ERROR
SqNode: service cmd unsupported	MEISqNodeMessageSRVC_UNSUPPORTED
SqNode: invalid service channel specified	MEISqNodeMessageSRVC_CHANNEL_INVALID
SqNode: node module did not complete the specified operation	MEISqNodeMessageCMD_NOT_SUPPORTED
SqNode: node specific discovery failure	MEISqNodeMessageDISCOVERY_FAILURE
SqNode: node specific command dispatch error	MEISqNodeMessageDISPATCH_ERROR
SqNode: node specific initialization failure	MEISqNodeMessageINIT_FAILURE
SqNode: node module doesn't support discovery	MEISqNodeMessageINTERFACE_ERROR1
SqNode: node type does not match the file provided for download	MEISqNodeMessageFILE_NODE_MISMATCH
SqNode: the file provided for download was not found	MEISqNodeMessageFILE_INVALID
SqNode: the header information in the download image is invalid	MEISqNodeMessageINVALID_HEADER
SqNode: node firmware download failed	MEISqNodeMessageDOWNLOAD_FAIL
SqNode: node firmware verify failed	MEISqNodeMessageVERIFY_FAIL
SqNode: firmware download not supported	MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED
SqNode: firmware verify not supported	MEISqNodeMessageVERIFY_NOT_SUPPORTED
SqNode: boot rom not recognized	MEISqNodeMessageBOOT_ROM_INVALID

SqNode : invalid resource table in node module	MEISqNodeMessageINVALID_TABLE
SqNode : invalid string length	MEISqNodeMessageINVALID_STR_LEN
SqNode: invalid feedback map attempt	MEISqNodeMessageFEEDBACK_MAP_INVAILD
SqNode: node failure	MEISqNodeMessageNODE_FAILURE
SqNode: exceeded maximum synqnet packet limit	MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT
SqNode: I/O Module Incompatibility	MEISqNodeMessageIO_MODULE_INCOMPATIBILITY
SqNode: I/O Module EEPROM not programmed	MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED
SqNode: I/O Module Count Exceeded	MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED
SqNode: I/O Module length check failed	MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED
SqNode: I/O Module 3.3V bus current exceeded	MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED
SqNode: I/O Module 24V bus current exceeded	MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED
SqNode: I/O Slice initalization fault	MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT
SqNode: I/O Slice initalization fault too many slices	MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MANY_SLICES
SqNode: I/O Slice initalization fault vendor mismatch	MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH
SqNode: I/O Slice initalization timeout	MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT
SqNode: I/O Slice topology mismatch	MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH
SqNode: I/O Slice service receive error	MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR
SqNode: I/O Slice service too many char	MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR
SqNode: I/O Slice service bus error code	MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE
SqNode: I/O Slice service unknown fault code	MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE

SqNode: I/O Slice service resource unavailable	MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILBLE
SqNode: I/O Slice service not supported	MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED
SqNode: I/O Slice service invalid attribute value	MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE
SqNode: I/O Slice service already in mode	MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE
SqNode: I/O Slice service state conflict	MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT
SqNode: I/O Slice service attribute not settable	MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE
SqNode: I/O Slice service not enough data	MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA
SqNode: I/O Slice service attribute not supported	MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED
SqNode: I/O Slice service too much data	MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA
SqNode: I/O Slice service object does not exist	MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST
SqNode: I/O Slice service invalid parameter	MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER
SqNode: I/O Slice service store operation failure	MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE
SqNode: I/O Slice service unknown error code	MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE
SqNode: I/O Slice service timeout	MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT
SqNode: I/O Slice service response format	MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT
SqNode: I/O Slice eeprom format	MEISqNodeMessageIO_SLICE_SERVICE_EEPROM_FORMAT
SqNode: I/O Slice too much IO	MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_IO
SqNode: Boot file not found or corrupt, kollmorgen_ember.a00 must be in path	MEISqNodeMessageBOOT_FILE_NOT_FOUND
SqNode: Parameter is read only	MEISqNodeMessagePARAM_READ_ONLY
SqNode: SFD motor selected, parameter is locked	MEISqNodeMessagePARAM_LOCKED

SqNode: Monitor config invalid, index not supported	MEISqNodeMessageMONITOR_INDEX
SqNode: Monitor config invalid, address not supported	MEISqNodeMessageMONITOR_ADDRESS
SynqNet	MEISynqNetMessageFIRST
SynqNet: synqNet invalid	MEISynqNetMessageSYNQNET_INVALID
SynqNet: maximum blocks exceeded	MEISynqNetMessageMAX_NODE_ERROR
SynqNet: unexpected network state	MEISynqNetMessageSTATE_ERROR
SynqNet: network communication is down	MEISynqNetMessageCOMM_ERROR
SynqNet: network down due to CRC errors	MEISynqNetMessageCOMM_ERROR_CRC
SynqNet: network down due to Rx errors	MEISynqNetMessageCOMM_ERROR_RX
SynqNet: network down due to Rx packet length errors	MEISynqNetMessageCOMM_ERROR_RX_LEN
SynqNet: network down due to Rx pack errors	MEISynqNetMessageCOMM_ERROR_RX_FIFO
SynqNet: network down due to Rx dribble	MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE
SynqNet: network down due to Rx CRC errors	MEISynqNetMessageCOMM_ERROR_RX_CRC
SynqNet: interface not available	MEISynqNetMessageINTERFACE_NOT_FOUND
SynqNet: network topology mismatch in dynamic memory - network in asynq mode	MEISynqNetMessageTOPOLOGY_MISMATCH
SynqNet: network topology mismatch in flash memory - network in asynq mode	MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH
SynqNet: timeout : reset request packet	MEISynqNetMessageRESET_REQ_TIMEOUT
SynqNet: timeout : reset complete packet	MEISynqNetMessageRESET_ACK_TIMEOUT
SynqNet: timeout : discovery problem	MEISynqNetMessageDISCOVERY_TIMEOUT
SynqNet: no nodes found on network	MEISynqNetMessageNO_NODES_FOUND

SynqNet: no timing data available in module	MEISynqNetMessageNO_TIMING_DATA_AVAIL
SynqNet: internal buffer size overflow: reduce network payload	MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW
SynqNet: too many motors configured for this node	MEISynqNetMessageINVALID_MOTOR_COUNT
SynqNet: invalid auxiliary encoder count	MEISynqNetMessageINVALID_AUX_ENC_COUNT
SynqNet: incomplete motor : fulfill motor packet requirements or remove all packet fields for motor	MEISynqNetMessageINCOMPLETE_MOTOR
SynqNet: invalid command configuration	MEISynqNetMessageINVALID_COMMAND_CFG
SynqNet: invalid pulse engine count	MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT
SynqNet: invalid feedback count	MEISynqNetMessageINVALID_ENCODER_COUNT
SynqNet: invalid capture count	MEISynqNetMessageINVALID_CAPTURE_COUNT
SynqNet: invalid compare count	MEISynqNetMessageINVALID_COMPARE_COUNT
SynqNet: invalid ioInput count	MEISynqNetMessageINVALID_INPUT_COUNT
SynqNet: invalid ioOutput count	MEISynqNetMessageINVALID_OUTPUT_COUNT
SynqNet: invalid monitor field configuration	MEISynqNetMessageINVALID_MONITOR_CFG
SynqNet: invalid analogIn count	MEISynqNetMessageINVALID_ANALOG_IN_COUNT
SynqNet: invalid digitalIn count	MEISynqNetMessageINVALID_DIGITAL_IN_COUNT
SynqNet: invalid digitalOut count	MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT
SynqNet: invalid analogOut count	MEISynqNetMessageINVALID_ANALOG_OUT_COUNT
SynqNet: cable number is not idle, status is unknown	MEISynqNetMessageLINK_NOT_IDLE
SynqNet: idle cable number unknown due to failed node(s)	MEISynqNetMessageIDLE_LINK_UNKNOWN
SynqNet: only supported with ring topologies	MEISynqNetMessageRING_ONLY
SynqNet: network is currently recovering from a fault	MEISynqNetMessageRECOVERING

SynqNet: detected an unsupported cable length	MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED
SynqNet: cable length measurement timeout	MEISynqNetMessageCABLE_LENGTH_TIMEOUT
SynqNet: cable length mismatch - network in asynq mode	MEISynqNetMessageCABLE_LENGTH_MISMATCH
SynqNet: nominal cable length out of range	MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL
SynqNet: minimum cable length out of range	MEISynqNetMessageCABLE_LENGTH_INVALID_MIN
SynqNet: maximum cable length out of range	MEISynqNetMessageCABLE_LENGTH_INVALID_MAX
SynqNet: node FPGA version mismatch warning	MEISynqNetMessageNODE_FPGA_VERSION
SynqNet: maximum motor count exceeded	MEISynqNetMessageMAX_MOTOR_ERROR
SynqNet: node PLL unable to lock with drive	MEISynqNetMessagePLL_ERROR
SynqNet: node specific initialization failed	MEISynqNetMessageNODE_INIT_FAIL
SynqNet: network topology is already clear	MEISynqNetMessageTOPOLOGY_CLEAR
SynqNet: network topology is already saved to flash. Use meiSynqNetFlashTopologyClear (...) first	MEISynqNetMessageTOPOLOGY_SAVED
SynqNet: network topology cannot be saved or cleared while amplifiers are enabled	MEISynqNetMessageTOPOLOGY_AMPS_ENABLED
SynqNet: node MAC version mismatch - network in asynq mode	MEISynqNetMessageNODE_MAC_VERSION
SynqNet: controller sample rate exceeds required analog input sample time	MEISynqNetMessageADC_SAMPLE_FAILURE
SynqNet: unrecoverable network scheduling error	MEISynqNetMessageSCHEDULING_ERROR
SynqNet: invalid probe count	MEISynqNetMessageINVALID_PROBE_CFG
SynqNet: invalid probe depth	MEISynqNetMessageINVALID_PROBE_DEPTH

SynqNet: controller sample rate must be a integer multiple of all node update periods	MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE
SynqNet: node control latency maximum exceeded	MEISynqNetMessageNODE_LATENCY_EXCEEDED
SynqNet Hot Restart: Not in Synq state	MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE
SynqNet Hot Restart: Recovering	MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING
SynqNet Hot Restart: Test Packet Active	MEISynqNetMessageHOT_RESTART_FAIL_TEST_PACKET
SynqNet Hot Restart: Node Address Assignment Failure	MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT
SynqNet Hot Restart: Not all nodes could be restarted	MEISynqNetMessageHOT_RESTART_NOT_ALL_NODES_RESTARTED
SynqNet: Nodes being shutdown need to be consecutive	MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE
SynqNet: Shutting down the nodes requested will result in stranding working drives on the system, check topology	MEISynqNetMessageSHUTDOWN_NODES_STRANDED
SynqNet: Shutting down nodes requires recovery to be enabled	MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED
WinNT	MEIPlatformMessageFIRST
WinNT: platform invalid	MEIPlatformMessagePLATFORM_INVALID
WinNT: device invalid	MEIPlatformMessageDEVICE_INVALID
WinNT: device error	MEIPlatformMessageDEVICE_ERROR
WinNT: device map error	MEIPlatformMessageDEVICE_MAP_ERROR
WinNT: copy64 failure	MEIPlatformMessageCOPY64_FAILURE

Table of Error Messages / Return Values

The table below provides an exhaustive list of the error messages / return values that are returned by the message.exe utility. For a complete description of the error message and tips on troubleshooting, click on the links under the Define column.

String (sort by message.exe output)	Define (sort alphabetically by Define)
Argument invalid	MPIMessageARG_INVALID
Axis	MPIAxisMessageFIRST
Axis: axis invalid	MPIAxisMessageAXIS_INVALID
Axis: not mapped to motion supervisor	MPIAxisMessageNOT_MAPPED_TO_MS
Axis: unable to set command position	MPIAxisMessageCOMMAND_NOT_SET
Can	MPICanMessageFIRST
Can: Bus off	MPICanMessageBUS_OFF
Can: can invalid	MPICanMessageCAN_INVALID
Can: DPR Command protocol	MPICanMessageCOMMAND_PROTOCOL
Can: file format incorrect	MPICanMessageFILE_FORMAT_ERROR
Can: firmware invalid	MPICanMessageFIRMWARE_INVALID
Can: firmware not initialized	MPICanMessageNOT_INITIALIZED
Can: firmware version mismatch	MPICanMessageFIRMWARE_VERSION
Can: IO not supported by node	MPICanMessageIO_NOT_SUPPORTED
Can: interface not found	MPICanMessageINTERFACE_NOT_FOUND
Can: node dead	MPICanMessageNODE_DEAD
Can: Receive buffer empty	MPICanMessageRX_BUFFER_EMPTY
Can: RTR transmit buffer overflow	MPICanMessageRTR_TX_OVERFLOW
Can: SDO abort	MPICanMessageSDO_ABORT
Can: SDO protocol	MPICanMessageSDO_PROTOCOL
Can: SDO Timeout	MPICanMessageSDO_TIMEOUT

Can: Signature invalid	MPICanMessageSIGNATURE_INVALID
Can: transmit buffer overflow	MPICanMessageTX_OVERFLOW
Can: user aborted operation	MPICanMessageUSER_ABORT
Capture	MEICaptureMessageFIRST
Capture: capture currently armed	MEICaptureMessageCAPTURE_ARMED
Capture: capture does not map to existing hardware	MEICaptureMessageNOT_MAPPED
Capture: capture is in an invalid state	MEICaptureMessageCAPTURE_STATE_INVALID
Capture: capture not enabled	MEICaptureMessageCAPTURE_NOT_ENABLED
Capture: capture not supported on primary encoder	MEICaptureMessageUNSUPPORTED_PRIMARY
Capture: capture not supported on secondary encoder	MEICaptureMessageUNSUPPORTED_SECONDARY
Capture: global capture cannot be enabled and be a trigger source at the same time	MEICaptureMessageGLOBAL_CONFIG_ERR
Capture: global config already enabled on another capture on this block	MEICaptureMessageGLOBAL_ALREADY_ENABLED
Capture: invalid capture number	MPICaptureMessageCAPTURE_INVALID
Capture: invalid capture type	MPICaptureMessageCAPTURE_TYPE_INVALID
Capture: invalid encoder index parameter	MPICaptureMessageENCODER_INVALID
Capture: invalid motor number	MPICaptureMessageMOTOR_INVALID
Capture: secondary index can not be used for multiple source capturing	MEICaptureMessageSECONDARY_INDEX_INVALID
Capture: vaild capture edge required	MEICaptureMessageINVALID_EDGE
Client	MEIClientMessageFIRST
Client: client invalid	MEIClientMessageCLIENT_INVALID
Client: header invalid	MEIClientMessageHEADER_INVALID
Client: method invalid	MEIClientMessageMETHOD_INVALID

Command	MPICommandMessageFIRST
Command: command invalid	MEICommandMessageCOMMAND_INVALID
Command: param invalid	MPICommandMessagePARAM_INVALID
Command: type invalid	MPICommandMessageTYPE_INVALID
Compensator	MPICompensatorMessageFIRST
Compensator: axis not enabled	MPICompensatorMessageAXIS_NOT_ENABLED
Compensator: compensator object invalid	MPICompensatorMessageCOMPENSATOR_INVALID
Compensator: compensator object not configured	MPICompensatorMessageNOT_CONFIGURED
Compensator: object not enabled	MPICompensatorMessageNOT_ENABLED
Compensator: position delta invalid	MPICompensatorMessagePOSITION_DELTA_INVALID
Compensator: table dimension not supported	MPICompensatorMessageDIMENSION_NOT_SUPPORTED
Compensator: table size is too small. adjust MPIControlConfig. compensatorPostionCount	MPICompensatorMessageTABLE_SIZE_ERROR
Control	MPIControlMessageFIRST
Control: address invalid	MPIControlMessageADDRESS_INVALID
Control: adcCount exceeds configuration limit	MPIControlMessageADC_COUNT_INVALID
Control: Application compiled with invalid packing alignment	MPIControlMessagePACK_ALIGNMENT
Control: application is not compatible with MPI DLL	MPIControlMessageLIBRARY_VERSION
Control: auxDacCount exceeds configuration limit	MPIControlMessageAUXDAC_COUNT_INVALID
Control: axisCount exceeds configuration limit	MPIControlMessageAXIS_COUNT_INVALID
Control: Bad FPGA hardware socket data	MEIControlMessageBAD_FPGA_SOCKET_DATA
Control: cannot configure while axes are running	MPIControlMessageAXIS_RUNNING

Control: cannot configure while recorders are running	MPIControlMessageRECORDER_RUNNING
Control: captureCount exceeds configuration limit	MPIControlMessageCAPTURE_COUNT_INVALID
Control: cmdDacCount exceeds configuration limit	MPIControlMessageCMDDAC_COUNT_INVALID
Control: compareCount exceeds configuration limit	MPIControlMessageCOMPARE_COUNT_INVALID
Control: compensatorCount exceeds configuration limit	MPIControlMessageCOMPENSATOR_COUNT_INVALID
Control: control invalid	MPIControlMessageCONTROL_INVALID
Control: control number exceeds maximum limit	MPIControlMessageCONTROL_NUMBER_INVALID
Control: filterCount exceeds configuration limit	MPIControlMessageFILTER_COUNT_INVALID
Control: firmware invalid	MEIControlMessageFIRMWARE_INVALID
Control: firmware version mismatch	MEIControlMessageFIRMWARE_VERSION
Control: FPGA hardware socket does not exist	MEIControlMessageNO_FPGA_SOCKET
Control: interrupts disabled	MPIControlMessageINTERRUPTS_DISABLED
Control: invalid axisFrameCount	MPIControlMessageAXIS_FRAME_COUNT_INVALID
Control: Invalid Motion Block count	MEIControlMessageINVALID_BLOCK_COUNT
Control: I/O bit selected is unavailable	MEIControlMessageIO_BIT_INVALID
Control: motionCount exceeds configuration limit	MPIControlMessageMOTION_COUNT_INVALID
Control: motorCount exceeds configuration limit	MPIControlMessageMOTOR_COUNT_INVALID
Control: No firmware found (factory default)	MEIControlMessageFIRMWARE_VERSION_NONE
Control: out of external memory	MPIControlMessageEXTERNAL_MEMORY_OVERFLOW
Control: recorderCount exceeds configuration limit	MPIControlMessageRECORDER_COUNT_INVALID
Control: sample rate value must be greater than or equal to 1000	MPIControlMessageSAMPLE_RATE_TO_LOW

