

# Select an MPI/MEI Object

**MPI Library:** 03.02.00

**Last Modified :** 10Feb2005

[Axis](#)

[Filter](#)

[Platform](#)

[CAN](#)

[Flash](#)

[Probe](#)

[Capture](#)

[Global](#)\*

[Recorder](#)

[Command](#)

[Message](#)\*

[Sequence](#)

[Compensator](#)

[Motion](#)

[Service](#)

[Control](#)

[Motor](#)

[SqNode](#)

[DriveMap](#)

[Notify](#)

[SynqNet](#)

[Event](#)

[Object](#)\*

[Trace](#)\*

[EventMask](#)

[Path](#)

[Xmp](#)

[EventMgr](#)

# Axis Objects

## Introduction

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

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

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

## Methods

### Create, Delete, Validate Methods

|                                        |                      |
|----------------------------------------|----------------------|
| <a href="#"><u>mpiAxisCreate</u></a>   | Create Axis object   |
| <a href="#"><u>mpiAxisDelete</u></a>   | Delete Axis object   |
| <a href="#"><u>mpiAxisValidate</u></a> | Validate Axis object |

### Configuration and Information Methods

|                                                  |                               |
|--------------------------------------------------|-------------------------------|
| <a href="#"><u>mpiAxisActualPositionGet</u></a>  | Get actual position           |
| <a href="#"><u>mpiAxisActualPositionSet</u></a>  | Set actual position           |
| <a href="#"><u>mpiAxisActualVelocity</u></a>     | Get actual velocity           |
| <a href="#"><u>mpiAxisConfigGet</u></a>          | Get Axis configuration        |
| <a href="#"><u>mpiAxisConfigSet</u></a>          | Set Axis configuration        |
| <a href="#"><u>mpiAxisCommandPositionGet</u></a> | Get command position          |
| <a href="#"><u>mpiAxisCommandPositionSet</u></a> | Set command position          |
| <a href="#"><u>mpiAxisFlashConfigGet</u></a>     | Get Axis flash config         |
| <a href="#"><u>mpiAxisFlashConfigSet</u></a>     | Set Axis flash config         |
| <a href="#"><u>mpiAxisOriginGet</u></a>          | Get Axis origin               |
| <a href="#"><u>mpiAxisOriginSet</u></a>          | Set Axis origin               |
| <a href="#"><u>mpiAxisPositionError</u></a>      | Get position error of an Axis |
| <a href="#"><u>mpiAxisStatus</u></a>             | Get Axis status               |

[mpiAxisTrajectory](#)

Get Axis trajectory

## Event Methods

[mpiAxisEventNotifyGet](#)

Get event mask

[mpiAxisEventNotifySet](#)

Set event mask

[mpiAxisEventReset](#)

## Memory Methods

[mpiAxisMemory](#)

Set Axis memory address

[mpiAxisMemoryGet](#)

Copy bytes of Axis memory to application memory

[mpiAxisMemorySet](#)

Copy bytes of application memory to Axis memory

## Relational Methods

[mpiAxisControl](#)

Return handle of Control associated with Axis

[mpiAxisFilterMapGet](#)

Get object map of Filters

[mpiAxisFilterMapSet](#)

Set object map of Filters

[mpiAxisMotorMapGet](#)

Get object map of Motors

[mpiAxisNumber](#)

Get index of Axis

## Data Types

[MPIAxisConfig](#) / [MEIAxisConfig](#)

[MPIAxisInPosition](#)

[MPIAxisMaster](#)

[MPIAxisMasterType](#)

[MPIAxisMessage](#)

# 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.

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <b>control</b> | a handle to Axis object.                                                                                         |
| <b>number</b>  | 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        |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to an Axis object                  |
| <b>MPIHandleVOID</b> | if the object could not be created |

## 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.

|             |                               |
|-------------|-------------------------------|
| <b>axis</b> | 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

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisDelete</i> successfully deletes an Axis object and invalidates its handle |
|---------------------|-------------------------------------------------------------------------------------|

## 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.

|             |                                             |
|-------------|---------------------------------------------|
| <b>axis</b> | a handle to the Axis object to be validated |
|-------------|---------------------------------------------|

### Return Values

|                     |                                        |
|---------------------|----------------------------------------|
| <b>MPIMessageOK</b> | if Axis is a handle to a valid object. |
|---------------------|----------------------------------------|

## See Also

[mpiAxisCreate](#) | [mpiAxisDelete](#)

# mpiAxisActualPositionGet

## Declaration

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

Required Header: stdmpi.h

## Description

|                |                                                               |
|----------------|---------------------------------------------------------------|
| <b>axis</b>    | a handle to an Axis object.                                   |
| <b>*actual</b> | a pointer to the Axis actual position returned by the method. |

| Return Values       |                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisActualPositionGet</i> successfully gets and writes the value of the actual postion of <i>axis</i> to the location. |

## See Also

[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***.

|               |                                                        |
|---------------|--------------------------------------------------------|
| <b>axis</b>   | a handle to an Axis object.                            |
| <b>actual</b> | a value to which the Axis actual position will be set. |

| Return Values       |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisActualPositionSet</i> successfully sets the value of the actual postion of an Axis to <b><i>actual</i></b> . |

## 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       |                                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisActualVelocity</i> succesfully gets and writes the value of the actual velocity of <i><b>axis</b></i> to the location. |

## See Also

# mpiAxisConfigGet

## Declaration

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

Required Header: stdmpi.h

## 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).

|                  |                                           |
|------------------|-------------------------------------------|
| <b>axis</b>      | a handle to an Axis object.               |
| <b>*config</b>   | pointer to the MPIAxisConfig structure    |
| <b>*external</b> | pointer to an external. See remarks below |

| Return Values       |                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisConfigGet</i> successfully gets the Axis configuration and writes it into the structure(s). |

## 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

## 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.

|                  |                                           |
|------------------|-------------------------------------------|
| <b>axis</b>      | a handle to an Axis object.               |
| <b>*config</b>   | pointer to the MPIAxisConfig structure    |
| <b>*external</b> | pointer to an external. See remarks below |

| Return Values       |                                                                   |
|---------------------|-------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisConfigSet</i> successfully sets the Axis configuration. |

## 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***.

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| <b>axis</b>     | a handle to an Axis object.                                   |
| <b>*command</b> | a pointer to the Axis command position returned by the method |

| Return Values       |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisCommandPositionGet</i> successfully gets the value of the command position of <i><b>axis</b></i> and puts it in the location. |

## See Also

[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***.