Control: sample rate value must be less than or equal to 100000	MPIControlMessageSAMPLE_RATE_TO_HIGH
Control: SynqNet state is invalid	MEIControlMessageSYNQNET_STATE
Control: Too many FPGA hardware socket types	MEIControlMessageFPGA_SOCKETS
Control: Too many SynqNet objects	MEIControlMessageSYNQNET_OBJECTS
Control: type invalid	MPIControlMessageTYPE_INVALID
driveMap	MEIDriveMapMessageFIRST
driveMap: Could not open drive map file	MEIDriveMapMessageMAP_FILE_OPEN_ERROR
driveMap: Drive firmware version not found in drives map file (see drive vendor for .dm file update)	MEIDriveMapMessageVERSION_NOT_FOUND_IN_MAP
driveMap: Drive parameter is read-only	MEIDriveMapMessageDRIVE_PARAM_READ_ONLY
driveMap: Format error in drive map file	MEIDriveMapMessageMAP_FILE_FORMAT_INVALID
driveMap: Node type not found in drives map files	MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP
Element	MEIElementMessageFIRST
Element: Acceleration must be positive	MPIPathMessageILLEGAL_ACCELERATION
Element: blending cannot be used with path interpolation method	MPIPathMessageINVALID_BLENDING
Element: element invalid	MEIElementMessageELEMENT_INVALID
Element: timeSlice must be positive	MPIPathMessageILLEGAL_TIMESLICE
Element: Velocity must be positive	MPIPathMessageILLEGAL_VELOCITY
Event	MPIEventMessageFIRST
Event: event invalid	MPIEventMessageEVENT_INVALID
EventMgr	MPIEventMgrMessageFIRST
EventMgr: eventMgr invalid	MPIEventMgrMessageEVENTMGR_INVALID

Fatal error	MPIMessageFATAL_ERROR
File close error	MPIMessageFILE_CLOSE_ERROR
File open error	MPIMessageFILE_OPEN_ERROR
File read error	MPIMessageFILE_READ_ERROR
File write error	MPIMessageFILE_WRITE_ERROR
Filter	MPIFilterMessageFIRST
Filter: filter algorithm invalid	MPIFilterMessageINVALID_ALGORITHM
Filter: Divide by zero in conversion	MPIFilterMessageCONVERSION_DIV_BY_0
Filter: DRate value out of range (0-7)	MPIFilterMessageINVALID_DRATE
Filter: filter invalid	MPIFilterMessageFILTER_INVALID
Filter: Invalid filter form	MPIFilterMessageINVALID_FILTER_FORM
Filter: KA1 value out of range (0-.999)	MPIFilterMessageINVALID_KA1
Filter: Specified postfilter section not enabled	MPIFilterMessageSECTION_NOT_ENABLED
Firmware File Mismatch	MPIMessageFILE_MISMATCH
Flash	MEIFlashMessageFIRST
Flash: flash file path is too long	MEIFlashMessagePATH
Flash: flash invalid	MEIFlashMessageFLASH_INVALID
Flash: flash verify error	MEIFlashMessageFLASH_VERIFY_ERROR
Flash: flash write error	MEIFlashMessageFLASH_WRITE_ERROR
Flash: write to flash failed. Network topology has not been saved to flash - use meiSynqNetTopologySave()	MEIFlashMessageNETWORK_TOPOLOGY_ERROR
Handle invalid	MPIMessageHANDLE_INVALID
List	MEIListMessageFIRST
List: element invalid	MEIListMessageELEMENT_INVALID
List: element not found	MEIListMessageELEMENT_NOT_FOUND
List: list invalid	MEIListMessageLIST_INVALID

Map	MEIMapMessageFIRST
Map: file invalid	MEIMapMessageFILE_INVALID
Map: index invalid	MEIMapMessageINDEX_INVALID
Map: name invalid	MEIMapMessageNAME_INVALID
Map: name not found	MEIMapMessageNAME_NOT_FOUND
Motion	MPIMotionMessageFIRST
Motion: attribute invalid	MPIMotionMessageATTRIBUTE_INVALID
Motion: auto-start	MPIMotionMessageAUTO_START
Motion: axis count invalid	MPIMotionMessageAXIS_COUNT
Motion: axis frame count invalid	MPIMotionMessageAXIS_FRAME_COUNT
Motion: axis not found	MPIMotionMessageAXIS_NOT_FOUND
Motion: Bad Path Data	MEIMotionMessageBAD_PATH_DATA
Motion: frame buffer empty	MPIMotionMessageFRAMES_EMPTY
Motion: frame buffer low	MPIMotionMessageFRAMES_LOW
Motion: frame buffer too small	MPIMotionMessageFRAME_BUFFER_TOO_SMALL
Motion: illegal delay	MPIMotionMessageILLEGAL_DELAY
Motion: motion invalid	MPIMotionMessageMOTION_INVALID
Motion: MPIStateERROR	MPIMotionMessageERROR
Motion: MPIStateIDLE	MPIMotionMessageIDLE
Motion: MPIStateMOVING	MPIMotionMessageMOVING
Motion: MPIStateSTOPPING	MPIMotionMessageSTOPPING
Motion: MPIStateSTOPPED	MPIMotionMessageSTOPPED
Motion: MPIStateSTOPPING_ERROR	MPIMotionMessageSTOPPING_ERROR
Motion: No Axis mapped	MEIMotionMessageNO_AXES_MAPPED
Motion: path error	MPIMotionMessagePATH_ERROR
Motion: profile error	MPIMotionMessagePROFILE_ERROR
Motion: profile not supported	MPIMotionMessagePROFILE_ERROR_NOT_SUPPORTED
Motion: RESERVED0	MEIMotionMessageRESERVED0

Motion: RESERVED1	MEIMotionMessageRESERVED1
Motion: RESERVED2	MEIMotionMessageRESERVED2
Motion: type invalid	MPIMotionMessageTYPE_INVALID
Motor	MPIMotorMessageFIRST
Motor: cannot set disable action to CMD_EQ_ACT when motor type is STEPPER	MEIMotorMessageSTEPPER_INVALID
Motor: cannot set motor type to STEPPER when disable action is CMD_EQ_ACT	MEIMotorMessageDISABLE_ACTION_INVALID
Motor: hardware not found	MEIMotorMessageHARDWARE_NOT_FOUND
Motor: motor invalid	MPIMotorMessageMOTOR_INVALID
Motor: motor not enabled	MEIMotorMessageMOTOR_NOT_ENABLED
Motor: Motor phase finding failure	MEIMotorMessagePHASE_FINDING_FAILED
Motor: motor type invalid	MPIMotorMessageMOTOR_TYPE_INVALID
Motor: secondary encoder not available	MEIMotorMessageSECONDARY_ENCODER_NA
Motor: stepper Pulse Width out of range (.00000006 < pulseWidth < .00100006)	MEIMotorMessagePULSE_WIDTH_INVALID
Motor: unable to disable the filter for specified encoder type	MEIMotorMessageFILTER_DISABLE_NA
Motor: unable to invert feedback for specified encoder type	MEIMotorMessageFEEDBACK_REVERSAL_NA
Motor: specified demand mode not supported by drive	MEIMotorMessageDEMAND_MODE_UNSUPPORTED
Motor: unable to switch demand mode while the amplifier is enabled	MEIMotorMessageDEMAND_MODE_NOT_SET
Notify	MPINotifyMessageFIRST
Notify: notify invalid	MPINotifyMessageNOTIFY_INVALID
Notify: wait in progress	MPINotifyMessageWAIT_IN_PROGRESS
Object freed	MPIMessageOBJECT_FREED
Object in use	MPIMessageOBJECT_IN_USE

Object not enabled	MPIMessageOBJECT_NOT_ENABLED
Object not found	MPIMessageOBJECT_NOT_FOUND
Object on list	MPIMessageOBJECT_ON_LIST
OK	MPIMessageOK
Out of memory	MPIMessageNO_MEMORY
Packet	MEIPacketMessageFIRST
Packet: address invalid	MEIPacketMessageADDRESS_INVALID
Packet: communication error	MEIPacketMessageCOMM_ERROR
Packet: connection closed	MEIPacketMessageCONNECTION_CLOSED
Packet: packet invalid	MEIPacketMessagePACKET_INVALID
Packet: receive error	MEIPacketMessageRECV_ERROR
Packet: send error	MEIPacketMessageSEND_ERROR
Parameter invalid	MPIMessagePARAM_INVALID
Path	MPIPathMessageFIRST
Path: Arc Dimension Out of Range	MPIPathMessageARC_ILLEGAL_DIMENSION
Path: Dimension Out of Range	MPIPathMessageILLEGAL_DIMENSION
Path: Helix Dimension Out of Range	MPIPathMessageHELIX_ILLEGAL_DIMENSION
Path: Illegal Element	MPIPathMessageILLEGAL_ELEMENT
Path: Illegal Radius	MPIPathMessageILLEGAL_RADIUS
Path: Path invalid	MPIPathMessagePATH_INVALID
Path: Path too long	MPIPathMessagePATH_TOO_LONG
Probe	MPIProbeMessageFIRST
Probe: invalid node number	MPIProbeMessageNODE_INVALID
Probe: invalid probe number	MPIProbeMessagePROBE_INVALID
Probe: invalid probe type	MPIProbeMessagePROBE_TYPE_INVALID
Recorder	MPIRecorderMessageFIRST

Recorder: already started	MPIRecorderMessageSTARTED
Recorder: already stopped	MPIRecorderMessageSTOPPED
Recorder: cannot configure while running	MPIRecorderMessageRUNNING
Recorder: no recorders available	MPIRecorderMessageNO_RECORDERS_AVAIL
Recorder: not configured	MPIRecorderMessageNOT_CONFIGURED
Recorder: not enabled	MPIRecorderMessageNOT_ENABLED
Recorder: recorder invalid	MPIRecorderMessageRECORDER_INVALID
Sequence	MPISequenceMessageFIRST
Sequence: command count invalid	MPISequenceMessageCOMMAND_COUNT
Sequence: command not found	MPISequenceMessageCOMMAND_NOT_FOUND
Sequence: MPISequenceStateSTARTED	MPISequenceMessageSTARTED
Sequence: MPISequenceStateSTOPPED	MPISequenceMessageSTOPPED
Sequence: sequence invalid	MPISequenceMessageSEQUENCE_INVALID
Server	MEIServerMessageFIRST
Server: header invalid	MEIServerMessageHEADER_INVALID
Server: method invalid	MEIServerMessageMETHOD_INVALID
Server: server invalid	MEIServerMessageSERVER_INVALID
SqNode	MEISqNodeMessageFIRST
SqNode: Boot file not found or corrupt, kollmorgen_ember.a00 must be in path	MEISqNodeMessageBOOT_FILE_NOT_FOUND
SqNode: boot rom not recognized	MEISqNodeMessageBOOT_ROM_INVALID
SqNode: Config file and network are different	MEISqNodeMessageCONFIG_NETWORK_MISMATCH
SqNode: Config file format invalid	MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID
SqNode: exceeded maximum synqnet packet limit	MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT

SqNode: firmware download not supported	MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED
SqNode: firmware verify not supported	MEISqNodeMessageVERIFY_NOT_SUPPORTED
SqNode: invalid	MEISqNodeMessageINVALID
SqNode: invalid feedback map attempt	MEISqNodeMessageFEEDBACK_MAP_INVAILD
SqNode: invalid resource table in node module	MEISqNodeMessageINVALID_TABLE
SqNode: invalid string length	MEISqNodeMessageINVALID_STR_LEN
SqNode: invalid service channel specified	MEISqNodeMessageSRVC_CHANNEL_INVALID
SqNode: I/O Module Count Exceeded	MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED
SqNode: I/O Module EEPROM not programmed	MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED
SqNode: I/O Module Incompatibility	MEISqNodeMessageIO_MODULE_INCOMPATIBILITY
SqNode: I/O Module length check failed	MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED
SqNode: I/O Module 3.3V bus current exceeded	MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED
SqNode: I/O Module 24V bus current exceeded	MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED
SqNode: I/O Slice eeprom format	MEISqNodeMessageIO_SLICE_EEPROM_FORMAT
SqNode: I/O Slice initalization fault	MEISqNodeMessageIO_SLICE_INITIALIZATION_ERROR
SqNode: I/O Slice initalization fault too many slices	MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MANY_SLICES
SqNode: I/O Slice initalization fault vendor mismatch	MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH
SqNode: I/O Slice initalization timeout	MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT
SqNode: I/O Slice service already in mode	MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE
SqNode: I/O Slice service attribute not settable	MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE

SqNode: I/O Slice service attribute not supported	MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED
SqNode: I/O Slice service bus error code	MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE
SqNode: I/O Slice service invalid attribute value	MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE
SqNode: I/O Slice service invalid parameter	MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER
SqNode: I/O Slice service not enough data	MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA
SqNode: I/O Slice service not supported	MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED
SqNode: I/O Slice service object does not exist	MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST
SqNode: I/O Slice service receive error	MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR
SqNode: I/O Slice service resource unavailable	MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE
SqNode: I/O Slice service response format	MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT
SqNode: I/O Slice service state conflict	MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT
SqNode: I/O Slice service store operation failure	MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE
SqNode: I/O Slice service timeout	MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT
SqNode: I/O Slice service too many char	MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR
SqNode: I/O Slice service too much data	MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA
SqNode: I/O Slice service unknown error code	MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE
SqNode: I/O Slice service unknown fault code	MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE
SqNode: I/O Slice too much IO	MEISqNodeMessageIO_SLICE_TOO_MUCH_IO
SqNode: I/O Slice topology mismatch	MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH
SqNode: Map and config files are different	MEISqNodeMessageMAP_CONFIG_MISMATCH

SqNode: Monitor config invalid, address not supported	MEISqNodeMessageMONITOR_ADDRESS
SqNode: Monitor config invalid, index not supported	MEISqNodeMessageMONITOR_INDEX
SqNode: network state error	MEISqNodeMessageNETWORK_STATE_ERROR
SqNode: node busy: service cmd ready timeout	MEISqNodeMessageREADY_TIMEOUT
SqNode: Node invalid	MEISqNodeMessageNODE_INVALID
SqNode: node module did not complete the specified operation	MEISqNodeMessageCMD_NOT_SUPPORTED
SqNode: node module doesn't support discovery	MEISqNodeMessageINTERFACE_ERROR1
SqNode: node specific command dispatch error	MEISqNodeMessageDISPATCH_ERROR
SqNode: node specific discovery failure	MEISqNodeMessageDISCOVERY_FAILURE
SqNode: node specific initialization failure	MEISqNodeMessageINIT_FAILURE
SqNode: node type does not match the file provided for download	MEISqNodeMessageFILE_NODE_MISMATCH
SqNode: node firmware download failed	MEISqNodeMessageDOWNLOAD_FAIL
SqNode: node firmware verify failed	MEISqNodeMessageVERIFY_FAIL
SqNode: node failure	MEISqNodeMessageNODE_FAILURE
SqNode: Not in Config File	MEISqNodeMessageNOT_IN_CONFIG_FILE
SqNode: Parameter is read only	MEISqNodeMessagePARAM_READ_ONLY
SqNode: service cmd error	MEISqNodeMessageSRVC_ERROR
SqNode: service cmd response timeout	MEISqNodeMessageRESPONSE_TIMEOUT
SqNode: service cmd unsupported	MEISqNodeMessageSRVC_UNSUPPORTED
SqNode: SFD motor selected, parameter is locked	MEISqNodeMessagePARAM_LOCKED
SqNode: the file provided for download was not found	MEISqNodeMessageFILE_INVALID

SqNode: the header information in the download image is invalid	MEISqNodeMessageINVALID_HEADER
SynqNet	MEISynqNetMessageFIRST
SynqNet: cable length measurement timeout	MEISynqNetMessageCABLE_LENGTH_TIMEOUT
SynqNet: cable length mismatch - network in asynq mode	MEISynqNetMessageCABLE_LENGTH_MISMATCH
SynqNet: cable number is not idle, status is unknown	MEISynqNetMessageLINK_NOT_IDLE
SynqNet: controller sample rate exceeds required analog input sample time	MEISynqNetMessageADC_SAMPLE_FAILURE
SynqNet: controller sample rate must be a integer multiple of all node update periods	MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE
SynqNet: detected an unsupported cable length	MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED
SynqNet Hot Restart: Node Address Assignment Failure	MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT
SynqNet Hot Restart: Not all nodes could be restarted	MEISynqNetMessageHOT_RESTART_NOT_ALL_NODES_RESTARTED
SynqNet Hot Restart: Not in Synq state	MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE
SynqNet Hot Restart: Recovering	MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING
SynqNet Hot Restart: Test Packet Active	MEISynqNetMessageHOT_RESTART_FAIL_TEST_PACKET
SynqNet: idle cable number unknown due to failed node(s)	MEISynqNetMessageIDLE_LINK_UNKNOWN
SynqNet: incomplete motor : fulfill motor packet requirements or remove all packet fields for motor	MEISynqNetMessageINCOMPLETE_MOTOR
SynqNet: interface not available	MEISynqNetMessageINTERFACE_NOT_FOUND
SynqNet: internal buffer size overflow: reduce network payload	MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW
SynqNet: invalid analogIn count	MEISynqNetMessageINVALID_ANALOG_IN_COUNT