**AxisCommandPositionSet** 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.

|                |                                                        |
|----------------|--------------------------------------------------------|
| <b>axis</b>    | a handle to the Axis object                            |
| <b>command</b> | value to which the Actual command position will be set |

## Remarks

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

| Return Values                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                  | if <i>AxisCommandPositionSet</i> successfully sets the value of the command position of <b><i>axis</i></b> from <b><i>command</i></b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>MPIMessageARG_INVALID</b>         | if <b><i>command</i></b> lies outside the range of signed 32-bit integers: [-2147483648, 2147483648)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>MPIAxisMessageCOMMAND_NOT_SET</b> | <p>May 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.</p> <p>To set the command position, enable the motor or take the motor out of the MEIMotorDisableActionCMD_EQ_ACT mode.</p> <p>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.</p> |

## 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 *sexternal* (if *external* is not NULL).

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

|                  |                                            |
|------------------|--------------------------------------------|
| <b>axis</b>      | a handle to the Axis object                |
| <b>*flash</b>    |                                            |
| <b>*config</b>   | pointer to an MPIAxisConfig structure      |
| <b>*external</b> | 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       |                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisEventNotifySet</i> successfully requests host notification of the event(s) that are specified by <i>eventMask</i> and generated by <i>motion</i> |

## 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).

|                  |                                            |
|------------------|--------------------------------------------|
| <b>axis</b>      | a handle to the Axis object                |
| <b>*flash</b>    |                                            |
| <b>*config</b>   | pointer to an MPIAxisConfig structure      |
| <b>*external</b> | 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       |                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisFlashConfigSet</i> successfully sets the Axis flash configuration using data from the structure(s) |

## See Also

[MEIFlash](#) | [mpiAxisFlashConfigGet](#) | [MEIAxisConfig](#)

# mpiAxisOriginGet

## Declaration

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

**Required Header:** stdmpi.h

## Description

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

|                |                                                    |
|----------------|----------------------------------------------------|
| <b>axis</b>    | a handle to the Axis object.                       |
| <b>*origin</b> | pointer to the Origin value returned by the method |

### Return Values

#### MPIMessageOK

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

## See Also

[mpiAxisOriginSet](#) | [Using the Origin Variable](#) | [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*.

|               |                                            |
|---------------|--------------------------------------------|
| <b>axis</b>   | a handle to the Axis object.               |
| <b>origin</b> | value to which the Axis Origin will be set |

| Return Values       |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisOriginSet</i> successfully sets the origin of an Axis to <i>origin</i> |

## 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).

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <b>axis</b>   | a handle to the Axis object                                 |
| <b>*error</b> | a pointer to the Axis position error returned by the method |

| Return Values       |                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisPositionError</i> successfully gets and writes the value of the position error into <b><i>*error</i></b> |

## 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).

|                  |                                                  |
|------------------|--------------------------------------------------|
| <b>axis</b>      | a handle to the Axis object                      |
| <b>*status</b>   | pointer to MPIStatus structure.                  |
| <b>*external</b> | pointer to an implementation-specific structure. |

## Remarks

***external*** should always be set to NULL.

| Return Values                |                                                                                            |
|------------------------------|--------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>AxisStatus</i> successfully gets the Axis status and writes it into the structure(s) |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL.                                                      |

## 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.

|                    |                                        |
|--------------------|----------------------------------------|
| <b>axis</b>        | a handle to the Axis object.           |
| <b>*trajectory</b> | 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       |                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisTrajectory</i> successfully gets the Axis trajectory and writes it into the structure |

## See Also

[mpiMotionStart](#) | [mpiMotionModify](#) | [MPITrajectory](#)

# mpiAxisEventNotifyGet

## Declaration

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

Required Header: stdmpi.h

## Description

**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).

|                   |                                           |
|-------------------|-------------------------------------------|
| <b>axis</b>       | a handle to the Axis object               |
| <b>*eventMask</b> | pointer to an MPIEventMask                |
| <b>*external</b>  | 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       |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisEventNotifyGet</i> successfully writes the event mask to the location(s) |

## See Also

[MEIEventNotifyData](#) | [MEIEventStatusInfo](#) | [mpiAxisEventNotifySet](#)

# mpiAxisEventNotifySet

## Declaration

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

Required Header: stdmpi.h

## Description

**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).

|                  |                             |
|------------------|-----------------------------|
| <b>axis</b>      | a handle to the Axis object |
| <b>eventMask</b> | pointer to an MPIEventMask  |
| <b>*external</b> | 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 <i>eventmask</i> with mpiEventMaskALL(eventMask)   |
| disable host notification of all events | configure <i>eventmask</i> with mpiEventMaskCLEAR(eventMask) |

**Return Values****MPIMessageOK**

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

**See Also**

[MEIEventNotifyData](#) | [MEIEventStatusInfo](#) | [MPIEventMask](#) | [MPIEventType](#) |  
[mpiEventMaskALL](#) | [mpiEventMaskCLEAR](#) | [mpiAxisEventNotifyGet](#) |  
[MEIEventNotifyData](#)



# mpiAxisEventNotifySet

## Declaration

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

Required Header: stdmpi.h

## Description

### Remarks

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

| To...                                   | Then...                                                             |
|-----------------------------------------|---------------------------------------------------------------------|
| enable host notification of all events  | configure <b><i>eventmask</i></b> with mpiEventMaskALL(eventMask)   |
| disable host notification of all events | configure <b><i>eventmask</i></b> with mpiEventMaskCLEAR(eventMask) |

| Return Values       |                                                                                                                                                                          |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisEventNotifySet</i> successfully requests host notification of the event(s) that are specified by <b><i>eventMask</i></b> and generated by <b><i>motion</i></b> |

## See Also

[MEIEventNotifyData](#) | [MEIEventStatusInfo](#) | [MPIEventType](#) | [mpiEventMaskALL](#) | [mpiEventMaskCLEAR](#) | [MPIEventMaskMOTION](#) | [MPIEventMaskAXIS](#) | [MPIEventMask](#)

# mpiAxisMemory

## Declaration

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

Required Header: stdmpi.h

## Description

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

|                 |                             |
|-----------------|-----------------------------|
| <b>axis</b>     | a handle to the Axis object |
| <b>**memory</b> |                             |

| Return Values       |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisMemory</i> successfully writes the Axis memory address to the contents of <i>memory</i> |