SynqNet: invalid analogOut count	MEISynqNetMessageINVALID_ANALOG_OUT_COUNT
SynqNet: invalid auxiliary encoder count	MEISynqNetMessageINVALID_AUX_ENC_COUNT
SynqNet: invalid capture count	MEISynqNetMessageINVALID_CAPTURE_COUNT
SynqNet: invalid command configuration	MEISynqNetMessageINVALID_COMMAND_CFG
SynqNet: invalid compare count	MEISynqNetMessageINVALID_COMPARE_COUNT
SynqNet: invalid digitalIn count	MEISynqNetMessageINVALID_DIGITAL_IN_COUNT
SynqNet: invalid digitalOut count	MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT
SynqNet: invalid feedback count	MEISynqNetMessageINVALID_ENCODER_COUNT
SynqNet: invalid ioInput count	MEISynqNetMessageINVALID_INPUT_COUNT
SynqNet: invalid ioOutput count	MEISynqNetMessageINVALID_OUTPUT_COUNT
SynqNet: invalid monitor field configuration	MEISynqNetMessageINVALID_MONITOR_CFG
SynqNet: invalid probe count	MEISynqNetMessageINVALID_PROBE_CFG
SynqNet: invalid probe depth	MEISynqNetMessageINVALID_PROBE_DEPTH
SynqNet: invalid pulse engine count	MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT
SynqNet: maximum blocks exceeded	MEISynqNetMessageMAX_NODE_ERROR
SynqNet: maximum cable length out of range	MEISynqNetMessageCABLE_LENGTH_INVALID_MAX
SynqNet: maximum motor count exceeded	MEISynqNetMessageMAX_MOTOR_ERROR
SynqNet: minimum cable length out of range	MEISynqNetMessageCABLE_LENGTH_INVALID_MIN
SynqNet: network communication is down	MEISynqNetMessageCOMM_ERROR
SynqNet: network down due to CRC errors	MEISynqNetMessageCOMM_ERROR_CRC
SynqNet: network down due to Rx errors	MEISynqNetMessageCOMM_ERROR_RX
SynqNet: network down due to Rx pack errors	MEISynqNetMessageCOMM_ERROR_RX_FIFO

SynqNet: network down due to Rx packet length errors	MEISynqNetMessageCOMM_ERROR_RX_LEN
SynqNet: network down due to Rx CRC errors	MEISynqNetMessageCOMM_ERROR_RX_CRC
SynqNet: network down due to Rx dribble	MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE
SynqNet: network is currently recovering from a fault	MEISynqNetMessageRECOVERING
SynqNet: network topology cannot be saved or cleared while amplifiers are enabled	MEISynqNetMessageTOPOLOGY_AMPS_ENABLED
SynqNet: network topology is already clear	MEISynqNetMessageTOPOLOGY_CLEAR
SynqNet: network topology is already saved to flash. Use meiSynqNetFlashTopologyClear (...) first	MEISynqNetMessageTOPOLOGY_SAVED
SynqNet: network topology mismatch in dynamic memory - network in asynq mode	MEISynqNetMessageTOPOLOGY_MISMATCH
SynqNet: network topology mismatch in flash memory - network in asynq mode	MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH
SynqNet: node control latency maximum exceeded	MEISynqNetMessageNODE_LATENCY_EXCEEDED
SynqNet: node FPGA version mismatch warning	MEISynqNetMessageNODE_FPGA_VERSION
SynqNet: node MAC version mismatch - network in asynq mode	MEISynqNetMessageNODE_MAC_VERSION
SynqNet: node PLL unable to lock with drive	MEISynqNetMessagePLL_ERROR
SynqNet: node specific initialization failed	MEISynqNetMessageNODE_INIT_FAIL
SynqNet: Nodes being shutdown need to be consecutive	MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE
SynqNet: nominal cable length out of range	MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL
SynqNet: no nodes found on network	MEISynqNetMessageNO_NODES_FOUND

SynqNet: no timing data available in module	MEISynqNetMessageNO_TIMING_DATA_AVAIL
SynqNet: only supported with ring topologies	MEISynqNetMessageRING_ONLY
SynqNet: Shutting down nodes requires recovery to be enabled	MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED
SynqNet: Shutting down the nodes requested will result in stranding working drives on the system, check topology	MEISynqNetMessageSHUTDOWN_NODES_STRANDED
SynqNet: synqNet invalid	MEISynqNetMessageSYNQNET_INVALID
SynqNet: timeout : discovery problem	MEISynqNetMessageDISCOVERY_TIMEOUT
SynqNet: timeout : reset complete packet	MEISynqNetMessageRESET_ACK_TIMEOUT
SynqNet: timeout : reset request packet	MEISynqNetMessageRESET_REQ_TIMEOUT
SynqNet: too many motors configured for this node	MEISynqNetMessageINVALID_MOTOR_COUNT
SynqNet: unexpected network state	MEISynqNetMessageSTATE_ERROR
SynqNet: unrecoverable network scheduling error	MEISynqNetMessageSCHEDULING_ERROR
Timeout	MPIMessageTIMEOUT
Unsupported	MPIMessageUNSUPPORTED
WinNT	MEIPlatformMessageFIRST
WinNT: copy64 failure	MEIPlatformMessageCOPY64_FAILURE
WinNT: device error	MEIPlatformMessageDEVICE_ERROR
WinNT: device invalid	MEIPlatformMessageDEVICE_INVALID
WinNT: device map error	MEIPlatformMessageDEVICE_MAP_ERROR
WinNT: platform invalid	MEIPlatformMessagePLATFORM_INVALID

Table of Error Messages / Return Values

The table below provides an exhaustive list of the error messages / return values that are returned by the message.exe utility. For a complete description of the error message and tips on troubleshooting, click on the links under the Define column.

(Sorted Alphabetically by Define)

Define (sort by message.exe output)	String (sort alphabetically by String)
MEICaptureMessageCAPTURE_ARMED	Capture: capture currently armed
MEICaptureMessageCAPTURE_NOT_ENABLED	Capture: capture not enabled
MEICaptureMessageCAPTURE_STATE_INVALID	Capture: capture is in an invalid state
MEICaptureMessageFIRST	Capture
MEICaptureMessageGLOBAL_ALREADY_ENABLED	Capture: global config already enabled on another capture on this block
MEICaptureMessageGLOBAL_CONFIG_ERR	Capture: global capture cannot be enabled and be a trigger source at the same time
MEICaptureMessageINVALID_EDGE	Capture: vaild capture edge required
MEICaptureMessageNOT_MAPPED	Capture: capture does not map to existing hardware
MEICaptureMessageSECONDARY_INDEX_INVALID	Capture: secondary index can not be used for multiple source capturing
MEICaptureMessageUNSUPPORTED_PRIMARY	Capture: capture not supported on primary encoder
MEICaptureMessageUNSUPPORTED_SECONDARY	Capture: capture not supported on secondary encoder
MEIClientMessageCLIENT_INVALID	Client: client invalid
MEIClientMessageFIRST	Client
MEIClientMessageHEADER_INVALID	Client: header invalid
MEIClientMessageMETHOD_INVALID	Client: method invalid
MEICommandMessageCOMMAND_INVALID	Command: command invalid

MEIControlMessageBAD_FPGA_SOCKET_DATA	Control: Bad FPGA hardware socket data
MEIControlMessageFIRMWARE_INVALID	Control: firmware invalid
MEIControlMessageFIRMWARE_VERSION	Control: firmware version mismatch
MEIControlMessageFIRMWARE_VERSION_NONE	Control: No firmware found (factory default)
MEIControlMessageFPGA_SOCKETS	Control: Too many FPGA hardware socket types
MEIControlMessageINVALID_BLOCK_COUNT	Control: Invalid Motion Block count
MEIControlMessageIO_BIT_INVALID	Control: I/O bit selected is unavailable
MEIControlMessageNO_FPGA_SOCKET	Control: FPGA hardware socket does not exist
MPIControlMessagePACK_ALIGNMENT	Control: Application compiled with invalid packing alignment
MPIControlMessageRECORDER_RUNNING	Control: cannot configure while recorders are running
MPIControlMessageSAMPLE_RATE_TO_LOW	Control: sample rate value must be greater than or equal to 1000
MPIControlMessageSAMPLE_RATE_TO_HIGH	Control: sample rate value must be less than or equal to 100000
MEIControlMessageSYNQNET_OBJECTS	Control: Too many SynqNet objects
MEIControlMessageSYNQNET_STATE	Control: SynqNet state is invalid
MEIDriveMapMessageDRIVE_PARAM_READ_ONLY	driveMap: Drive parameter is read-only
MEIDriveMapMessageFIRST	driveMap
MEIDriveMapMessageMAP_FILE_FORMAT_INVALID	driveMap: Format error in drive map file
MEIDriveMapMessageMAP_FILE_OPEN_ERROR	driveMap: Could not open drive map file
MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP	driveMap: Node type not found in drives map files
MEIDriveMapMessageVERSION_NOT_FOUND_IN_MAP	driveMap: Drive firmware version not found in drives map file (see drive vendor for .dm file update)

MEIElementMessageELEMENT_INVALID	Element: element invalid
MEIElementMessageFIRST	Element
MEIFlashMessageFIRST	Flash
MEIFlashMessageFLASH_INVALID	Flash: flash invalid
MEIFlashMessageFLASH_VERIFY_ERROR	Flash: flash verify error
MEIFlashMessageFLASH_WRITE_ERROR	Flash: flash write error
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	Flash: write to flash failed. Network topology has not been saved to flash - use meiSynqNetTopologySave()
MEIFlashMessagePATH	Flash: flash file path is too long
MEIListMessageELEMENT_INVALID	List: element invalid
MEIListMessageELEMENT_NOT_FOUND	List: element not found
MEIListMessageFIRST	List
MEIListMessageLIST_INVALID	List: list invalid
MEIMapMessageFILE_INVALID	Map: file invalid
MEIMapMessageFIRST	Map
MEIMapMessageINDEX_INVALID	Map: index invalid
MEIMapMessageNAME_INVALID	Map: name invalid
MEIMapMessageNAME_NOT_FOUND	Map: name not found
MEIMotionMessageBAD_PATH_DATA	Motion: Bad Path Data
MEIMotionMessageNO_AXES_MAPPED	Motion: No Axis mapped
MEIMotionMessageRESERVED0	Motion: RESERVED0
MEIMotionMessageRESERVED1	Motion: RESERVED1
MEIMotionMessageRESERVED2	Motion: RESERVED2
MEIMotorMessageDEMAND_MODE_UNSUPPORTED	Motor: specified demand mode not supported by drive
MEIMotorMessageDEMAND_MODE_NOT_SET	Motor: unable to switch demand mode while the amplifier is enabled

MEIMotorMessageDISABLE_ACTION_INVALID	Motor: cannot set motor type to STEPPER when disable action is CMD_EQ_ACT
MEIMotorMessageFEEDBACK_REVERSAL_NA	Motor: unable to invert feedback for specified encoder type
MEIMotorMessageFILTER_DISABLE_NA	Motor: unable to disable the filter for specified encoder type
MEIMotorMessageHARDWARE_NOT_FOUND	Motor: hardware not found
MEIMotorMessageMOTOR_NOT_ENABLED	Motor: motor not enabled
MEIMotorMessagePHASE_FINDING_FAILED	Motor: Motor phase finding failure
MEIMotorMessagePULSE_WIDTH_INVALID	Motor: stepper Pulse Width out of range (.00000006 < pulseWidth < .00100006)
MEIMotorMessageSECONDARY_ENCODER_NA	Motor: secondary encoder not available
MEIMotorMessageSTEPPER_INVALID	Motor: cannot set disable action to CMD_EQ_ACT when motor type is STEPPER
MEIPacketMessageADDRESS_INVALID	Packet: address invalid
MEIPacketMessageCOMM_ERROR	Packet: communication error
MEIPacketMessageCONNECTION_CLOSED	Packet: connection closed
MEIPacketMessageFIRST	Packet
MEIPacketMessagePACKET_INVALID	Packet: packet invalid
MEIPacketMessageRECV_ERROR	Packet: receive error
MEIPacketMessageSEND_ERROR	Packet: send error
MEIPlatformMessageCOPY64_FAILURE	WinNT: copy64 failure
MEIPlatformMessageDEVICE_ERROR	WinNT: device error
MEIPlatformMessageDEVICE_INVALID	WinNT: device invalid
MEIPlatformMessageDEVICE_MAP_ERROR	WinNT: device map error
MEIPlatformMessagePLATFORM_INVALID	WinNT: platform invalid
MEIServerMessageFIRST	Server
MEIServerMessageHEADER_INVALID	Server: header invalid

MEIServerMessageMETHOD_INVALID	Server: method invalid
MEIServerMessageSERVER_INVALID	Server: server invalid
MEISqNodeMessageBOOT_FILE_NOT_FOUND	SqNode: Boot file not found or corrupt, kollmorgen_ember.a00 must be in path
MEISqNodeMessageBOOT_ROM_INVALID	SqNode: boot rom not recognized
MEISqNodeMessageCMD_NOT_SUPPORTED	SqNode: node module did not complete the specified operation
MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID	SqNode: Config file format invalid
MEISqNodeMessageCONFIG_NETWORK_MISMATCH	SqNode: Config file and network are different
MEISqNodeMessageDISCOVERY_FAILURE	SqNode: node specific discovery failure
MEISqNodeMessageDISPATCH_ERROR	SqNode: node specific command dispatch error
MEISqNodeMessageDOWNLOAD_FAIL	SqNode: node firmware download failed
MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED	SqNode: firmware download not supported
MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT	SqNode: exceeded maximum synqnet packet limit
MEISqNodeMessageFEEDBACK_MAP_INVAILD	SqNode: invalid feedback map attempt
MEISqNodeMessageFILE_INVALID	SqNode: the file provided for download was not found
MEISqNodeMessageFILE_NODE_MISMATCH	SqNode: node type does not match the file provided for download
MEISqNodeMessageFIRST	SqNode
MEISqNodeMessageINIT_FAILURE	SqNode: node specific initialization failure
MEISqNodeMessageINTERFACE_ERROR1	SqNode: node module doesn't support discovery
MEISqNodeMessageINVALID	SqNode: invalid
MEISqNodeMessageINVALID_HEADER	SqNode: the header information in the download image is invalid

MEISqNodeMessageINVALID_STR_LEN	SqNode : invalid string length
MEISqNodeMessageINVALID_TABLE	SqNode : invalid resource table in node module
MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED	SqNode: I/O Module Count Exceeded
MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED	SqNode: I/O Module EEPROM not programmed
MEISqNodeMessageIO_MODULE_INCOMPATIBILITY	SqNode: I/O Module Incompatibility
MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED	SqNode: I/O Module length check failed
MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED	SqNode: I/O Module 3.3V bus current exceeded
MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED	SqNode: I/O Module 24V bus current exceeded
MEISqNodeMessageIO_SLICE_EEPROM_FORMAT	SqNode: I/O Slice eeprom format
MEISqNodeMessageIO_SLICE_INITIALIZATION_ERROR	SqNode: I/O Slice initalization fault
MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MANY_SLICES	SqNode: I/O Slice initalization fault too many slices
MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH	SqNode: I/O Slice initalization fault vendor mismatch
MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT	SqNode: I/O Slice initalization timeout
MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE	SqNode: I/O Slice service already in mode
MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE	SqNode: I/O Slice service attribute not settable
MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED	SqNode: I/O Slice service attribute not supported
MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE	SqNode: I/O Slice service bus error code
MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE	SqNode: I/O Slice service invalid attribute value
MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER	SqNode: I/O Slice service invalid parameter
MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA	SqNode: I/O Slice service not enough data

MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED	SqNode: I/O Slice service not supported
MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST	SqNode: I/O Slice service object does not exist
MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR	SqNode: I/O Slice service receive error
MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE	SqNode: I/O Slice service resource unavailable
MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT	SqNode: I/O Slice service response format
MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT	SqNode: I/O Slice service state conflict
MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE	SqNode: I/O Slice service store operation failure
MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT	SqNode: I/O Slice service timeout
MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR	SqNode: I/O Slice service too many char
MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA	SqNode: I/O Slice service too much data
MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE	SqNode: I/O Slice service unknown error code
MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE	SqNode: I/O Slice service unknown fault code
MEISqNodeMessageIO_SLICE_TOO_MUCH_IO	SqNode: I/O Slice too much IO
MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH	SqNode: I/O Slice topology mismatch
MEISqNodeMessageMAP_CONFIG_MISMATCH	SqNode: Map and config files are different
MEISqNodeMessageMONITOR_ADDRESS	SqNode: Monitor config invalid, address not supported
MEISqNodeMessageMONITOR_INDEX	SqNode: Monitor config invalid, index not supported
MEISqNodeMessageNODE_FAILURE	SqNode: node failure
MEISqNodeMessageNODE_INVALID	SqNode: Node invalid
MEISqNodeMessageNOT_IN_CONFIG_FILE	SqNode: Not in Config File
MEISqNodeMessagePARAM_LOCKED	SqNode: SFD motor selected, parameter is locked

MEISqNodeMessagePARAM_READ_ONLY	SqNode: Parameter is read only
MEISqNodeMessageREADY_TIMEOUT	SqNode: node busy : service cmd ready timeout
MEISqNodeMessageRESPONSE_TIMEOUT	SqNode: service cmd response timeout
MEISqNodeMessageSRVC_CHANNEL_INVALID	SqNode: invalid service channel specified
MEISqNodeMessageSRVC_ERROR	SqNode: service cmd error
MEISqNodeMessageSRVC_UNSUPPORTED	SqNode: service cmd unsupported
MEISqNodeMessageSTATE_ERROR	SqNode: network state error
MEISqNodeMessageVERIFY_FAIL	SqNode: node firmware verify failed
MEISqNodeMessageVERIFY_NOT_SUPPORTED	SqNode: firmware verify not supported
MEISynqNetMessageADC_SAMPLE_FAILURE	SynqNet: controller sample rate exceeds required analog input sample time
MEISynqNetMessageCABLE_LENGTH_INVALID_MAX	SynqNet: maximum cable length out of range
MEISynqNetMessageCABLE_LENGTH_INVALID_MIN	SynqNet: minimum cable length out of range
MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL	SynqNet: nominal cable length out of range
MEISynqNetMessageCABLE_LENGTH_MISMATCH	SynqNet: cable length mismatch - network in asynq mode
MEISynqNetMessageCABLE_LENGTH_TIMEOUT	SynqNet: cable length measurement timeout
MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED	SynqNet: detected an unsupported cable length
MEISynqNetMessageCOMM_ERROR	SynqNet: network communication is down
MEISynqNetMessageCOMM_ERROR_CRC	SynqNet: network down due to CRC errors
MEISynqNetMessageCOMM_ERROR_RX	SynqNet: network down due to Rx errors
MEISynqNetMessageCOMM_ERROR_RX_CRC	SynqNet: network down due to Rx CRC errors

MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE	SynqNet: network down due to Rx dribble
MEISynqNetMessageCOMM_ERROR_RX_FIFO	SynqNet: network down due to Rx pack errors
MEISynqNetMessageCOMM_ERROR_RX_LEN	SynqNet: network down due to Rx packet length errors
MEISynqNetMessageDISCOVERY_TIMEOUT	SynqNet: timeout : discovery problem
MEISynqNetMessageFIRST	SynqNet
MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT	SynqNet Hot Restart: Node Address Assignment Failure
MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE	SynqNet Hot Restart: Not in Synq state
MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING	SynqNet Hot Restart: Recovering
MEISynqNetMessageHOT_RESTART_FAIL_TEST_PACKET	SynqNet Hot Restart: Test Packet Active
MEISynqNetMessageHOT_RESTART_NOT_ALL_NODES_RESTARTED	SynqNet Hot Restart: Not all nodes could be restarted
MEISynqNetMessageIDLE_LINK_UNKNOWN	SynqNet: idle cable number unknown due to failed node(s)
MEISynqNetMessageINCOMPLETE_MOTOR	SynqNet: incomplete motor : fulfill motor packet requirements or remove all packet fields for motor
MEISynqNetMessageINTERFACE_NOT_FOUND	SynqNet: interface not available
MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW	SynqNet: internal buffer size overflow: reduce network payload
MEISynqNetMessageINVALID_ANALOG_IN_COUNT	SynqNet: invalid analogIn count
MEISynqNetMessageINVALID_ANALOG_OUT_COUNT	SynqNet: invalid analogOut count
MEISynqNetMessageINVALID_AUX_ENC_COUNT	SynqNet: invalid auxiliary encoder count
MEISynqNetMessageINVALID_CAPTURE_COUNT	SynqNet: invalid capture count
MEISynqNetMessageINVALID_COMMAND_CFG	SynqNet: invalid command configuration
MEISynqNetMessageINVALID_COMPARE_COUNT	SynqNet: invalid compare count

MEISynqNetMessageINVALID_DIGITAL_IN_COUNT	SynqNet: invalid digitalIn count
MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT	SynqNet: invalid digitalOut count
MEISynqNetMessageINVALID_ENCODER_COUNT	SynqNet: invalid feedback count
MEISynqNetMessageINVALID_INPUT_COUNT	SynqNet: invalid ioInput count
MEISynqNetMessageINVALID_MONITOR_CFG	SynqNet: invalid monitor field configuration
MEISynqNetMessageINVALID_MOTOR_COUNT	SynqNet: too many motors configured for this node
MEISynqNetMessageINVALID_OUTPUT_COUNT	SynqNet: invalid ioOutput count
MEISynqNetMessageINVALID_PROBE_CFG	SynqNet: invalid probe count
MEISynqNetMessageINVALID_PROBE_DEPTH	SynqNet: invalid probe depth
MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT	SynqNet: invalid pulse engine count
MEISynqNetMessageLINK_NOT_IDLE	SynqNet: cable number is not idle, status is unknown
MEISynqNetMessageMAX_MOTOR_ERROR	SynqNet: maximum motor count exceeded
MEISynqNetMessageMAX_NODE_ERROR	SynqNet: maximum blocks exceeded
MEISynqNetMessageNO_NODES_FOUND	SynqNet: no nodes found on network
MEISynqNetMessageNO_TIMING_DATA_AVAIL	SynqNet: no timing data available in module
MEISynqNetMessageNODE_FPGA_VERSION	SynqNet: node FPGA version mismatch warning
MEISynqNetMessageNODE_INIT_FAIL	SynqNet: node specific initialization failed
MEISynqNetMessageNODE_LATENCY_EXCEEDED	SynqNet: node control latency maximum exceeded
MEISynqNetMessageNODE_MAC_VERSION	SynqNet: node MAC version mismatch - network in asynq mode
MEISynqNetMessagePLL_ERROR	SynqNet: node PLL unable to lock with drive
MEISynqNetMessageRECOVERING	SynqNet: network is currently recovering from a fault

MEISynqNetMessageRESET_ACK_TIMEOUT	SynqNet: timeout : reset complete packet
MEISynqNetMessageRESET_REQ_TIMEOUT	SynqNet: timeout : reset request packet
MEISynqNetMessageRING_ONLY	SynqNet: only supported with ring topologies
MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE	SynqNet: controller sample rate must be a integer multiple of all node update periods
MEISynqNetMessageSCHEDULING_ERROR	SynqNet: unrecoverable network scheduling error
MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE	SynqNet: Nodes being shutdown need to be consecutive
MEISynqNetMessageSHUTDOWN_NODES_STRANDED	SynqNet: Shutting down the nodes requested will result in stranding working drives on the system, check topology
MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED	SynqNet: Shutting down nodes requires recovery to be enabled
MEISynqNetMessageSTATE_ERROR	SynqNet: unexpected network state
MEISynqNetMessageSYNQNET_INVALID	SynqNet: synqNet invalid
MEISynqNetMessageTOPOLOGY_AMPS_ENABLED	SynqNet: network topology cannot be saved or cleared while amplifiers are enabled
MEISynqNetMessageTOPOLOGY_CLEAR	SynqNet: network topology is already clear
MEISynqNetMessageTOPOLOGY_MISMATCH	SynqNet: network topology mismatch in dynamic memory - network in asynq mode
MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH	SynqNet: network topology mismatch in flash memory - network in asynq mode
MEISynqNetMessageTOPOLOGY_SAVED	SynqNet: network topology is already saved to flash. Use meiSynqNetFlashTopologyClear (...) first
MEIPlatformMessageDEVICE_ERROR	WinNT: device error
MEIPlatformMessageDEVICE_INVALID	WinNT: device invalid

MEIPlatformMessageDEVICE_MAP_ERROR	WinNT: device map error
MEIPlatformMessageFIRST	WinNT
MEIPlatformMessagePLATFORM_INVALID	WinNT: platform invalid
MPIAxisMessageAXIS_INVALID	Axis: axis invalid
MPIAxisMessageCOMMAND_NOT_SET	Axis: unable to set command position
MPIAxisMessageFIRST	Axis
MPIAxisMessageNOT_MAPPED_TO_MS	Axis: not mapped to motion supervisor
MPICanMessageBUS_OFF	Can: Bus off
MPICanMessageCAN_INVALID	Can: can invalid
MPICanMessageCOMMAND_PROTOCOL	Can: DPR Command protocol
MPICanMessageFILE_FORMAT_ERROR	Can: file format incorrect
MPICanMessageFIRMWARE_INVALID	Can: firmware invalid
MPICanMessageFIRMWARE_VERSION	Can: firmware version mismatch
MPICanMessageFIRST	Can
MPICanMessageINTERFACE_NOT_FOUND	Can: interface not found
MPICanMessageIO_NOT_SUPPORTED	Can: IO not supported by node
MPICanMessageNODE_DEAD	Can: node dead
MPICanMessageNOT_INITIALIZED	Can: firmware not initialized
MPICanMessageRTR_TX_OVERFLOW	Can: RTR transmit buffer overflow
MPICanMessageRX_BUFFER_EMPTY	Can: Receive buffer empty
MPICanMessageSDO_ABORT	Can: SDO abort
MPICanMessageSDO_PROTOCOL	Can: SDO protocol
MPICanMessageSDO_TIMEOUT	Can: SDO Timeout
MPICanMessageSIGNATURE_INVALID	Can: Signature invalid
MPICanMessageTX_OVERFLOW	Can: transmit buffer overflow
MPICanMessageUSER_ABORT	Can: user aborted operation

MPICaptureMessageCAPTURE_INVALID	Capture: invalid capture number
MPICaptureMessageCAPTURE_TYPE_INVALID	Capture: invalid capture type
MPICaptureMessageENCODER_INVALID	Capture: invalid encoder index parameter
MPICaptureMessageMOTOR_INVALID	Capture: invalid motor number
MPICommandMessageFIRST	Command
MPICommandMessagePARAM_INVALID	Command: param invalid
MPICommandMessageTYPE_INVALID	Command: type invalid
MPICompensatorMessageAXIS_NOT_ENABLED	Compensator: axis not enabled
MPICompensatorMessageCOMPENSATOR_INVALID	Compensator: compensator object invalid
MPICompensatorMessageDIMENSION_NOT_SUPPORTED	Compensator: table dimension not supported
MPICompensatorMessageFIRST	Compensator
MPICompensatorMessageNOT_CONFIGURED	Compensator: compensator object not configured
MPICompensatorMessageNOT_ENABLED	Compensator: object not enabled
MPICompensatorMessagePOSITION_DELTA_INVALID	Compensator: position delta invalid
MPICompensatorMessageTABLE_SIZE_ERROR	Compensator: table size is too small. adjust MPIControlConfig. compensatorPostionCount
MPIControlMessageADC_COUNT_INVALID	Control: adcCount exceeds configuration limit
MPIControlMessageADDRESS_INVALID	Control: address invalid
MPIControlMessageAUXDAC_COUNT_INVALID	Control: auxDacCount exceeds configuration limit
MPIControlMessageAXIS_COUNT_INVALID	Control: axisCount exceeds configuration limit
MPIControlMessageAXIS_FRAME_COUNT_INVALID	Control: invalid axisFrameCount
MPIControlMessageAXIS_RUNNING	Control: cannot configure while axes are running
MPIControlMessageCAPTURE_COUNT_INVALID	Control: captureCount exceeds configuration limit

MPIControlMessageCMDDAC_COUNT_INVALID	Control: cmdDacCount exceeds configuration limit
MPIControlMessageCOMPARE_COUNT_INVALID	Control: compareCount exceeds configuration limit
MPIControlMessageCOMPENSATOR_COUNT_INVALID	Control: compensatorCount exceeds configuration limit
MPIControlMessageCONTROL_INVALID	Control: control invalid
MPIControlMessageCONTROL_NUMBER_INVALID	Control: control number exceeds maximum limit
MPIControlMessageEXTERNAL_MEMORY_OVERFLOW	Control: out of external memory
MPIControlMessageFILTER_COUNT_INVALID	Control: filterCount exceeds configuration limit
MPIControlMessageFIRST	Control
MPIControlMessageINTERRUPTS_DISABLED	Control: interrupts disabled
MPIControlMessageLIBRARY_VERSION	Control: application is not compatible with MPI DLL
MPIControlMessageMOTION_COUNT_INVALID	Control: motionCount exceeds configuration limit
MPIControlMessageMOTOR_COUNT_INVALID	Control: motorCount exceeds configuration limit
MPIControlMessageRECORDER_COUNT_INVALID	Control: recorderCount exceeds configuration limit
MPIControlMessageSAMPLE_RATE_TO_LOW	Control: sample rate value must be greater than or equal to 1000
MPIControlMessageTYPE_INVALID	Control: type invalid
MPIEventMessageEVENT_INVALID	Event: event invalid
MPIEventMessageFIRST	Event
MPIEventMgrMessageEVENTMGR_INVALID	EventMgr: eventMgr invalid
MPIEventMgrMessageFIRST	EventMgr
MPIFilterMessageCONVERSION_DIV_BY_0	Filter: Divide by zero in conversion
MPIFilterMessageFILTER_INVALID	Filter: filter invalid
MPIFilterMessageFIRST	Filter
MPIFilterMessageINVALID_ALGORITHM	Filter: filter algorithm invalid

MPIFilterMessageINVALID_DRATE	Filter: DRate value out of range (0-7)
MPIFilterMessageINVALID_FILTER_FORM	Filter: Invalid filter form
MPIFilterMessageINVALID_KA1	Filter: KA1 value out of range (0-.999)
MPIFilterMessageSECTION_NOT_ENABLED	Filter: Specified postfilter section not enabled
MPIMessageARG_INVALID	Argument invalid
MPIMessageFATAL_ERROR	Fatal error
MPIMessageFILE_CLOSE_ERROR	File close error
MPIMessageFILE_MISMATCH	Firmware File Mismatch
MPIMessageFILE_OPEN_ERROR	File open error
MPIMessageFILE_READ_ERROR	File read error
MPIMessageFILE_WRITE_ERROR	File write error
MPIMessageHANDLE_INVALID	Handle invalid
MPIMessageNO_MEMORY	Out of memory
MPIMessageOBJECT_FREED	Object freed
MPIMessageOBJECT_IN_USE	Object in use
MPIMessageOBJECT_NOT_ENABLED	Object not enabled
MPIMessageOBJECT_NOT_FOUND	Object not found
MPIMessageOBJECT_ON_LIST	Object on list
MPIMessageOK	OK
MPIMessagePARAM_INVALID	Parameter invalid
MPIMessageTIMEOUT	Timeout
MPIMessageUNSUPPORTED	Unsupported
MPIMotionMessageATTRIBUTE_INVALID	Motion: attribute invalid
MPIMotionMessageAUTO_START	Motion: auto-start
MPIMotionMessageAXIS_COUNT	Motion: axis count invalid
MPIMotionMessageAXIS_FRAME_COUNT	Motion: axis frame count invalid

MPIMotionMessageAXIS_NOT_FOUND	Motion: axis not found
MPIMotionMessageERROR	Motion: MPIStateERROR
MPIMotionMessageFIRST	Motion
MPIMotionMessageFRAME_BUFFER_TOO_SMALL	Motion: frame buffer too small
MPIMotionMessageFRAMES_EMPTY	Motion: frame buffer empty
MPIMotionMessageFRAMES_LOW	Motion: frame buffer low
MPIMotionMessageIDLE	Motion: MPIStateIDLE
MPIMotionMessageILLEGAL_DELAY	Motion: illegal delay
MPIMotionMessageMOTION_INVALID	Motion: motion invalid
MPIMotionMessageMOVING	Motion: MPIStateMOVING
MPIMotionMessagePATH_ERROR	Motion: path error
MPIMotionMessagePROFILE_ERROR	Motion: profile error
MPIMotionMessagePROFILE_ERROR_NOT_SUPPORTED	Motion: profile not supported
MPIMotionMessageSTOPPED	Motion: MPIStateSTOPPED
MPIMotionMessageSTOPPING	Motion: MPIStateSTOPPING
MPIMotionMessageSTOPPING_ERROR	Motion: MPIStateSTOPPING_ERROR
MPIMotionMessageTYPE_INVALID	Motion: type invalid
MPIMotorMessageFIRST	Motor
MPIMotorMessageMOTOR_INVALID	Motor: motor invalid
MPIMotorMessageMOTOR_TYPE_INVALID	Motor: motor type invalid
MPINotifyMessageFIRST	Notify
MPINotifyMessageNOTIFY_INVALID	Notify: notify invalid
MPINotifyMessageWAIT_IN_PROGRESS	Notify: wait in progress
MPIPathMessageARC_ILLEGAL_DIMENSION	Path: Arc Dimension Out of Range
MPIPathMessageFIRST	Path
MPIPathMessageHELIX_ILLEGAL_DIMENSION	Path: Helix Dimension Out of Range

MPIPathMessageILLEGAL_ACCELERATION	Element: Acceleration must be positive
MPIPathMessageILLEGAL_DIMENSION	Path: Dimension Out of Range
MPIPathMessageILLEGAL_ELEMENT	Path: Illegal Element
MPIPathMessageILLEGAL_RADIUS	Path: Illegal Radius
MPIPathMessageILLEGAL_TIMESLICE	Element: timeSlice must be positive
MPIPathMessageILLEGAL_VELOCITY	Element: Velocity must be positive
MPIPathMessageINVALID_BLENDING	Element: blending cannot be used with path interpolation method
MPIPathMessagePATH_INVALID	Path: Path invalid
MPIPathMessagePATH_TOO_LONG	Path: Path too long
MPIProbeMessageFIRST	Probe
MPIProbeMessageNODE_INVALID	Probe: invalid node number
MPIProbeMessagePROBE_INVALID	Probe: invalid probe number
MPIProbeMessagePROBE_TYPE_INVALID	Probe: invalid probe type
MPIRecorderMessageFIRST	Recorder
MPIRecorderMessageNO_RECORDERS_AVAIL	Recorder: no recorders available
MPIRecorderMessageNOT_CONFIGURED	Recorder: not configured
MPIRecorderMessageNOT_ENABLED	Recorder: not enabled
MPIRecorderMessageRECORDER_INVALID	Recorder: recorder invalid
MPIRecorderMessageRUNNING	Recorder: cannot configure while running
MPIRecorderMessageSTARTED	Recorder: already started
MPIRecorderMessageSTOPPED	Recorder: already stopped
MPISequenceMessageCOMMAND_COUNT	Sequence: command count invalid
MPISequenceMessageCOMMAND_NOT_FOUND	Sequence: command not found
MPISequenceMessageFIRST	Sequence
MPISequenceMessageSEQUENCE_INVALID	Sequence: sequence invalid

MPISequenceMessageSTARTED	Sequence: MPISequenceStateSTARTED
MPISequenceMessageSTOPPED	Sequence: MPISequenceStateSTOPPED

Descriptions of Error Messages / Return Values

See Also: Table of Error Messages / Return Values

A - B - C - D - E - F - G - H - I - J - K - L - M - N - O - P - Q - R - S - T - U - V - W - X - Y - Z

General Return Values

MPIMessageOK
The operation was successful. This message code is returned by a method when the operation completed successfully. If a message code other than MPIMessageOK is returned by a method, then an error occurred or operation status information is being reported. Check return values from ALL method calls. The only exceptions are methods that return object handles. For example, the mpi*Object*Create(...) methods return a handle to an object.