## See Also

[mpiAxisMemoryGet](#) | [mpiAxisMemorySet](#)

# mpiAxisMemoryGet

## Declaration

```
long mpiAxisMemoryGet(MPIAxis axis,
                      void      *dst,
                      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*).

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

| Return Values       |                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisMemoryGet</i> successfully writes the Axis memory address to the contents of memory |

## See Also

[mpiAxisMemory](#) | [mpiAxisMemorySet](#)

# mpiAxisMemorySet

## Declaration

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

Required Header: stdmpi.h

## Description

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

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

| Return Values       |                                                                                                     |
|---------------------|-----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisMemorySet</i> successfully copies <i>count</i> bytes of application memory to Axis memory |

## 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.

|             |                             |
|-------------|-----------------------------|
| <b>axis</b> | a handle to an Axis object. |
|-------------|-----------------------------|

### Return Values

|                      |                           |
|----------------------|---------------------------|
| <b>MPIHandleVOID</b> | if <i>axis</i> is invalid |
|----------------------|---------------------------|

## 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*.

|                   |                                                         |
|-------------------|---------------------------------------------------------|
| <b>axis</b>       | a handle to the Axis object                             |
| <b>*filterMap</b> | 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       |                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisFilterMapGet</i> successfully gets and writes the object map to the structure |

## See Also

[mpiAxisFilterMapSet](#)

# mpiAxisFilterMapSet

## Declaration

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

Required Header: stdmpi.h

## Description

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

|                  |                                            |
|------------------|--------------------------------------------|
| <b>axis</b>      | a handle to the Axis object                |
| <b>filterMap</b> | 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       |                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisFilterMapSet</i> successfully sets the Filters using data from the object map |

## See Also

[mpiAxisFilterMapGet](#) | [MPIObjectMap](#)

# mpiAxisMotorMapGet

## Declaration

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

Required Header: stdmpi.h

## Description

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

|                  |                                                        |
|------------------|--------------------------------------------------------|
| <b>axis</b>      | a handle to the Axis object.                           |
| <b>*motorMap</b> | 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       |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisMotorMapGet</i> successfully gets the object map and writes it into the structure |

## 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***.

|                |                             |
|----------------|-----------------------------|
| <b>axis</b>    | a handle to the Axis object |
| <b>*number</b> | pointer to the number       |

| Return Values       |                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>AxisNumber</i> successfully writes the index of Axis to the contents of <b><i>number</i></b> |

## See Also

# MPIAxisConfig / MEIAxisConfig

## Definition: MPIAxisConfig

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

## Description

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

## Definition: MEIAxisConfig

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

## Description

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

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

For a 1:1 ratio of encoder input to reported encoder input set:

Coeff = MEIXmpFRACTIONAL\_UNITY.

For 0.5:1 ratio, set:

Coeff = MEIXmpFRACTIONAL\_UNITY / 2.

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

### Filter

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

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

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

#### Example:

```
MEIXmpData      *firmware;
```

```
MEIXmpBufferData *bufferData;
```

```
mpiControlMemory(control,&firmware,&bufferData);
```

```
...
```

```
msgCHECK(mpiAxisConfigGet(axis, &axisConfig, &axisConfigXmp));
```

```
axisConfigXmp.Gear.Ptr = &bufferData->PreFilter[0].Output;
```

```
msgCHECK(mpiAxisConfigSet(axis, &axisConfig, &axisConfigXmp));
```

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

## Sample Code

```
/* Change axis encoder scaling.  
   limit scale to +/- 2.0 */  
void axisScale(MPIAxis axis, float scale)  
{  
    MPIAxisConfig config;  
    MEIAxisConfig xmpConfig;  
  
    mpiAxisConfigGet(axis, &config, &xmpConfig);  
    xmpConfig.APos[0].Coeff = (long)(scale * MEIXmpFRACTIONAL_UNITY);  
    mpiAxisConfigSet(axis, &config, &xmpConfig);  
}
```

## See Also

[mpiAxisConfigGet](#) | [mpiAxisConfigSet](#) | [MPIAxisInPosition](#)

# MPIAxisInPosition

## Definition

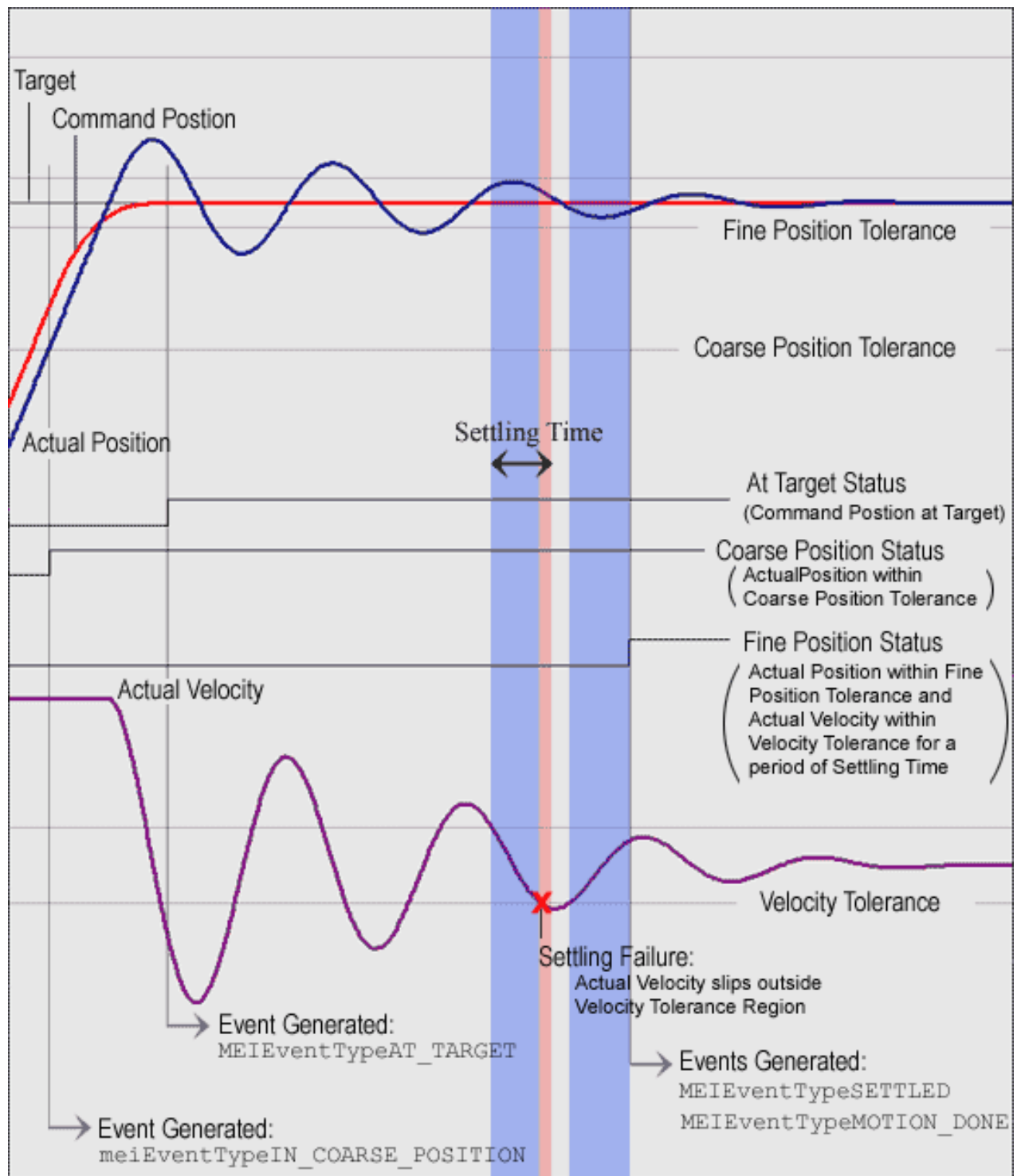
```
typedef struct MPIAxisInPosition {
    struct {
        float    positionFine;
        long     positionCoarse;
        float    velocity;
    } tolerance;
    float    settlingTime; /* seconds */
    long     settleOnStop;
    long     settleOnEstop;
    long     settleOnEstopCmdEqAct;
} MPIAxisInPosition;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