MPIMessageARG_INVALID
An argument is not valid. This message code is returned by a method when one or more arguments fail an integrity check. The following integrity checks are made, depending on the argument types:
- Pointers and addresses are not NULL.
- Memory addresses are within a valid range.
- Object maps have a valid number of members.

MPIMessagePARAM_INVALID
A parameter is not valid. Parameters are the data inside an argument. This message code is returned by a method when one or more parameters fail an integrity check. The following integrity checks are made, depending on the parameter types:
- Pointers and addresses in structures are not NULL.
- Memory addresses in structures are within a valid range.
- Object numbers are within a valid range.
- Values in structures are within a valid range.

MPIMessageHANDLE_INVALID
An object handle is not valid. This message code is returned by a method when the object handle argument is NULL. To correct this problem, create an object using an object create method. For example, to create an axis object use mpiAxisCreate(...).

MPIMessageNO_MEMORY
Memory is not available. This message code is returned by a method when an object, thread or buffer is not created due to a memory allocation failure. All host memory allocation occurs through meiPlatformAlloc(...). To correct this problem, check your host computer resources and remove unused components or add more memory.
NOTE: Some methods may make several memory allocations or may call other methods that allocate memory. If a memory allocation fails, the previous allocations will not be freed.

MPIMessageOBJECT_FREED
The object has been freed. This message code is set within the object when the object is deleted. This message code is used internally. This message code is returned by an object delete method if the object has already been deleted. To prevent this problem, make sure to delete objects only one time.

MPIMessageOBJECT_NOT_ENABLED
The object is not active in the controller. This message code is returned by a method if real-time data or a handshake is required between the MPI and the controller, and the specified object is not enabled in the controller. To correct this problem, use mpiControlConfigSet(...) to enable the object, by setting the object count to greater than the specified object number. For example, to enable motor objects 0 to 3, set motorCount to 4.

MPIMessageOBJECT_NOT_FOUND
The object is not found on an object list. This message code is returned by a method when an object does not exist on the specified list or the object list does not exist. This message code can be returned from methods that insert or remove objects from a list. To correct this problem, specify an object that exists on the list or specify a different object list containing the object. To prevent object creation/deletion problems, delete objects in the reverse order they were created.

MPIMessageOBJECT_ON_LIST
The object is a member of a list. This message code is returned when a method tries to add an object to a list and the object already exists on the list. It is also returned when a method tries to delete an object when the object exists on a list. For example, if an axis object is a member of a list of axis objects or an axis object is associated with a motion object, calling mpiAxisDelete(...) will return the object on list message code. To correct this problem when adding an object to a list, check to make sure the object is not already a member. To correct this problem when deleting an object, remove the object from the list and delete the parent object. To prevent object creation/deletion problems, delete objects in the reverse order they were created.

MPIMessageOBJECT_IN_USE
This message is returned when an operation cannot be completed because the object is currently in use. For example, when threadDelete(…) from the apputil library attempts to delete a thread that is currently running, MPIMessage*OBJECT*_IN_USE will be returned. To correct this problem, make sure the object is not in use before you attempt the operation.

MPIMessageTIMEOUT
The wait time has expired. This message code is returned by a method waiting for a response from the controller, a hardware resource, or a semaphore lock and the maximum wait time value expired. The library uses timeout values to prevent lock-up conditions when unexpected behavior occurs. To correct this problem, check the hardware health, power, and network connections.

MPIMessageUNSUPPORTED
The software or controller does not support a feature. This message code is returned by a method if the MPI library, controller, or network device does not support a feature.

MPIMessageFATAL_ERROR
The software or hardware failed. This message code is returned by a method when an inexplicable data value is read from the controller or host memory. This message code indicates a serious

problem with the host computer, memory, or controller. Contact an MEI applications engineer if you receive this message code.

MPIMessageFILE_CLOSE_ERROR
An error occurred when closing a file. This message code is returned by a method that fails to close a file. Internally, files are closed using meiPlatformClose(...) which makes a platform specific call. File closing errors indicate a missing file, bad file, or other file system problems.

MPIMessageFILE_OPEN_ERROR
An error occurred when opening a file. This message code is returned by a method that fails to open a file. Internally, files are opened using meiPlatformOpen(...) which makes a platform specific call. File opening errors indicate a missing file, bad file, or other file system problems.

MPIMessageFILE_READ_ERROR
An error occurred when reading from a file. This message code is returned by a method that fails to read from a file. Internally, files are read using meiPlatformFileRead(...) which makes a platform specific call. File reading errors indicate a bad file or other file system problems.

MPIMessageFILE_WRITE_ERROR
An error occurred when writing to a file. This message code is returned by a method that fails to write to a file. Internally, files are written using meiPlatformFileWrite(...) which makes a platform specific call. File writing errors indicate a bad file or other file system problems.

MPIMessageFILE_MISMATCH
The file being downloaded does not match the installed hardware. For example, if a user attempts to download an XMP file to a ZMP controller or a ZMP file to an XMP controller, the MPI will return error code MPIMessageFILE_MISMATCH.

Axis

MPIAxisMessageFIRST
First message in list of messages, placeholder only.

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

MPIAxisMessageCOMMAND_NOT_SET
The axis command position did not get set. This message code is returned by mpiAxisCommandPositionSet(...) if the controller's command position does not match the specified value. Internally, the mpiAxisCommandPositionSet(...) method requests the controller to change the command position, waits for the controller to process the request, and reads back the controller's command position. There are several cases where the controller will calculate a new command position to replace the requested command position. For example, if motion is in progress, stopped, or if the amp enable is disabled (when the motor's disableAction is configured for command equals

actual), the controller will calculate a new command position every sample. To prevent this problem, set the command position when the motion is in an IDLE state and the motor's disableAction is configured for no action. It may also indicate that the motor associated to this axis is configured to mode MEIMotorDisableActionCMD_EQ_ACT and the motor is disabled. When the motor is disabled and in the MEIMotorDisableActionCMD_EQ_ACT mode, the command position is continually set to the actual position. To set the command position, enable the motor or take the motor out of the MEIMotorDisableActionCMD_EQ_ACT mode. Alternatively, setting the origin for the motor can often perform an equivalent result in this situation, as the command position will be set to the actual position the next sample.

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

CAN

MEICanMessageFIRST
First message in list of messages, placeholder only.

MEICanMessageCAN_INVALID
The can network number is out of range. This message code is returned by meiCanCreate(...) if the network number is less than zero or greater than or equal to MEICanNetworkMAX.

MEICanMessageFIRMWARE_INVALID
The CAN firmware is not valid. This message code is returned by meiCanCreate(...) if the CAN hardware bootloader detects no firmware has been loaded or the firmware signature is not recognized. To correct this problem, download valid firmware with meiCanFirmwareDownload(...).

MEICanMessageFIRMWARE_VERSION
The CAN firmware version does not match the software version. This message code is returned by meiCanCreate(...), meiCanFirmwareDownload(...), or meiCanFirmwareUpload(...) if the CAN firmware version is not compatible with the MPI library. To correct this problem, download the proper firmware version with meiCanFirmwareDownload(...).

MEICanMessageNOT_INITIALIZED
The CAN firmware did not initialize. This message code is returned by meiCanCreate(...) if the controller did not copy the configuration structure from flash to memory after power-on or controller reset. To correct this problem, verify the controller firmware is correct and the controller hardware is operating properly.

MEICanMessageIO_NOT_SUPPORTED
The CAN node does not support the specified I/O. This message code is returned by CAN methods

that read/write to a digital or analog input/output that is out of range. To prevent this problem, specify a supported I/O bit.

MEICanMessageFILE_FORMAT_ERROR
The CAN firmware file format has an error. This message code is returned by meiCanFirmwareDownload(...) if the specified file has an error in its internal headers. This indicates a corrupted file. To correct this problem, use the original CAN firmware file or reinstall the software distribution.

MEICanMessageUSER_ABORT
The CAN firmware loading was aborted. This message code is returned by meiCanFirmwareDownload(...) or meiCanFirmwareUpload(...) when the firmware loading is aborted by the user via the callback function. This message code is returned for application notification. It is not an error.

MEICanMessageCOMMAND_PROTOCOL
The CAN command failed due to a protocol error. This message code is returned by CAN methods that do not get a valid response from a CAN node. To correct this problem, check your CAN nodes for proper operation.

MEICanMessageINTERFACE_NOT_FOUND
The CAN interface is not available. This message code is returned by meiCanCreate(...) if the specified controller does not support a CAN network interface. To correct this problem, use a controller that has a CAN interface.

MEICanMessageNODE_DEAD
The CAN node does not respond. This message code is returned by CAN methods that read/write from a CAN node and the node fails the health check. This message code indicates a node hardware or network connection problem. To correct this problem, verify the node operation and network connections.

MEICanMessageSDO_TIMEOUT
The CAN command failed due to a timeout. This message code is returned by CAN methods that do not get a response from a CAN node within the timeout period. To correct this problem, check your CAN nodes for proper operation.

MEICanMessageSDO_ABORT
The CAN command failed due to a user abort. This message code is returned by CAN methods when an SDO transaction is aborted.

MEICanMessageSDO_PROTOCOL
The CAN command failed due to an SDO protocol error. This message code is returned by CAN methods when an SDO transaction fails because the node did not conform to the CANOpen protocol.

MEICanMessageTX_OVERFLOW
The controller's transmit buffer overflowed. This message code is returned by CAN methods that failed to transmit a message due to an internal memory buffer overflow.

MEICanMessageRTR_TX_OVERFLOW

The controller's transmit buffer overflowed. This message code is returned by CAN methods that failed to transmit a message due to an internal memory buffer overflow.

MEICanMessageRX_BUFFER_EMPTY
The controller's receive buffer is empty. This message code is returned by CAN methods that expected to get a response from a CAN node, but the controller's receive buffer was empty.

MEICanMessageBUS_OFF
The CAN network bus is in the off state. This message code is returned by CAN methods that are not able to use the CAN network because the bus is off. To correct this problem, verify the node operation and network connections.

MEICanMessageSIGNATURE_INVALID
When initialising the CAN system, some tests are performed to make sure that the CAN processor is returning a valid signature value. If an unexpected signature is returned, this error message is returned. A probable cause for this error is that the bootloader is invalid. To correct this problem, you will need to return the controller to MEI to fix the bootloader.

Capture

MPICaptureMessageFIRST
First message in list of messages, placeholder only.

MPICaptureMessageMOTOR_INVALID
The capture motor number is not valid. This message code is returned by mpiCaptureConfigSet(...) if the captureMotorNumber does not have node hardware or the value is MPICaptureNOT_MAPPED.

MPICaptureMessageCAPTURE_TYPE_INVALID
The capture type is not valid. This message code is returned by mpiCaptureConfigSet(...) if the type is not one of the values defined by the enum MPICaptureType.

MPICaptureMessageCAPTURE_INVALID
The capture number is out of range. This message code is returned by mpiCaptureCreate(...) if the capture number is less than zero or greater than or equal to MEIXmpMaxCapturesPerMotor.

MPICaptureMessageENCODER_INVALID
The encoder index is out of range. This message code is returned by mpiCaptureCreate(...) if the encoder index is less than MPIMotorEncoderFIRST or greater than or equal to MPIMotorEncoderLAST. See MPIMotorEncoder.

MEICaptureMessageINVALID_EDGE
The encoder edge trigger type is not valid. This message code is returned by mpiCaptureConfigSet (...) if the encoder capture edge type is not a member of the MPICaptureEdge enumeration.

MEICaptureMessageGLOBAL_CONFIG_ERR
The global trigger configuration is not valid. This message code is returned by mpiCaptureConfigSet (...) if the capture's trigger source is set to global and the capture's global trigger is enabled simultaneously. To correct this problem, either set the capture's trigger source to global or enable the capture's global trigger (not both).

MEICaptureMessageGLOBAL_ALREADY_ENABLED
The global trigger is already enabled. This message code is returned by mpiCaptureConfigSet(...) if a global trigger is already enabled on another capture on the same node. Only one global trigger enable is allowed per node. To prevent this problem, do not enable a second global trigger on a single node.

MEICaptureMessageCAPTURE_NOT_ENABLED
The capture object's hardware resource is not available. This message code is returned by mpiCaptureCreate(...) if the node hardware for the specified motor and encoder is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor and capture objects. A capture object cannot be created if there is no mapped hardware to support it. To correct this problem, verify that all expected nodes were found. Use meiSynqNetInfo(...) and meiSqNodeInfo(...) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEICaptureMessageCAPTURE_STATE_INVALID
This value is returned by mpiCaptureStatus(...) when the communication between the controller and the capture logic on the node fails resulting in an invalid capture state. See MPICaptureState.

MEICaptureMessageNOT_MAPPED
The capture object's hardware resource is not available. This message code is returned by mpiCaptureCreate(...) if the node hardware for the specified motor and encoder is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor and capture objects. A capture object cannot be created if there is no mapped hardware to support it. To correct this problem, verify that all expected nodes were found. Use meiSynqNetInfo(...) and meiSqNodeInfo(...) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEICaptureMessageUNSUPPORTED_PRIMARY
The capture hardware does not support the primary encoder. This message code is returned by mpiCaptureCreate(...) if the node hardware's primary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageUNSUPPORTED_SECONDARY
The capture hardware does not support the secondary encoder. This message code is returned by mpiCaptureCreate(...) if the node hardware's secondary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageSECONDARY_INDEX_INVALID
This message is returned from MPICaptureConfigSet(...) when the secondary encoder's index is specified as a trigger source in conjunction with other capture sources.

MEICaptureMessageCAPTURE_ARMED
The Capture resource being configured is already armed and cannot be reconfigured until it is disabled or triggered. See mpiCaptureConfigSet(...).

Client

MEIClientMessageFIRST
First message in list of messages, placeholder only.

MEIClientMessageCLIENT_INVALID
Not supported.

MEIClientMessageMETHOD_INVALID
The client method is out of range. This message code is returned by meiClientMethod(...) if the method number is not a member of the MEIRemoteMethod enumeration.

MEIClientMessageHEADER_INVALID
The client packet header is not valid. This message code is returned by meiClientMethod(...) if the packet header is corrupted. This indicates a problem with the message transmission. Check your network hardware.

Command

MPICommandMessageFIRST
First message in list of messages, placeholder only.

MPICommandMessageCOMMAND_INVALID
Not supported.

MPICommandMessageTYPE_INVALID
The command type is not valid. This message code is returned by mpiCommandCreate(...) if the command type is not a member of the MPICommandType enumeration.

MPICommandMessagePARAM_INVALID
Not supported.

Compensator

MPICompensatorMessageFIRST
First message in list of messages, placeholder only.

MPICompensatorMessageCOMPENSATOR_INVALID
The compensator number is not valid. This message code is returned by mpiCompensatorCreate(...) if the compensator number is less than 0 or greater than MPIControlMAX_COMPENSATORS.

MPICompensatorMessageNOT_CONFIGURED
MPI Compensator object must be configured before calling mpiCompensatorTableGet/ Set or mpiCompensatorInfo(...).

MPICompensatorMessageNOT_ENABLED
The compensator is not available on the controller. This message code is returned by mpiCompensatorValidate(…) if the compensator number is not within the range of enabled compensators on the controller. To correct the problem, use MPIControlConfig to configure the compensatorCount to be greater than the required compensator number.

MPICompensatorMessageAXIS_NOT_ENABLED
The axis is not available on the controller. This message coder is returned by mpiCompensatorConfigSet(...) if the axisOutNumber or any of the inputAxis[n].axisNumbers are not within the range of enabled axes on the controller. To correct the problem, use MPIControlConfig to configure the axisCount to be greater than the required axis number.

MPICompensatorMessageTABLE_SIZE_ERROR
The host compenstation table will not fit within the controller's configured compensation table. See Determining Required Compensator Table Size.

MPICompensatorMessagePOSITION_DELTA_INVALID
The positionDelta is either out of range or is not a multiple of the range. This message code is returned by mpiCompensatorConfigSet(…). To correct the problem, check the valid range values for MPICompensatorInputAxis.

MPICompensatorMessageDIMENSION_NOT_SUPPORTED
The dimensionCount is out of range. This message code is returned by mpiCompensatorConfigSet (…). To correct the problem, check the valid range values for MPICompensatorConfig.

Control

MPIControlMessageFIRST
First message in list of messages, placeholder only.

MPIControlMessageLIBRARY_VERSION
The MPI Library does not match the application. This message code is returned by mpiControlInit(...) if the MPI's library (DLL) version does not match the MPI header files that were compiled with the application. To correct this problem, the application must be recompiled using the same MPI software installation version that the application uses at run-time.

MPIControlMessageADDRESS_INVALID
The controller address is not valid. This message code is returned by mpiControlInit(...) if the controller address is not within a valid memory range. mpiControlInit(...) only requires memory addresses for certain operating systems. To correct this problem, verify the controller memory address.

MPIControlMessageCONTROL_INVALID
Not supported.

MPIControlMessageCONTROL_NUMBER_INVALID
The controller number is out of range. This message code is returned by mpiControlInit(...) if the controller number is less than zero or greater than or equal to MaxBoards(8).

MPIControlMessageTYPE_INVALID
The controller type is not valid. This message code is returned by mpiControlInit(...) if the controller type is not a member of the MPIControlType enumeration.

MPIControlMessageINTERRUPTS_DISABLED
The controller interrupt is disabled. This message code is returned by mpiControlInterruptWait(...) if the controller's interrupt is not enabled. This prevents an application from waiting for an interrupt that will never be generated. To correct this problem, enable controller interrupts with mpiControlInterruptEnable(...) before waiting for an interrupt.

MPIControlMessageEXTERNAL_MEMORY_OVERFLOW
The controller's external memory will overflow. This message code is returned by mpiControlConfigSet(...) if the dynamic memory allocation exceeds the external memory available on the controller. To correct the problem, reduce the number/size of control configuration resources or use a controller model with a larger static memory component.

MPIControlMessageADC_COUNT_INVALID
The ADC count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of ADCs is greater than MEIXmpMAX_ADCs.

MPIControlMessageAXIS_COUNT_INVALID
The axis count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of axes is greater than MEIXmpMAX_Axes.

MPIControlMessageAXIS_FRAME_COUNT_INVALID
This message is returned from mpiControlConfigSet(...) if the value for MPIControlConfig. axisFrameCount is not a power of two or if axisFrameCount is less than MPIControlMIN_AXIS_FRAME_COUNT.

MPIControlMessageAXIS_FRAME_COUNT_INVALID
This message is returned from mpiControlConfigSet() if the value for MPIControlConfig. axisFrameCount is not a power of two or if axisFrameCount is less than MPIControlMIN_AXIS_FRAME_COUNT.

MPIControlMessageCAPTURE_COUNT_INVALID
The capture count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of captures is greater than MEIXmpMAX_Captures.

MPIControlMessageCOMPARE_COUNT_INVALID
The compare count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of compares is greater than MEIXmpMAX_Compares.

MPIControlMessageCMDDAC_COUNT_INVALID ------
The command DAC count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of command DACs is greater than MEIXmpMAX_DACs.

MPIControlMessageAUXDAC_COUNT_INVALID ----
The auxiliary DAC count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of auxiliary DACs is greater than MEIXmpMAX_DACs.

MPIControlMessageFILTER_COUNT_INVALID
The filter count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of filters is greater than MEIXmpMAX_Filters.

MPIControlMessageMOTION_COUNT_INVALID
The motion count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of motions is greater than MEIXmpMAX_MSs.

MPIControlMessageMOTOR_COUNT_INVALID
The motor count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of motors is greater than MEIXmpMAX_Motors.

MPIControlMessageSAMPLE_RATE_TO_LOW
The controller sample rate is too small. This message code is returned by mpiControlConfigSet(…) if the sample rate is less than MPIControlMIN_SAMPLE_RATE (1kHz). SynqNet does not support cyclic data rates below 1kHz. The controller's sample rate specifies the SynqNet cyclic rate.

MPIControlMessageSAMPLE_RATE_TO_HIGH
The controller sample rate is too big. This message code is returned by mpiControlConfigSet(…) if the sample rate is greater than MPIControlMAX_SAMPLE_RATE (100kHz).

MPIControlMessageRECORDER_COUNT_INVALID
The recorder count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of recorders is greater than MEIXmpMAX_Recorders.

MPIControlMessageCOMPENSATOR_COUNT_INVALID
The compensator count is not valid. This message code is returned by mpiControlConfigSet(...) if the number of compensators is greater than MPIControlMAX_COMPENSATORS.

MPIControlMessageAXIS_RUNNING
Attempting to configure the control object while axes are running. It is recommended that all configuration of the control object occur prior to commanding motion.

MPIControlMessageRECORDER_RUNNING
Attempting to configure the control object while a recorder is running. It is recommended that all configuration of the control object occur prior to operation of any recorder objects.

MPIControlMessagePACK_ALIGNMENT
The application was compiled with a packing alignment that is different from the MPI library. For Windows, use the MSVC “/Zp8” compiler option or surround the MPI header files with the following #pragma pack statements:

```
#pragma pack(push, 8)
#include "stdmpi.h"
#include "stdmei.h"
#include "apputil.h"
#pragma pack(pop)
```

For Linux, use the “-malign-double” compiler option.

MEIControlMessageFIRMWARE_INVALID
The controller firmware is not valid. This message code is returned by mpiControlInit(...) if the MPI library does not recognize the controller signature. After power-up or reset, the controller loads the firmware from flash memory. When the firmware executes, it writes a signature value into external memory. If mpiControlInit(...) does not recognize the signature, then the firmware did not execute properly. To correct this problem, download firmware and verify the controller hardware is working properly.

MEIControlMessageFIRMWARE_VERSION_NONE
The controller firmware version is zero. This message code is returned by control methods do not find a firmware version. This indicates the firmware did not execute at controller power-up or reset. To correct this problem, download firmware and verify the controller hardware is working properly.

MEIControlMessageFIRMWARE_VERSION
The controller firmware version does not match the software version. This message code is returned by control methods if the firmware version is not compatible with the MPI library. To correct this problem, either download compatible firmware or install a compatible MPI run-time library.

MEIControlMessageFPGA_SOCKETS
The maximum number of FPGA socket types has been exceeded. This message code is returned by meiFlashMemoryFromFile(...) if the controller has more FPGA types than the controller has flash memory space to support them.

MEIControlMessageBAD_FPGA_SOCKET_DATA
Not supported.

MEIControlMessageNO_FPGA_SOCKET

The FPGA socket type does not exist. This message code is returned by meiFlashMemoryFromFile(...) if the controller does not support the FPGA type that was specified in the FPGA image file. To correct this problem, use a different FPGA image that is compatible with the controller.

MEIControlMessageINVALID_BLOCK_COUNT
Not supported.

MEIControlMessageSYNQNET_OBJECTS
Not supported.

MEIControlMessageSYNQNET_STATE
The controller's SynqNet state is not expected. This message code is returned by mpiControlInit(...), mpiControlReset(...) and mpiControlConfigSet(...) if the SynqNet network initialization fails to reach the SYNQ state. To correct this problem, check your node hardware and network connections.

MEIControlMessageIO_BIT_INVALID
The controller I/O bit is not valid. This message code is returned by meiControlIoGet(...) or meiControlIoSet(...) if the controller I/O bit is not a member of the MEIControlIoBit enumeration.

DriveMap

MEIDriveMapMessageFIRST
First message in list of messages, placeholder only.

MEIDriveMapMessageMAP_FILE_OPEN_ERROR
The drive map file could not be opened. Either it doesn't exist or the user does not have permission to access it.

MEIDriveMapMessageMAP_FILE_FORMAT_INVALID
The contents of the drive map file does not match the format of a drive map file.

MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP
The node type, specified by the "nodeName" parameter, was not found in the drive map file.

MEIDriveMapMessageVERSION_NOT_FOUND_IN_MAP
The drive firmware version, specified by the "firmwareVersion" parameter, was not found in the drive map file.

MEIDriveMapMessageDRIVE_PARAM_READ_ONLY
An attempt was made to modify a read-only drive parameter.

Element

MEIElementMessageFIRST
First message in list of messages, placeholder only.

MEIElementMessageELEMENT_INVALID
The element is not valid. This message code is used internally to guarantee an element module and element number are valid.

Event

MPIEventMessageFIRST
First message in list of messages, placeholder only.

MPIEventMessageEVENT_INVALID
The event type is not valid. This message code is returned by mpiEventStatusSet(...) if the event type is not a member of the MPIEventType or MEIEventType enumerations.

EventMgr

MPIEventMgrMessageFIRST
First message in list of messages, placeholder only.

MPIEventMgrMessageEVENTMGR_INVALID
Not supported.

Filter

MPIFilterMessageFIRST
First message in list of messages, placeholder only.

MPIFilterMessageFILTER_INVALID
The filter number is out of range. This message code is returned by mpiFilterCreate(...) if the filter number is less than zero or greater than or equal to MEIXmpMAX_Filters.

MPIFilterMessageINVALID_ALGORITHM
The filter algorithm is not valid. This message code is returned by mpiFilterIntegratorReset(...) if the filter algorithm is not a member of the MEIXmpAlgorithm enumeration (does not support integrators). This problem occurs if the filter type is set to user or an unknown type with mpiFilterConfigSet(...).

MPIFilterMessageINVALID_DRATE
The filter derivative rate is not valid. This message code is returned by mpiFilterConfigSet(...) if the filter derivative rate is less than 0 or greater than 7.
NOTE: The derivative rate for all gain tables must be in the range [0,7], not just the derivative rate for the current gain table.

MPIFilterMessageCONVERSION_DIV_BY_0
Returned when meiFilterPostfilterGet(...) or meiFilterPostfilterSectionGet(...) cannot convert digital coefficients to analog coefficients. When this error occurs, the offending section(s) will report its type as MEIFilterTypeUNKNOWN and will not contain any analog data.

MPIFilterMessageSECTION_NOT_ENABLED
Returned when meiFilterPostfilterGet(...) or meiFilterPostfilterSectionGet(...) attempt to read postfilter data when no postfilter sections are enabled.

MPIFilterMessageINVALID_FILTER_FORM
Returned when meiFilterPostfilterGet(...) or meiFilterPostfilterSectionGet(...) cannot interpret the current postfilter's form (when the form is something other than NONE, IIR, or BIQUAD).

MPIFilterMessageINVALID_KA1
The filter Ka1 value is not valid. This message code is returned by mpiFilterConfigSet(...) if the filter Ka1 value is less than 0 or greater than or equal to 1 for the PIV algorithm.

Flash

MEIFlashMessageFIRST
First message in list of messages, placeholder only.

MEIFlashMessageFLASH_INVALID
The flash object is not valid. This message code is returned by meiFlashMemoryFromFile(...), meiFlashMemoryFromFileType(...), meiFlashMemoryToFile(...), or meiFlashMemoryVerify(...) if the flash object's memory cache has not been allocated. The flash memory cache is allocated in mpiFlashCreate(...). To prevent this problem, create flash objects with mpiFlashCreate(...) and check the return value.

MEIFlashMessageFLASH_VERIFY_ERROR
The flash memory read failed. This message code is returned by meiFlashMemoryVerify(...) if the read from flash memory does not match the read from the file. This indicates the specified flash file is different from the flash memory. This message code can also be returned from meiFlashMemoryFromFile(...), since it calls meiFlashMemoryVerify(...). To correct this problem, first check that the specified file is the one you intended. If the specified file is correct, use meiFlashMemoryFromFile(...) to download the file.

MEIFlashMessageFLASH_WRITE_ERROR

The flash memory write failed. This message code is returned by meiFlashMemoryErase(...), meiFlashMemoryFromFile(...), or meiFlashMemoryFromFileType(...) if the flash memory write fails. This indicates a problem in the flash memory component or subsystem.

MEIFlashMessagePATH
The flash file path is too long. This message code is returned by meiFlashMemoryFromFile(...) if the file path is longer than MEIFlashFileMaxPathChars. To correct this problem, use a shorter path.

MEIFlashMessageNETWORK_TOPOLOGY_ERROR
The network topology has not been saved to flash. This message code is returned by object flash config set methods if the network topology has not been previously saved to flash. This message code serves as a warning of a potential safety problem. Saving object configurations to flash will cause those values to be sent to nodes during the next network intialization, even if the flash configurations are not compatible with the network topology. If the network topology is saved to flash, then the controller will automatically verify the topology before sending object configuration values to the nodes. To prevent this message code return, use meiSynqNetFlashTopologySave(...) to save the network topology to the controller's flash. For more information, please see Save/Clear Topology to Flash.

List

MEIListMessageFIRST
First message in list of messages, placeholder only.

MEIListMessageLIST_INVALID
List invalid.

MEIListMessageELEMENT_NOT_FOUND
The element is not a member of the list. This message code is returned by meiListElement(...) if the index to the list is greater than or equal to the list count. This message code is used by the MPI internally to check object list handling.

MEIListMessageELEMENT_INVALID
The element is not valid. This message code is returned by meiListInsertNow(...) or meiListRemoveNow(...) if the element cannot be inserted or removed from the list. This message code is used by the MPI internally to check object list handling.

Map

MEIMapMessageFIRST
First message in list of messages, placeholder only.

MEIMapMessageNAME_INVALID
The map name is not valid. This message code is returned by meiMapNameToParams(...) if the name is not a member of a map.

MEIMapMessageNAME_NOT_FOUND
The map name is not found. This message code is returned by meiMapCreate(...), meiMapNameToParams(...), or meiMapNameParse(...) if the name is not found in the map.

MEIMapMessageINDEX_INVALID
The map index is not valid. This message code is returned by meiMapNameToParams(...) if the map index is less than 0 or greater than or equal to the maximum instance count.

MEIMapMessageFILE_INVALID
The map file is not valid. This message code is returned by meiMapNameParse(...) if the map file cannot be parsed.

Motion

MPIMotionMessageFIRST
First message in list of messages, placeholder only.

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.

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 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 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 or increase the number of points in the frame buffer.

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.

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.

Motor

MPIMotorMessageFIRST
First message in list of messages, placeholder only.

MPIMotorMessageMOTOR_INVALID
The motor number is out of range. This message code is returned by mpiMotorCreate(...) if the motor number is less than zero or greater than or equal to MEIXmpMAX_Motors.

MPIMotorMessageTYPE_INVALID
The motor type is not valid. This message code is returned by mpiMotorConfigGet/ Set if the motor type is not a value defined in the enumeration MPIMotorType. To avoid this error, use one of the defined motor types in MPIMotorType.

MPIMotorMessageSTEPPER_NA
The hardware does not support the Stepper motor type. This message code is returned by mpiMotorConfigSet(...) if the motor type is configured for MPIMotorTypeSTEPPER when the node hardware does not support steppers. To correct the problem, do not select the stepper motor type.

MEIMotorMessageMOTOR_NOT_ENABLED
The motor number is not active in the controller. This message code is returned by mpiMotorEventConfigGet/ Set if the specified motor is not enabled in the controller. To correct the problem, use mpiControlConfigSet(...) to enable the motor object, by setting the motorCount to greater than the motor number. For example, to enable motor 0 to 3, set motorCount to 4.

MEIMotorMessageSECONDARY_ENCODER_NA
The motor's secondary encoder is not available. This message code is returned by mpiMotorConfigSet

(...) or mpiMotorEventConfigSet(...) if the encoder fault trigger is configured for a secondary encoder when the hardware does not support a secondary encoder. To correct the problem, do not select the secondary encoder when configuring the encoder fault conditions.

MEIMotorMessageHARDWARE_NOT_FOUND
The motor object's hardware resource is not available. This message code is returned by mpiMotorConfigGet/ Set. if the node hardware for the motor is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor objects. An application should not configure a motor object if there is no mapped hardware to receive the service commands. To correct this problem, verify that all expected nodes were found. Use meiSynqNetInfo(...) and meiSqNodeInfo(...) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEIMotorMessageDISABLE_ACTION_INVALID
The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(...) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessageSTEPPER_INVALID
The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(...) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessagePULSE_WIDTH_INVALID
The motor stepper pulse width is not valid. This message code is returned by mpiMotorConfigSet(...) if the pulseWidth is out of range. To correct the problem, specify the the pulse width value between 100 nanoseconds and 1 millisecond.

MEIMotorMessageFEEDBACK_REVERSAL_NA
The feedback reversal is not applicable for the feedback type specified. This message code is returned by mpiMotorConfigSet(...) if the feedback type is not MEIMotorEncoderTypeQUAD_AB and the encoderPhase is set to TRUE. To correct the problem, set encoderPhase to FALSE. Some drives may support feedback reversal via drive parameters. Please consult the drive manufacturer's documentation for details.

MEIMotorMessageFILTER_DISABLE_NA
The feedback filter disable is not applicable for the feedback type specified. This message code is returned by mpiMotorConfigSet(...) if the feedback type is not MEIMotorEncoderTypeQUAD_AB and the filterDisable is set to TRUE. To correct the problem, set filterDisable to FALSE.

MEIMotorMessagePHASE_FINDING_FAILED
The drive failed to complete the phase finding procedure. This message code is returned by

meiMotorPhaseFindStart(…) if the drive failed to successfully initailized commutation. To determine the cause of the failure, check the fault and warning codes from the drive with meiMotorAmpFault(…) and meiMotorAmpWarning(…).

MEIMotorMessageDEMAND_MODE_UNSUPPORTED
The motor does not support the specified demand mode. This message is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the demand mode is configured for a mode that the node/drive does not support. To correct this problem, use the default demand mode or specify a different demand mode.