## Description

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>tolerance</b>      | Includes the following 3 elements that determine settling tolerances for an axis.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>positionFine</b>   | 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.                                                                                                                                                                                                        |
| <b>positionCoarse</b> | 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.                                                                                                                                                                                                                                                                                                                                                    |
| <b>velocity</b>       | 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: <ul style="list-style-type: none"> <li>• a position-based move is complete and an MPIEventTypeMOTION_DONE event is generated.</li> <li>• a velocity move is complete and an MPIEventTypeMOTION_AT_VELOCITY event is generated.</li> <li>• an axis is settled and an MPIEventTypeSETTLED event is generated.</li> </ul> |
| <b>settlingTime</b>   | Duration in seconds that an axis must satisfy the positionFine and/or velocity tolerance, before the respective status flag is set.                                                                                                                                                                                                                                                                                                                                                                                      |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>settleOnStop</b>          | <p>If TRUE, the controller will use settle on stop mode. If FALSE, the controller will not use the settle on stop mode.</p> <p>When in settleOnStop mode and a STOP event has occurred, the axis will stay in an MPIStateSTOPPING state until:</p> <ul style="list-style-type: none"> <li>• The settling criteria are satisfied AND.</li> <li>• The stop duration for the axis' Motion Supervisor has elapsed.</li> <li>• This state can be read with mpiAxisStatus(MPIAxis axis, MPIStatus *status, void *external).</li> </ul> <p>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.</p>                                                                                                                                                                                                                                                              |
| <b>settleOnEstop</b>         | <p>If TRUE, the controller will use settle on Estop mode. If FALSE, the controller will not use the settle on Estop mode.</p> <p>When in settleOnEstop mode and a ESTOP event has occurred, the axis will stay in an MPIStateSTOPPING_ERROR state until:</p> <ul style="list-style-type: none"> <li>• The settling criteria are satisfied AND.</li> <li>• The Estop duration for the axis' Motion Supervisor has elapsed.</li> <li>• This state can be read with mpiAxisStatus(MPIAxis axis, MPIStatus *status, void *external).</li> </ul> <p>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.</p>                                                                                                                                                                                                                                           |
| <b>settleOnEstopCmdEqAct</b> | <p>If TRUE, the controller will use settle on EstopCmdEqAct mode. If FALSE, the controller will not use the settle on EstopCmdEqAct mode.</p> <p><b>***settleOnEstopCmdEqAct mode is not recommended***</b></p> <p>SettleOnEstopCmdEqAct is an alternative to Estop mode. When this mode is enabled, the following things happen:</p> <ul style="list-style-type: none"> <li>• During normal motion, there is no difference.</li> <li>• 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.</li> <li>• After the Estop time elapses, the axis' motors will disable the</li> </ul> |

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 |

|                                |                                                                                                                                                                                                                            |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>                    | This field defines the type of master position source is being used.                                                                                                                                                       |
| <b>number</b>                  | the motor or axis number.                                                                                                                                                                                                  |
| <b>address</b>                 | The controller address to be used as the master position.                                                                                                                                                                  |
| <b>encoderFaultMotorNumber</b> | 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 <a href="#">Master Encoder Faults</a> . |

## See Also

[MPIAxisMasterType](#)

# MPIAxisMasterType

## Definition

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

## Description

**MPIAxisMasterType** specifies the type of master position source used with cam motions. See also [MPIAxisMaster](#).

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

## See Also

[MPIAxisMaster](#)

# MPIAxisMessage

## 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

# 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, and meiCanFlashConfigGet, meiCanFlashConfigSet, meiCanFlashNodeConfigGet, and meiCanFlashNodeConfigSet 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](#) | [XMP Overview](#) |

## Methods

### Create, Delete, Validate Methods

|                                |                     |
|--------------------------------|---------------------|
| <a href="#">meiCanCreate</a>   | Create Can object   |
| <a href="#">meiCanDelete</a>   | Delete Can object   |
| <a href="#">meiCanValidate</a> | Validate Can object |

### Configuration and Information Methods

[meiCanConfigGet](#)

Get Can's configuration

[meiCanConfigSet](#)

Set Can's configuration

[meiCanFlashConfigGet](#)

Get Can's flash configuration

[meiCanFlashConfigSet](#)

Set Can's flash configuration

[meiCanStatus](#)

Get status of the CAN controller.

[meiCanVersion](#)

Returns the version information about a controller's CAN system.

[meiCanCommand](#)

Get Can's flash configuration

[meiCanNodeConfigGet](#)

Return a copy of the current configuration

[meiCanNodeConfigSet](#)

Update the current configuration that the specified CAN node is using.

[meiCanNodeFlashConfigGet](#)

Get the flash configuration of the Can node

[meiCanNodeFlashConfigSet](#)

Set the flash configuration of the Can node

[meiCanNodeStatus](#)

Get the instantaneous state of the local CAN interface.

[meiCanNodeInfo](#)

Return the node information after the XMP finishes scanning the network.

## I/O Methods

[meiCanNodeAnalogInputGet](#)

Get current analog input

[meiCanNodeAnalogOutputGet](#)

Get current analog output

[meiCanNodeAnalogOutputSet](#)

Set current analog output

[meiCanNodeDigitalInputGet](#)

Get the current state of the digital input bit.

[meiCanNodeDigitalInputsGet](#)

Get the current state of ALL the digital input bits.

[meiCanNodeDigitalOutputGet](#)

Get the current state of the digital output bit.

[meiCanNodeDigitalOutputsGet](#)

Get the current state of all the digital output bits.

[meiCanNodeDigitalOutputSet](#)

changes the state of the digital output bit.

[meiCanNodeDigitalOutputsSet](#)

Changes the current state of all the digital output bits.

## Event Methods

[meiCanEventNotifyGet](#)

Get event mask of events for which host notification has been requested

[meiCanEventNotifySet](#)

Set event mask of events for which host notification will be requested

## Firmware Methods

[meiCanFirmwareDownload](#)

Downloads firmware to the Can controller

[meiCanFirmwareErase](#)

Erases firmware on the Can controller

[meiCanFirmwareUpload](#)

Uploads firmware from the Can controller

## Memory Methods

[meiCanMemory](#)

Get address to Can's memory

[meiCanMemoryGet](#)

Copy data from Can memory to application memory

[meiCanMemorySet](#)

Copy data from application memory to Recorder memory

## Relational Methods

[meiCanControl](#)

[MEICanNumber](#)

## Data Types

[MEICanBitRate](#)

[MEICanBusState](#)

[MEICanCallback](#)

[MEICanCommand](#)

[MEICanCommandType](#)

[MEICanConfig](#)

[MEICanDigitalIO](#)

[MEICanHealthType](#)

[MEICanMessage](#)

[MEICanNodeConfig](#)

[MEICanNodeInfo](#)

[MEICanNodeStatus](#)

[MEICanNodeType](#)

[MEICanNMTState](#)

[MEICanStatus](#)

[MEICanTransmissionType](#)

[MEICanVersion](#)

## Constants

[MEICanNetworkMAX](#)

# 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.

|                |                                                                                                                                                                        |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>control</b> | a handle to the controller object that contains the CAN object.                                                                                                        |
| <b>number</b>  | 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

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <b>handle</b>        | Handle to the CAN object created or MPIHandleVOID. |
| <b>MPIHandleVOID</b> | 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 initialise 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.

|            |                                     |
|------------|-------------------------------------|
| <b>can</b> | handle to the CAN object to delete. |
|------------|-------------------------------------|

### Return Values

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanDelete</i> successfully deletes a CAN object and invalidates its handle |
|---------------------|----------------------------------------------------------------------------------|

## 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.

|            |                          |
|------------|--------------------------|
| <b>can</b> | handle to the CAN object |
|------------|--------------------------|

| Return Values                |                                                                                                         |
|------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>CanValidate</i> successfully validates that the XMP is properly fitted for the CANOpen interface. |
| <b>MPIMessageUNSUPPORTED</b> | indicates that the XMP is not properly fitted for the CANOpen interface.                                |

## 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.

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <b>can</b>    | a handle to the CAN object                                                             |
| <b>config</b> | a pointer to the CAN configuration structure that will be filled in by this function.. |

| Return Values       |                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanConfigGet</i> successfully returns the copy of the current configuration of the CAN controller. |

## 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.

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <b>can</b>    | a handle to the CAN object                                                     |
| <b>config</b> | a pointer to the CAN configuration structure containing the new configuration. |

| Return Values       |                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanConfigSet</i> successfully updates the current configuration of the CAN controller. |

## 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.

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <b>can</b>    | handle to the CAN object                                                              |
| <b>flash</b>  | normally NULL                                                                         |
| <b>config</b> | a pointer to the CAN configuration structure that will be filled in by this function. |

| Return Values       |                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanFlashConfigGet</i> successfully returns a copy of the current flash configuration that the CAN controller is using. |

## 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.

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <b>can</b>    | handle to the CAN object                                                              |
| <b>flash</b>  | normally NULL                                                                         |
| <b>config</b> | a pointer to the CAN configuration structure that will be filled in by this function. |

| Return Values       |                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanFlashConfigSet</i> successfully updates the current flash configuration that the CAN controller is using. |

## 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  | if <i>CanStatus</i> successfully gets the instantaneous state of the local CAN interface to the CAN network. |

## 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  | if <i>CanVersion</i> successfully returns the version of the firmware being used by the CAN controller. |

## 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.

|                |                                                                                                                                                                |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>can</b>     | a handle to the CAN object                                                                                                                                     |
| <b>command</b> | 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       |                                                                           |
|---------------------|---------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanCommand</i> successfully performs the set of specified commands. |

## 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.

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

| Return Values       |                                                                                                                  |
|---------------------|------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeConfigGet</i> successfully returns the copy of the current configuration of the specified CAN node. |

## 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.

|                   |                                                                                     |
|-------------------|-------------------------------------------------------------------------------------|
| <b>can</b>        | a handle to the CAN object                                                          |
| <b>node</b>       | the node number of the CANOpen node                                                 |
| <b>nodeConfig</b> | a pointer to the CAN node configuration structure containing the new configuration. |

| Return Values       |                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeConfigSet</i> successfully updates the current configuration of the specified CAN node. |

## 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.

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

| Return Values       |                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeFlashConfigGet</i> successfully returns the copy of the current flash configuration of the CAN controller. |

## 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.

|                   |                                                                                     |
|-------------------|-------------------------------------------------------------------------------------|
| <b>can</b>        | a handle to the CAN object                                                          |
| <b>flash</b>      | normally NULL                                                                       |
| <b>node</b>       | the node number of the CANOpen node                                                 |
| <b>nodeConfig</b> | a pointer to the CAN node configuration structure containing the new configuration. |

| Return Values       |                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeFlashConfigSet</i> successfully updates the current flash configuration for the node. |