MEIMotorMessageDEMAND_MODE_NOT_SET
The demand mode cannot be changed while the amplifier is enabled. This message is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the demand mode is changed while the motor's amplifier is enabled. To avoid this error message, disable the motor's amp enable with mpiMotorAmpEnableSet(…) before changing the demand mode.

Notify

MPINotifyMessageFIRST
First message in list of messages, placeholder only.

MPINotifyMessageNOTIFY_INVALID
The notify object is not valid. This message code is returned by a notify method if the notify object handle is not valid. This problem can be caused by failed mpiNotifyCreate(...). To prevent this problem, check your notify objects after creation by using mpiNotifyValidate(...).

MPINotifyMessageWAIT_IN_PROGRESS
The notify object is waiting for an event. This message code is returned by mpiNotifyEventWait(...) if the notify object is already waiting for an event in another thread. To prevent this problem, make sure a thread does not share notify objects with other threads.

Packet

MEIPacketMessageFIRST
First message in list of messages, placeholder only.

MEIPacketMessagePACKET_INVALID
Not supported.

MEIPacketMessageADDRESS_INVALID
The server socket address is not valid. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the server name is not specified or the server address is not valid. This

message code comes from a lower level routine, meiPacketClient(...). To correct the problem, make sure the server is operational and the server name or address is valid.

MEIPacketMessageCOMM_ERROR
The socket communication failed. This message code is returned by a packet method that fails to send or receive data correctly. This usually indicates a hardware problem. To correct this problem, make sure your network hardware is functioning properly and all network connections are reliable.

MEIPacketMessageCONNECTION_CLOSED
The socket communication connection is closed. This message code is returned by meiPacketRecv (...) or meiPacketSend(...) if zero bytes of data are sent or received. This usually means the socket communication was not opened or was closed. To correct this problem, make sure meiPacketCreate (...) was successful and that another thread did not close the socket connection.

MEIPacketMessageRECV_ERROR
Not supported. See MEIPacketMessageCOMM_ERROR.

MEIPacketMessageSEND_ERROR
Not supported. See MEIPacketMessageCOMM_ERROR.

Path

MPIPathMessageFIRST
First message in list of messages, placeholder only.

MPIPathMessagePATH_INVALID
The path object is not valid. This message code is returned by a path method if the path object handle is not valid. This problem can be caused by a failed mpiPathCreate(...). To prevent this problem, check your path objects after creation by using mpiPathValidate(...).

MPIPathMessageILLEGAL_DIMENSION
The path dimensions are not valid. This message code is returned by mpiPathParamsSet(...) or mpiPathMotionParamsGenerate(...) if the path dimension is less than one or greater than or equal to MPIPathPointDIMENSION_MAX. Also, this message code is returned if specific path element types have dimension restrictions. For example, the ARC type is limited to 2 dimensions and the ARC_END_POINT type is limited to 3 dimensions. To correct this problem, select an appropriate dimension for the path element type.

MPIPathMessageILLEGAL_ELEMENT
The path element type is not valid. This message code is returned by mpiPathAppend(...) if the specified path element type is not a member of the MPIPathElementType enumeration.

MPIPathMessageARC_ILLEGAL_DIMENSION
Illegal arc dimension (too many axes in arc). Arc dimension is out of range. This message code occurs when the specified dimension is out of range. An arc element dimension must be 2.

MPIPathMessageHELIX_ILLEGAL_DIMENSION
Not supported.

MPIPathMessageILLEGAL_RADIUS
The path element arc radius is not valid. This message code is returned by mpiPathAppend(...) if the ARC element radius is less than or equal to zero. To correct this problem, set the arc radius to a value greater than zero.

MPIPathMessagePATH_TOO_LONG
The path length is not valid. This message code is returned by mpiPathMotionParamsGenerate(...) if the path length is greater than MAX_PATH_POINTS_MAX. To correct the problem, specify a path with fewer points than MAX_PATH_POINTS_MAX.

MPIPathMessageILLEGAL_VELOCITY
The path element velocity is not valid. This message code is returned by mpiPathAppend(...) if the specified velocity is less than or equal to zero. To correct this problem, set the element velocity to a value greater than zero.

MPIPathMessageILLEGAL_ACCELERATION
The path element velocity is not valid. This message code is returned by mpiPathAppend(...) if the specified velocity is less than or equal to zero. To correct this problem, set the element velocity to a value greater than zero.

MPIPathMessageILLEGAL_TIMESLICE
The path element time slice is not valid. This message code is returned by mpiPathAppend(...) if the specified time slice is less than or equal to zero. To correct this problem, set the element time slice to a value greater than zero.

MPIPathMessageINVALID_BLENDING
The path element blending is not valid. This message code is returned by mpiPathMotionParamsGenerate(...) if the element blending is set to TRUE and the motion type does not support blending. To correct this problem, either set the element blending to FALSE or select a different motion type.

Platform

MEIPlatformMessageFIRST
First message in list of messages, placeholder only.

MEIPlatformMessagePLATFORM_INVALID
The platform object is not valid. This message code is returned by a platform method if the platform object handle is not valid. Most applications do not use the platform module. The MPI library uses the platform module internally. If an application needs a platform handle, use meiControlPlatform(...). Do NOT create your own platform object with meiPlatformCreate(...).

MEIPlatformMessageDEVICE_INVALID
The platform device driver is not valid. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the platform device handle is not valid. This message code comes from the lower level routines, meiPlatformInit(...) or meiPlatformDeviceClose(...). To correct the problem, make sure the device driver is installed and operational.

MEIPlatformMessageDEVICE_ERROR
The platform device failed. This message code is returned by the platform methods that fail to access a controller via a device driver. It occurs if the specified board type is not a member of the MEIPlatformBoardType enumeration. It also occurs if the device driver fails to read/write controller memory or there is an interrupt handling failure. To correct the problem, verify the platform has support for your controller and the device drive is installed and operational. Check for any resource conflicts (memory range, I/O port range, and interrupts) with other devices.

MEIPlatformMessageDEVICE_MAP_ERROR
The platform device memory mapping failed. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the controller memory could not be mapped to the operating system's memory space. To correct this problem, verify there are no memory resource conflicts. Also, make sure the host computer and operating system have enough free memory for the controller (XMP-Series requires 8MB).

MEIPlatformMessageCOPY64_FAILURE
The 64-bit read failed. This message is returned by meiPlatformMemoryGet64(…), mpiAxisCommandPositionGet(…), or mpiAxisActualPositionGet(…), if the 64-bit position data cannot be read atomically. Internally, the MPI uses an algorithm to construct the 64-bit position data via multiple 32-bit reads. If the 64-bit position value is not valid after multiple attempts, this error will be returned. If your application experiences this error message, if possible, use the equivalent 32-bit methods, mpiAxisCommandPositionGet32(…), mpiAxisActualPositionGet32(…), or contact MEI.

Probe

MPIProbeMessageFIRST
First message in list of messages, placeholder only.

MPIProbeMessageNODE_INVALID
The SynqNet node number is not available on the network. This message code is returned by MPI methods that fail a service command transaction due to the specified node number being greater than or equal to the total number of nodes discovered during network initialization. To correct this problem, check the discovered node count with meiSynqNetInfo(...). If the node count is not what you expected, check your network wiring, node condition, and re-initialize the network with mpiControlReset(...).

MPIProbeMessagePROBE_TYPE_INVALID
The Probe data type is not valid. This message is returned by mpiProbeConfigSet(...) if the specified

Probe data type is not one of the enumerated values.

MPIProbeMessagePROBE_INVALID
The Probe is not valid. This message is returned by mpiProbeConfigGet(...) or mpiProbeConfigSet(...) if the specified Probe params or configurations are out of range. To correct this problem, check your Probe params and config structures. The probe number specified was invalid. The likely cause of this is that the node you are referring to does not support the probe feature. Please refer to the node feature table to see if your node supports probe.

Recorder

MPIRecorderMessageFIRST
First message in list of messages, placeholder only.

MPIRecorderMessageNO_RECORDERS_AVAIL
This message code is returned when there are no more recorders available to create. If the mpiRecorderCreate(control, -1) is called to create the next available recorder object and either all 32 recorders have already been allocated or there are not enough recorders enabled on the controller, this error message will be returned. Check to make sure that enough recorders are enabled on the controller. To enable more recorders, use mpiControlConfigSet(...).

MPIRecorderMessageNOT_ENABLED
This message code is returned by a recorder method when a specific recorder is not enabled on the controller. When a recorder number is explicitly requested when creating a recorder such as mpiRecorderCreate(control, 3) and the corresponding recorder is not enabled on the controller, this message will be returned.

MPIRecorderMessageRECORDER_INVALID
The recorder object is not valid. This message code is returned by a recorder method if the recorder object handle is not valid. This problem can be caused by a failed mpiRecorderCreate(...). To prevent this problem, check your recorder objects after creation by using mpiRecorderValidate(...).

MPIRecorderMessageRUNNING
This message is returned by mpiRecorderConfigSet(...) and mpiRecorderConfigGet(...) when an application attempts to configure a recorder that is currently running.

MPIRecorderMessageSTARTED
The data recorder is already running. This message code is returned by mpiRecorderStart(...) if the data recorder has already been started. If this is a problem, call mpiRecorderStop(...) to stop the data recorder or wait for the recorder to collect the number of specified records and stop.

MPIRecorderMessageSTOPPED
The data recorder is not running. This message code is returned by mpiRecorderStop(...) if the data recorder has already been stopped. If this is a problem, call mpiRecorderStart(...) to start the data

recorder.

MPIRecorderMessageNOT_CONFIGURED
The data recorder has not been configured. This message code is returned by mpiRecorderRecordGet(...) if the data address count has not been configured. To correct this problem, configure the data recorder with mpiRecorderConfigSet(...).

Sequence

MPISequenceMessageFIRST
First message in list of messages, placeholder only.

MPISequenceMessageSEQUENCE_INVALID
The sequence number is out of range. This message code is returned by mpiSequenceCreate(...) if the sequence number is less than zero or greater than or equal to MEIXmpMAX_PSs. This message code is also returned if the specified sequence number is not active in the controller. To correct this problem, use mpiControlConfigSet(...) to enable the sequence object, by setting the sequenceCount to greater than the sequence number. For example, to enable sequence 0 to 3, set sequenceCount to 4. This message code is returned by mpiSequenceLoad(...) if the sequence buffer size and the sequence page size are not equal. This indicates an internal MPI Library problem.

MPISequenceMessageCOMMAND_COUNT
The sequence command count is out of range. This message code is returned by mpiSequenceStart(...), or meiSequenceCompile(...) if the sequence command count is less than or equal to zero. To correct this problem, set the command count to a value greater than zero.

MPISequenceMessageCOMMAND_NOT_FOUND
The sequence command is not found. This message code is returned by mpiSequenceLoad(...), mpiSequenceStart(...), or meiSequenceCompile(...) if the specified command is not a member of the sequence. To correct this problem, specify a command that is a member of the sequence.

MPISequenceMessageSTARTED
The program sequencer is already running. This message code is returned by mpiSequenceResume(...), mpiSequenceStart(...), or mpiSequenceStep(...) if the program sequencer has already been started. If this is a problem, call mpiSequenceStop(...) to stop the program sequencer or monitor the sequence status and wait for the state to equal STOPPED.

MPISequenceMessageSTOPPED
The program sequencer is not running. This message code is returned by mpiSequenceStop(...) if the program sequencer has already been stopped. If this is a problem, call mpiSequenceStart(...) to start the program sequencer.

Server

MEIServerMessageFIRST
First message in list of messages, placeholder only.

MEIServerMessageSERVER_INVALID
Not supported.

MEIServerMessageMETHOD_INVALID
The server method is out of range. This message code is returned by meiServerMethod(...) if the method number is not a member of the MEIRemoteMethod enumeration.

MEIServerMessageHEADER_INVALID
The server packet header is not valid. This message code is returned by meiServerMethod(...) if the packet header is corrupted. This indicates a problem with the message transmission. Check your network hardware. This message code can also be returned when there is an MPI version mismatch between the server.exe utility and the connecting client application.

SqNode

MEISqNodeMessageFIRST
First message in list of messages, placeholder only.

MEISqNodeMessageINVALID
The SqNode type is out of range. This message code is returned by SynqNet node methods if the node type is not a member of the SQNodeLibNodeType enumeration.

MEISqNodeMessageNODE_INVALID
The node number is not available on the network. This message code is returned by MPI methods that fail a service command transaction due to the specified node number is greater than or equal to the total number of nodes discovered during network initialization. To correct this problem, check the discovered node count with meiSynqNetInfo(...). If the node count is not what you expected check your network wiring, node condition, and re-initialize the network with mpiControlReset(...).

MEISqNodeMessageSTATE_ERROR
Some methods can only be executed in SYNQ state. This message will be returned when the network is not in the expected network state (i.e ASYNQ state).

MEISqNodeMessageCONFIG_NETWORK_MISMATCH
The type of map file specified in meiSqNodeDriveMapParamFileSet(...) does not match the type of drive found on the network.

MEISqNodeMessageNOT_IN_CONFIG_FILE
The parameter name or number specified in meiSqNodeDriveMapParamFileSet(...) was not valid for

the specified drive.

MEISqNodeMessageBOOT_FILE_NOT_FOUND
The boot file "kollmorgen_ember.a00" was not found. When downloading drive images to Kollmorgen CD, DASA, and PicoDAD drives, a boot file is downloaded to the drive prior to the actual drive image. This boot file needs to be located in the same directory as the drive's image file that is provided for download.

MEISqNodeMessageMAP_CONFIG_MISMATCH
The parameter name or number specified in meiSqNodeDriveMapParamFileSet(...) was not valid for the specified drive.

MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID
A file with an incorrect format was used in meiSqNodeDriveMapParamFileSet(...).

MEISqNodeMessageRESPONSE_TIMEOUT
The drive/node did not respond to the service command issued in a reasonable amount of time.

MEISqNodeMessageREADY_TIMEOUT
The node/drive did not complete the hand shaking for the service command.

MEISqNodeMessageSRVC_ERROR
There was an error in carrying out the service command issued.

MEISqNodeMessageSRVC_UNSUPPORTED
The service command issued is either not supported or recognized by the drive.

MEISqNodeMessageSRVC_CHANNEL_INVALID
Invalid service channel specified. See MEISqNodeCmdHeader.

MEISqNodeMessageCMD_NOT_SUPPORTED
The service command is not supported by the node.

MEISqNodeMessageDISCOVERY_FAILURE
Unable to discover node rescources.

MEISqNodeMessageDISPATCH_ERROR
Is the default error code returned when a node specific routine has failed. Check the node FPGA version to verify whether or not it is correct.

MEISqNodeMessageINIT_FAILURE
A node specific initialization routine was unable to successfully complete its routine. Verify that the node FPGA is the default version for your MPI version. See MEISqNodeInfoFpga.

MEISqNodeMessageINTERFACE_ERROR1
This is an outdated node, which does not support the current discovery routine.

MEISqNodeMessageFILE_NODE_MISMATCH
Node type does not match the file provided for download.

MEISqNodeMessageFILE_INVALID
The file provided for download was not found or was corrupted.

MEISqNodeMessageINVALID_HEADER
The header information in the download image is invalid. Please verify firmware file to be correct and retry download. If firmware file is correct please contact firmware manufacturer.

MEISqNodeMessageDOWNLOAD_FAIL
Node firmware download failed. Verify that the firmware file is correct and retry the download.
NOTE: A network reset my be required.

MEISqNodeMessageVERIFY_FAIL
The node FPGA firmware does not match the FPGA image file.

MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED
The downloading of the node firmware (FPGA) image is not supported for this node.

MEISqNodeMessageVERIFY_NOT_SUPPORTED
The Node specified for verification does not support the upload of the FPGA image. Therefore, the image cannot be verified.

MEISqNodeMessageBOOT_ROM_INVALID
The SqNode Boot Rom identification or version is not recognized by the MPI.

MEISqNodeMessageINVALID_TABLE
Invalid resource table in node module. This is a fatal error within the MPI. Please verify MPI and node FPGA versions to be correct and then contact MEI's Technical Support.

MEISqNodeMessageINVALID_STR_LEN
An attempt to write information to the node has failed due to an invalid string length.

MEISqNodeMessageFEEDBACK_MAP_INVALID
Returned from meiSqNodeConfigSet(...) when the given secondary encoder (n) is not mappable to the motor on the node specified by MEISqNodeFeedbackSecondary[n].motorIndex. See MEISqNodeFeedbackSecondary.

MEISqNodeMessageNODE_FAILURE
An attempt was made to access a SynqNet node that has a node failure event active.

MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT
When initializing the SynqNet network one of the node requires a SynqNet packet larger that what can be supported.

MEISqNodeMessageIO_MODULE_INCOMPATIBILITY

Two modules attached to a SQID node are incompatible. This error message code is returned when initializing a SQID node. Different types of I/O module may be incompatible and will not work on the same SQID node.

MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED
The EEPROM on one of the modules attached to a SQID node has not been programmed.

MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED
The maximum number of I/O that can be supported by a SQID node has been exceeded.

MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED
When initializing an SQIO node, checks are performed to confirm that the node actually supports the correct number of I/O. This error is returned when one of these tests fails. An error will occur when the length of at least one of the inter module (SPI) buses did not match the length calculated from data held in the module EEPROMs. This error message can be returned by MPI functions that reset the SynqNet Network such as, mpiControlReset(...), mpiControlInit(...), meiSynqNetInit(...). This error can ONLY be generated by SQIO nodes. This fault can also be caused by a poor electrical connection between the SQID and the I/O modules. If the modules are firmly connected and the error persists then you will need to contact your supplier of the I/O Modules to correct the hardware.

MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED
During the initialization of the modules attached to a SQID node, the maximum current that can be drawn from the inter module 3.3V bus exceeds the allowable current.

MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED
During the initialization of the modules attached to a SQID node, the maximum current that can be drawn from the inter module 5V bus exceeds the allowable current.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT
An error was encountered while initalizing the slice node.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MANY_SLICES
Slice I/O nodes can only support 32 slices attached to the network adapter. This error is returned when more than 32 slices are detected.

MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT
The slice node did not initalize within the expected time.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH
A slice is attached to the node that is not supplied by MEI. You can only use slices supplied by MEI.

MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH
When attempting to restore the slice parameters to the attached slices of a Slice I/O node, the arrangement of slices did not match the expected arrangement. If you wish to clear the previous slice parameters, you can stop this error by calling meiSqNodeSegmentParamClear(...).

MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR
When attempting to access data on Slice I/O, a communication fault was detected. The message

received from the slice was badly formed, errors include CRC and missing start/stop bits.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR
When attempting to access data on Slice I/O, a communication fault was detected. The slice returned too many characters when responding to this request.

MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE
When attempting to access data on Slice I/O, a communication fault was detected. The slice returned an error code.

MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE
When attempting to access data on Slice I/O, a communication fault was detected. An unknown fault was detected when accessing this slice.

MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE
When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED
When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the action requested on this data. For example, a read or write.

MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE
When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE
When attempting to access data on Slice I/O, a communication fault was detected. The slice is already in the requested mode.

MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT
When attempting to access data on Slice I/O, a communication fault was detected. The slice can not perform the requested action in its current state.

MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE
When attempting to access data on Slice I/O, a communication fault was detected. You cannot modify the data on this slice.

MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA
When attempting to access data on Slice I/O, a communication fault was detected. Not enough data was supplied to the slice for this operation.

MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED
When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA
When attempting to access data on Slice I/O, a communication fault was detected. Too much data

was supplied to the slice for this operation.

MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST
When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER
When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the specified parameter.

MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE
When attempting to access data on Slice I/O, a communication fault was detected. The slice failed during a store operation.

MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE
When attempting to access data on Slice I/O, a communication fault was detected. The error code from the slice was not recognized.

MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT
When attempting to access data on Slice I/O, a communication fault was detected. The operation on the slice exceeded the timeout threshold.

MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT
When attempting to access data on Slice I/O, a communication fault was detected. The response from the slice was not formatted correctly.

MEISqNodeMessageIO_SLICE_SERVICE_EEPROM_FORMAT
The data held on the EEPROM on the network adaptor was not formatted correctly.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_IO
When attempting to access data on Slice I/O, a communication fault was detected. You should remove some slices to stop this error message from being generated.

MEISqNodeMessagePARAM_READ_ONLY
The drive parameter that the user is attempting to set is read only.

MEISqNodeMessagePARAM_LOCKED
The drive parameter that the user is attempting to set is not accessible. SelSFDParam must be set to 0, otherwise the SFD motor parameters will be used.

MEISqNodeMessageMONITOR_INDEX
Drive does not support the configuring of Monitors through indexing.

MEISqNodeMessageMONITOR_ADDRESS
Drive does not support the configuring of Monitors through addressing.

SynqNet

MEISynqNetMessageFIRST
First message in list of messages, placeholder only.

MEISynqNetMessageSYNQNET_INVALID
The SynqNet number is out of range. This message code is returned by meiSynqNetCreate(...) if the SynqNet network number is less than zero or greater than or equal to MEIXmpMaxSynqNets.

MEISynqNetMessageMAX_NODE_ERROR
The SynqNet node number is out of range. This message code is returned by a SynqNet method if the specified node number is greater than or equal to MEIXmpMaxSynqNetBlocks.

MEISynqNetMessageSTATE_ERROR
The SynqNet network state is not valid. This message code is returned by any method that initializes a SynqNet network if the controller's network state is not a member of the MEIXmpSynqNetState or MEIXmpSynqNetInternalState enumeration. The most commonly used methods that initialize the SynqNet network are: mpiControlInit(...), mpiControlReset(...) and meiSynqNetInit(...). This message code indicates a failure in the controller's initialization sequence. To correct this problem, call mpiControlReset(...).

MEISynqNetMessageCOMM_ERROR
The SynqNet network communication failed. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_CRC
The SynqNet network communication failed due to excessive CRC errors. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX
The SynqNet network communication failed due to a Rincon receive error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_LEN
The SynqNet network communication failed due to a Rincon receive length error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_FIFO
The SynqNet network communication failed due to a Rincon receive buffer error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This

message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE
The SynqNet network communication failed due to a Rincon receive dribble error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_CRC
The SynqNet network communication failed due to a Rincon receive CRC error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageINTERFACE_NOT_FOUND
The controller does not support a SynqNet interface. This message code is returned by meiSynqNetValidate(...), meiSynqNetFlashTopologySave(...), meiSynqNetFlashTopologyClear(...), mpiControlInit(...), and mpiControlReset(...) if the controller does not have a SynqNet hardware interface. To correct this problem, use a controller that supports SynqNet.

MEISynqNetMessageTOPOLOGY_MISMATCH
The network topology does not match the expected network topology. This message code is returned by mpiControlInit(...) or meiSynqNetInit(...) if the discovered network topology does not match the controller's expected network topology (stored in dynamic memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into it's dynamic memory. This message code indicates the number of nodes, the node order, or types of nodes have changed since the initial network initialization. To correct this problem, either change the network topology to the original configuration or clear the controller's memory with mpiControlReset(...).

MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH
The network topology does not match the expected network topology. This message code is returned by mpiControlInit(...) or meiSynqNetInit(...) if the discovered network topology does not match the controller's expected network topology (stored in flash memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into it's dynamic memory. Later, when meiSynqNetFlashTopologySave(...) is called the topology information is stored into flash memory. This message indicates the number of nodes, the node order, or types of nodes have changed since the topology information was stored in flash. To correct this problem, either change the network topology to the saved configuration or clear the controller's flash topology with meiSynqNetFlashTopologyClear(...).

MEISynqNetMessageRESET_REQ_TIMEOUT
The network reset request packet exceeded the timeout. This message code is returned by mpiControlInit(...), mpiControlReset(...), or meiSynqNetInit(...) if the reset request packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageRESET_ACK_TIMEOUT
The network reset complete packet exceeded the timeout. This message code is returned by mpiControlInit(...), mpiControlReset(...), or meiSynqNetInit(...) if the reset complete packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageDISCOVERY_TIMEOUT
The network topology discovery exceeded the timeout. This message code is returned by mpiControlInit(...), mpiControlReset(...), or meiSynqNetInit(...) if the controller failed to discover the network topology in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageNO_NODES_FOUND
The controller did not find network nodes. This message code is returned by mpiControlInit(...), mpiControlReset(...), or meiSynqNetInit(...) if the controller failed to discover any nodes during network initialization. This message code indicates the first node has failed or the network connection from the controller to the first node is faulty. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageNO_TIMING_DATA_AVAIL
The corresponding SynqNet node module does not contain timing data, so the network cannot be initialized. Contact MEI or drive manufacturer for an updated node module.

MEISynqNetMessageINCOMPLETE_MOTOR
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a motor configuration that is missing feedback or command fields.

MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW
The controller's SynqNet buffer size was exceeded. This message code is returned by mpiControlInit(...), mpiControlReset(...) if the controller's SynqNet buffer could not be allocated due to overflow. This message code indicates the controller does not have enough memory to initialize the network topology. To correct the problem, either reduce the network nodes or use a different controller model.

MEISynqNetMessageINVALID_MOTOR_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more motors than supported by the node.

MEISynqNetMessageINVALID_AUX_ENC_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more secondary encoders than supported by the node.

MEISynqNetMessageINVALID_COMMAND_CFG
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a different command configuration than is supported by the node.

MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains

an illegal number of pulse engines.

MEISynqNetMessageINVALID_ENCODER_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more encoders than supported by the node.

MEISynqNetMessageINVALID_CAPTURE_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger capture count than is supported by the node.

MEISynqNetMessageINVALID_COMPARE_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger compare count than is supported by the node.

MEISynqNetMessageINVALID_INPUT_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger ioInput count than is supported by the node.

MEISynqNetMessageINVALID_OUTPUT_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger ioOutput count than is supported by the node.

MEISynqNetMessageINVALID_MONITOR_CFG
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains more monitor fields than the node can support.

MEISynqNetMessageINVALID_ANALOG_IN_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger analogIn count than is supported by the node.

MEISynqNetMessageINVALID_DIGITAL_IN_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger digitalIn count than is supported by the node.

MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger digitalOut count than is supported by the node.

MEISynqNetMessageINVALID_ANALOG_OUT_COUNT
This message is returned by meiSynqNetPacketConfigSet(...), when the packet configuration contains a larger analogOut count than is supported by the node.

MEISynqNetMessageLINK_NOT_IDLE
This message is returned by meiSynqNetIdleCableStatus(...) when the cable number supplied is not the idle link. The status check can only be run on an idle cable. See meiSynqNetIdleCableListGet(...).

MEISynqNetMessageIDLE_LINK_UNKNOWN
This message is returned by meiSynqNetIdleCableListGet(...) when an idle cable number in a ring topology cannot be determined. This is due to one or more failed nodes on a ring topology. Be sure to check the status of the nodes.

MEISynqNetMessageRING_ONLY
This message is returned by meiSynqNetIdleCableListGet(...) and meiSynqNetIdleCableStatus(...) because idle cables exist on ring topologies only. It is also returned by meiSynqNetConfigSet(...) when attempting to enable SynqNet recovery mode, which is only supported for ring topologies.

MEISynqNetMessageRECOVERING
The network is recovering from a fault condition. This message is returned by meiSynqNetIdleCableStatus(...) during network fault recovery, because the network traffic is being redirected around the faulted cable and a new idle cable is in the process of being assigned. If you receive this message, wait for recovery to complete and then check the identity of the idle cable with meiSynqNetIdleCableListGet(...).

MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED
This message is returned when a cable on the network is too long.

MEISynqNetMessageCABLE_LENGTH_TIMEOUT
The cable length discovery failed to complete due to a timeout. This message is returned by mpiControlInit(…), meiSynqNetInit(…), or meiSynqNetNodeRestart(…) during SynqNet network initialization if the node fails to respond to a cable length measurement packet. This message indicates a hardware problem. To correct the problem, check your cabling and node hardware.

MEISynqNetMessageCABLE_LENGTH_MISMATCH
This message is returned at network initialization when topology has been saved to flash and a cable length is detected that lies outside the MEISynqNetConfig.cableLength[n].minimum and maximum. Check cable length (if necessary) and save new cable length values to flash. See MEISynqNetConfig.

MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL
The nominal cable length is too long.

MEISynqNetMessageCABLE_LENGTH_INVALID_MIN
The minimum cable length is too long or exceeds nominal value.

MEISynqNetMessageCABLE_LENGTH_INVALID_MAX
The maximum cable length is too long or is less than nominal value.

MEISynqNetMessageNODE_FPGA_VERSION
The node FGPA version is out of date. Use sqNodeFlash.exe to load most current FPGA version to node.

MEISynqNetMessageMAX_MOTOR_ERROR
The number of motors on the network exceeds the number set by MEISynqNetMaxMOTORS.

MEISynqNetMessagePLL_ERROR
The node PLL is unable to lock with drive.

MEISynqNetMessageNODE_INIT_FAIL
A node specific initialization routine was unable to complete successfully. Verify node FPGA is the default version for your MPI version. See MEISqNodeInfoFpga.

MEISynqNetMessageTOPOLOGY_CLEAR
An attempt to clear the saved network topology was made when no network topology has been saved.

MEISynqNetMessageTOPOLOGY_SAVED
An attempt to save network topology was made when the network topology has already been saved. Clear the network topology before attempting to save another topology to flash.

MEISynqNetMessageTOPOLOGY_AMPS_ENABLED
An meiSynqNetFlashTopologySave(...) and meiSynqNetFlashTopologyClear(...) routine has been called while one or more motor amplifiers are enabled. Disables all motor amplifiers (mpiMotorAmpEnableSet(...)) before calling these low-level routines. To avoid this error message, disable the motor(s) amp enable before calling meiSynqNetFlashTopologySave/Clear.

MEISynqNetMessageNODE_MAC_VERSION
A node on the network has an incompatible MAC version. Use sqNodeFlash.exe to load the correct node .sff file that came with your MPI release.

MEISynqNetMessageADC_SAMPLE_FAILURE
A check to verify that all analog inputs can be sampled during the given controller sample rate has failed. Either reduce the number of analog inputs on your network (see meiSynqNetPacketConfigGet(...) and meiSynqNetPacketConfigSet(...)) or decrease your controller sample rate (see mpiControlConfigGet(...) and mpiControlConfigSet(...)).

MEISynqNetMessageSCHEDULING_ERROR
This is a generic return value to indicate that a problem has occurred while calculating the network scheduling. For more specific information about the error, run your application with timing assignment tracing turned on, using the trace mask:

0x00100000	SynqNet: Display timing assignments

MEISynqNetMessageINVALID_PROBE_CFG
This message is returned by meiSynqNetPacketConfigSet(...), when the MEISynqNetPacketCfgProbe. count contains a larger count than is supported by the node. See MEISynqNetPacketCfgProbe.

MEISynqNetMessageINVALID_PROBE_DEPTH
This message is returned by meiSynqNetPacketConfigSet(...), when the MEISynqNetPacketCfgProbe. depth contains a larger count than is supported by the node. See MEISynqNetPacketCfgProbe.

MEISynqNetMessageSAMPLE_PERIOD_NOT_MULTIPLE
The controller update period is not an integer multiple of all the SynqNet nodes drive periods.

This message is returned from mpiControlConfigSet(…) if the specified controller sample rate is not an integer multiple of the node drive periods. For more details, see Sample Rate for more details.

MEISynqNetMessageNODE_LATENCY_EXCEEDED
The node latency exceeded the maximum control latency limit. This message is returned by mpiControlInit(…) or meiSynqNetInit(…) during SynqNet initialization if the node latency exceeds the maximum control latency configured by mpiSqNodeConfigSet(…). To avoid this problem, either leave the maximum control latency configuration at the default value or increase the maximum control latency limit.

MEISynqNetMessageHOT_RESTART_FAIL_NOT_SYNQ_STATE
The SynqNet network is currently not in SYNQ mode. This message is returned by meiSynqNetNodeRestart(…) if the SynqNet network is not in the SYNQ state when the nodes are restarted. To avoid this problem, make sure that the network state is in SYNQ mode before attempting to restart any shutdown nodes. Use meiSynqNetInit(…) to initialize a SynqNet network to the SYNQ state.

MEISynqNetMessageHOT_RESTART_FAIL_RECOVERING
The SynqNet network is not ready for a node restart. The network is currently in a recovery state (see MEISynqNetState) or the network has recovery mode disabled. Wait for the network to finish recovering or enable recovery mode. This message is returned by meiSynqNetNodeRestart(…) if the SynqNet network fault recovery is in process. To avoid this problem, wait for fault recovery to complete before restarting a node.

MEISynqNetMessageHOT_RESTART_FAIL_TEST_PACKET
The SynqNet node restart did not complete due to a failed restart packet. This message is returned by meiSynqNetNodeRestart(…) if the restart packet failed. The problem is that the test packet is in use, so hot restart cannot take place. Test packets should not be in use when using Hot Restart because restart takes over the test packet's memory locations.

MEISynqNetMessageHOT_RESTART_FAIL_ADDRESS_ASSIGNMENT
The SynqNet node restart did not complete due to failed node identification. The node failed to receive its address assignment and most likely does not support the SynqNet Hot Restart feature. This message is returned by meiSynqNetNodeRestart(…), if the node did not respond to a service command to read the node’s identification data. This message indicates a hardware problem. To correct the problem, check your cabling and node hardware.

MEISynqNetMessageHOT_RESTART_NOT_ALL_NODES_RESTARTED
One or more SynqNet nodes were not restarted. This message is returned by meiSynqNetNodeRestart(…), if any node could not be restarted. This message is for information purposes. Possible causes for nodes not being restarted are if they are not connected to the network or do not have power.

MEISynqNetMessageSHUTDOWN_NODES_NONCONSECUTIVE
The specified nodes to shutdown are not sequential. This message is returned by meiSynqNetShutdown(…) if the shutdown nodeMask does not specify sequential nodes. This message protects the user from shutting down nodes without specifying them in the nodeMask. For

example, suppose there are three nodes in a string (0, 1, 2) and the shutdown nodeMask = 0x5 (node 0 and 2). Node 1 would fail when nodes 0 and 2 are shutdown.

MEISynqNetMessageSHUTDOWN_NODES_STRANDED
The specified nodes to shutdown will cause additional nodes to fail. This message is returned by meiSynqNetShutdown(…) if the shutdown nodeMask will cause additional nodes not specified in the nodeMask to fail. This message protects the user from shutting down nodes without specifying them in the nodeMask. For example, suppose there are two nodes in a string (0, 1) and the shutdown nodeMask = 0x1 (node 0). Node 1 would fail when node 0 is shutdown.

MEISynqNetMessageSHUTDOWN_RECOVERY_DISABLED
The SynqNet network recovery mode is disabled. This message is returned by meiSynqNetShutdown(…) if the network is a ring topology and the recovery mode is disabled. To avoid this message, set the SynqNet network recovery configuration to Single-Shot or Auto-Arm.