## 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.

|                   |                                                            |
|-------------------|------------------------------------------------------------|
| <b>can</b>        | handle to the CAN object                                   |
| <b>node</b>       | the node number of the CANOpen node.                       |
| <b>nodeStatus</b> | a pointer to where this function will put the node status. |

| Return Values       |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeStatus</i> successfully gets the instantaneous state of the specified node on the CAN network. |

## 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.

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

| Return Values       |                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeInfo</i> successfully returns the node information for the specified node. |

## See Also

[meiCanNodeStatus](#) | [meiCanStatus](#)



# meiCanNodeAnalogInputGet

## Declaration

```
long meiCanNodeAnalogInputGet(MEICan can,
                              long node,
                              long index,
                              double* data);
```

**Required Header:** stdmei.h

## Description

**meiCanNodeAnalogInputGet** gets the current analog input from the specified CAN Node. The analog data returned is scaled to between  $\pm 1.0$ .

|              |                                                          |
|--------------|----------------------------------------------------------|
| <b>can</b>   | handle to the CAN object                                 |
| <b>node</b>  | the node number of the CANOpen node.                     |
| <b>index</b> | the index to the analog input on the node.               |
| <b>data</b>  | a pointer to where the current analog input is returned. |

| Return Values       |                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeAnalogInputGet</i> successfully gets the current analog input from the specified CAN Node. |

## See Also

# meiCanNodeAnalogOutputGet

## Declaration

```
long meiCanNodeAnalogOutputGet(MEICan can,
                                long node,
                                long index,
                                double* data);
```

Required Header: stdmei.h

## Description

**meiCanNodeAnalogOutputGet** gets the current analog output from the specified CAN node and channel. The analog data returned is scaled to between  $\pm 1.0$ .

|       |                                                           |
|-------|-----------------------------------------------------------|
| can   | handle to the CAN object                                  |
| node  | the node number of the CANOpen node.                      |
| index | the index to the analog output on the node.               |
| data  | a pointer to where the current analog output is returned. |

| Return Values |                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>CanNodeAnalogOutputGet</i> successfully gets the current analog output from the specified CAN Node and channel. |

## See Also

[meiCanNodeAnalogOutputSet](#)

# meiCanNodeAnalogOutputSet

## Declaration

```
long meiCanNodeAnalogOutputSet(MEICan    can,  
                                long        node,  
                                long        index,  
                                double*    data);
```

Required Header: stdmei.h

## Description

**meiCanNodeAnalogOutputSet** sets the current analog output for the specified CAN node and channel. The analog data used is assumed to be between  $\pm 1.0$ .

|              |                                             |
|--------------|---------------------------------------------|
| <b>can</b>   | handle to the CAN object                    |
| <b>node</b>  | the node number of the CANOpen node.        |
| <b>index</b> | the index to the analog output on the node. |
| <b>data</b>  | the new analog value to be output.          |

| Return Values       |                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeAnalogOutputSet</i> successfully sets the current analog output for the specified CAN Node and channel. |

## See Also

[meiCanNodeAnalogOutputGet](#)

# meiCanNodeDigitalInputGet

## 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](#).)

|             |                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------|
| <b>can</b>  | Handle to the CAN object.                                                                            |
| <b>node</b> | The node number of the CANOpen node.                                                                 |
| <b>bit</b>  | Which bit on this node.<br>This value should be between 0 and 63.                                    |
| <b>data</b> | A pointer to where the current digital bit is returned.<br>The value returned will be either 0 or 1. |

| Return Values       |                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeDigitalInputGet</i> successfully gets the current digital input from the specified CAN Node. |

## 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

## 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  | if <i>CanNodeDigitalInputsGet</i> successfully gets the current state of all the digital input bits on the specified CAN Node. |

## See Also

[meiCanNodeDigitalInputGet](#)

# meiCanNodeDigitalOutputGet

## 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](#).)

|             |                                                         |
|-------------|---------------------------------------------------------|
| <b>can</b>  | handle to the CAN object                                |
| <b>node</b> | the node number of the CANOpen node.                    |
| <b>bit</b>  | which bit on this node.                                 |
| <b>data</b> | a pointer to where the current digital bit is returned. |

| Return Values       |                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeDigitalOutputGet</i> successfully gets the current digital output bit on the specified CAN Node. |

## See Also

[meiCanNodeDigitalOutputSet](#) | [meiCanNodeDigitalOutputsGet](#) | [meiCanNodeDigitalOutputsSet](#)

# meiCanNodeDigitalOutputsGet

## 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 [meiCanNodeDigitalOutputsGet](#).)

|      |                                                         |
|------|---------------------------------------------------------|
| 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  | if <i>CanNodeDigitalOutputsGet</i> successfully gets the current digital output bits on the specified CAN node. |

## See Also

[meiCanNodeDigitalOutputGet](#) | [meiCanNodeDigitalOutputSet](#) | [meiCanNodeDigitalOutputsSet](#)



# meiCanNodeDigitalOutputSet

## 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](#).)

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| <b>can</b>  | Handle to the CAN object                                                       |
| <b>node</b> | The node number of the CANOpen node.                                           |
| <b>bit</b>  | Which bit on this node.<br>This value should be between 0 and 63.              |
| <b>data</b> | The new state of the digital bit.<br>The value returned will be either 0 or 1. |

| Return Values       |                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanNodeDigitalOutputSet</i> successfully changes the state of the digital output bit on the specified CAN Node. |

## See Also

[meiCanNodeDigitalOutputGet](#) | [meiCanNodeDigitalOutputsGet](#) | [meiCanNodeDigitalOutputsSet](#)

# meiCanNodeDigitalOutputsSet

## 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  | if <i>CanNodeDigitalOutputsSet</i> successfully changes the current state of all the digital output bits on the specified CAN node. |

## See Also

[meiCanNodeDigitalOutputGet](#) | [meiCanNodeDigitalOutputSet](#) | [meiCanNodeDigitalOutputsGet](#)

# meiCanEventNotifyGet

## Declaration

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

Required Header: stdmei.h

## Description

**meiCanEventNotifyGet** gets the current CAN event mask.

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

| Return Values       |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventNotifyGet</i> successfully gets the current CAN event mask. |

## 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.

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

| Return Values       |                                                                           |
|---------------------|---------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanEventNotifySet</i> successfully sets the current CAN event mask. |

## See Also

[meiCanEventNotifyGet](#)

# meiCanFirmwareDownload

## Declaration

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

Required Header: stdmei.h

## 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  | if <i>CanFirmwareDownload</i> successfully uploads the CAN controller's firmware. |

## 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.

|            |                          |
|------------|--------------------------|
| <b>can</b> | handle to the CAN object |
|------------|--------------------------|

| Return Values       |                                                                               |
|---------------------|-------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanFirmwareErase</i> successfully erases the CAN controller's firmware. |

## See Also

[meiCanFirmwareDownload](#) | [meiCanFirmwareUpload](#)

# meiCanFirmwareUpload

## Declaration

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

**Required Header:** stdmei.h

## 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.

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

| Return Values       |                                                                                                     |
|---------------------|-----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanFirmwareUpload</i> successfully retrieves a copy of the current CAN controller's firmware. |

## 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  | if <i>CanMemory</i> successfully returns a pointer to the base of the CAN processors DPR. |

## See Also

[meiCanMemoryGet](#) | [meiCanMemorySet](#)



# meiCanMemoryGet

## Declaration

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

**Required Header:** stdmei.h

## 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.

|              |                                     |
|--------------|-------------------------------------|
| <b>can</b>   | handle to the CAN object            |
| <b>dst</b>   | the base address of the destination |
| <b>src</b>   | the base address of the source      |
| <b>count</b> | the number of bytes to copy         |

| Return Values       |                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CanMemoryGet</i> successfully copies the specified number of bytes to the application's memory. |

## See Also

[meiCanMemory](#) | [meiCanMemorySet](#)

# meiCanMemorySet

## Declaration

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

Required Header: stdmei.h

## 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  | if <i>CanMemorySet</i> successfully copies the specified number of bytes to the controller's memory. |

## See Also

[meiCanMemory](#) | [meiCanMemoryGet](#)

# 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.

|            |                           |
|------------|---------------------------|
| <b>can</b> | a handle to a Can object. |
|------------|---------------------------|

| Return Values        |                               |
|----------------------|-------------------------------|
| <b>MPIControl</b>    | a handle to a control object. |
| <b>MPIHandleVOID</b> | if Can is not valid.          |

## 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.).

|                |                                       |
|----------------|---------------------------------------|
| <b>can</b>     | a handle to a Can object.             |
| <b>*number</b> | a pointer to the index of a Can node. |

| Return Values                   |                                                                                                                                  |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>             | if <i>CanNumber</i> successfully reads the index of a Can object and writes it into the contents of a long pointed to by number. |
| <b>MPIMessageARG_INVALID</b>    | See <a href="#">description</a> .                                                                                                |
| <b>MPIMessageHANDLE_INVALID</b> | See <a href="#">description</a> .                                                                                                |

## 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.

|             |                                   |
|-------------|-----------------------------------|
| <b>type</b> | The type of CAN command.          |
| <b>data</b> | 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

# 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.

|                     |                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bitRate</b>      | The bit rate the CAN bus uses.<br>See also <a href="#">CAN Bit Rate</a> .                                                                                                                                                                                                                                                                                                                    |
| <b>cyclicPeriod</b> | The period (milliseconds) between sending consecutive SYNC messages. A value of zero will disable the SYNC messages from being produced.<br>See also <a href="#">CAN Transmission Types</a> .                                                                                                                                                                                                |
| <b>healthPeriod</b> | 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).<br>See also <a href="#">CAN Node Health</a> . |
| <b>nodeNumber</b>   | 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.<br>See also <a href="#">CAN Node Numbers</a> .                                                                                                                                                                                      |
| <b>inhibitTime</b>  | The global time used for the node health protocols.<br>See also <a href="#">CAN Transmission Types</a> .                                                                                                                                                                                                                                                                                     |

## See Also

[meiCanConfigGet](#) | [meiCanConfigSet](#) | [meiCanNodeConfigGet](#) | [meiCanNodeConfigSet](#)

# MEICanDigitalIO

## Definition

```
typedef struct MEICanDigitalIO {  
    unsigned long    data[2];  
} MEICanDigitalIO;
```

## Description

**MEICanCommandDigitalIO** holds the state of all the digital inputs or outputs on a CANOpen node.

**NOTE:** the maximum number of inputs or outputs on a single node supports is 64.

|             |                                   |
|-------------|-----------------------------------|
| <b>data</b> | Data associated with the command. |
|-------------|-----------------------------------|

## See Also

# 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_INITIALIZED,
    MEICanMessageCAN_INVALID,
    MEICanMessageIO_NOT_SUPPORTED,
    MEICanMessageFILE_FORMAT_ERROR,
    MEICanMessageUSER_ABORT,
    MEICanMessageCOMMAND_PROTOCOL,
    MPICanMessageINTERFACE_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

# MEICanNodeInfo

## Definition

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

## Description

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

|                           |                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>               | An enumeration indicating the type of node found at startup, or MEICanNodeTypeNONE if no node was found.                                                                                                                                                                                                                                                                                            |
| <b>digitalInputCount</b>  | 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.      |
| <b>digitalOutputCount</b> | 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. |
| <b>analogInputCount</b>   | The number of analog inputs supported by this node.                                                                                                                                                                                                                                                                                                                                                 |
| <b>analogOutputCount</b>  | The number of analog outputs supported by this node.                                                                                                                                                                                                                                                                                                                                                |
| <b>healthType</b>         | The type of health checking protocol being used with this node. Also see <a href="#">CAN Node Health</a> .                                                                                                                                                                                                                                                                                          |

|                      |                                                                                                                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>vendorId</b>      | 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.                                            |
| <b>productCode</b>   | 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. |
| <b>versionNumber</b> | 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.                                                   |
| <b>serialNumber</b>  | 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

# MEICanNodeStatus

## Definition

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

## Description

**MEICanNodeStatus** holds the current status of a node.

|                 |                                                      |
|-----------------|------------------------------------------------------|
| <b>live</b>     | Set if the node is alive, clear if the node is dead. |
| <b>nmtState</b> | 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.

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

# 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.

|                            |                                           |
|----------------------------|-------------------------------------------|
| <b>bootloaderVersion</b>   | The version number of the CAN bootloader. |
| <b>firmwareVersion</b>     | The CAN firmware version.                 |
| <b>firmwareRevision</b>    | The CAN firmware revision.                |
| <b>firmwareSubRevision</b> | 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

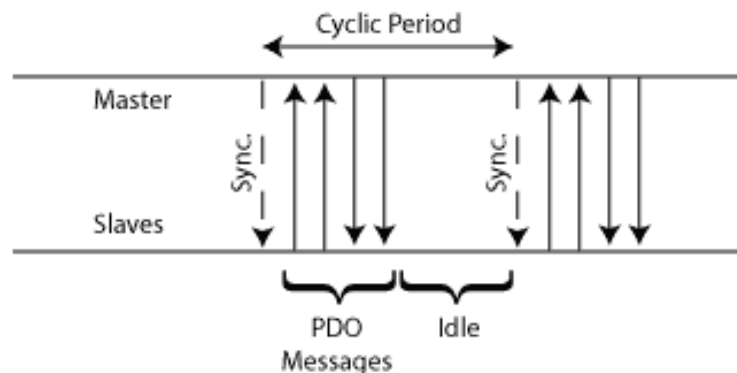
# CAN Transmission Types

## 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

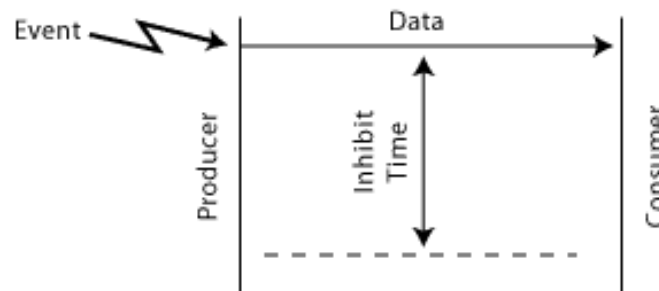
The cyclicPeriod field within the [MEICanConfig](#) structure allows the user to specify the period (in milliseconds) that the XMP will use between the successive transmission of synchronization messages. The minimum cyclic period that can be used is dependent upon the chosen bit rate and the number 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

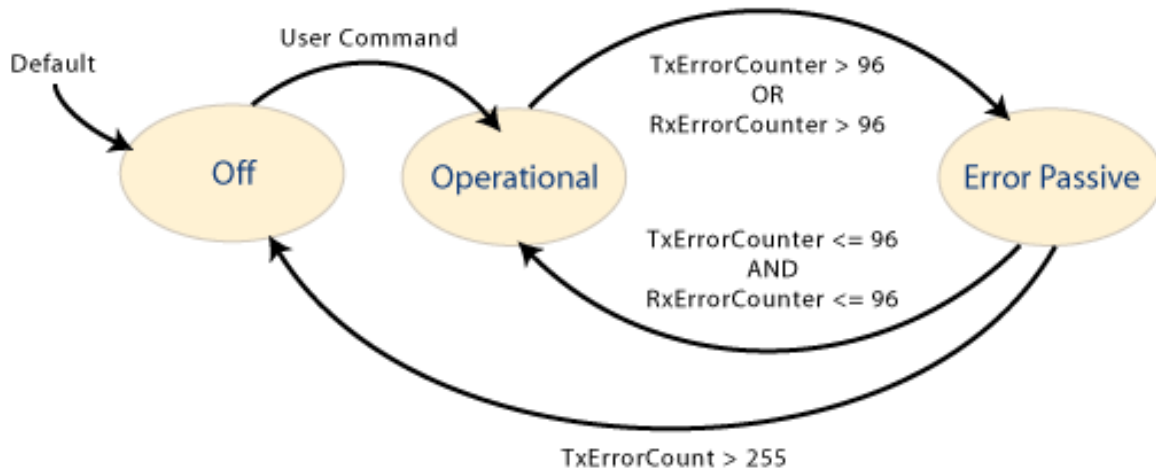
If the source node's events occur at a very fast rate, the number of messages generated can swamp the network and consequently block out other messages. To prevent an excess of messages, nodes can optionally support inhibit times for their transmit PDOs. This value defines the minimum time between two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



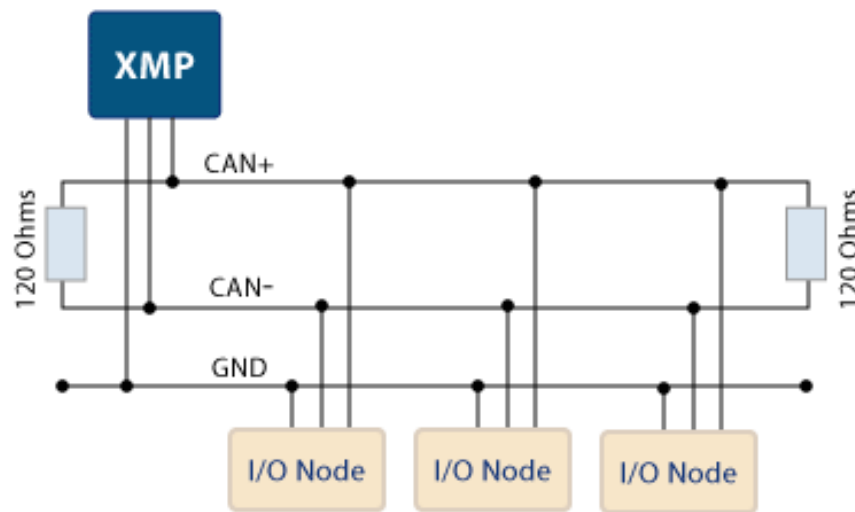
When a node is in the **Operational** state it will participate fully with all communications over the network, as the errors increase the CAN hardware will become **Passive** (detecting errors but not generating error messages), before turning **Off** and isolating the node from the network once the 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 than 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.

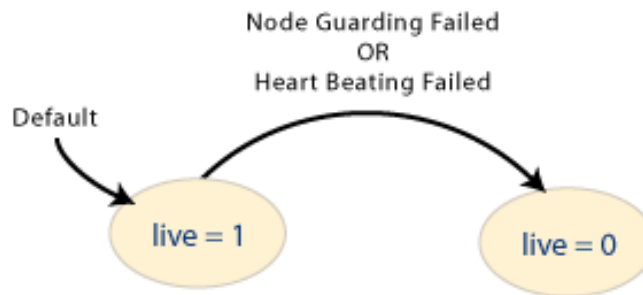
Each node on the network must have a unique node number, in the range of 1 to 127. The node number is commonly set with a bank of DIP switches on each node. If two nodes are given the same node number, network errors are generated and unpredictable problems will be encountered. The node number of the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



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.



## Node Health Times

| Protocol Times  | Value        |
|-----------------|--------------|
| Node Guard Time | healthPeriod |

|                         |                  |
|-------------------------|------------------|
| Heartbeat Producer Time | healthPeriod / 2 |
| Heartbeat Consumer Time | healthPeriod     |

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

# CAN Emergency Messages

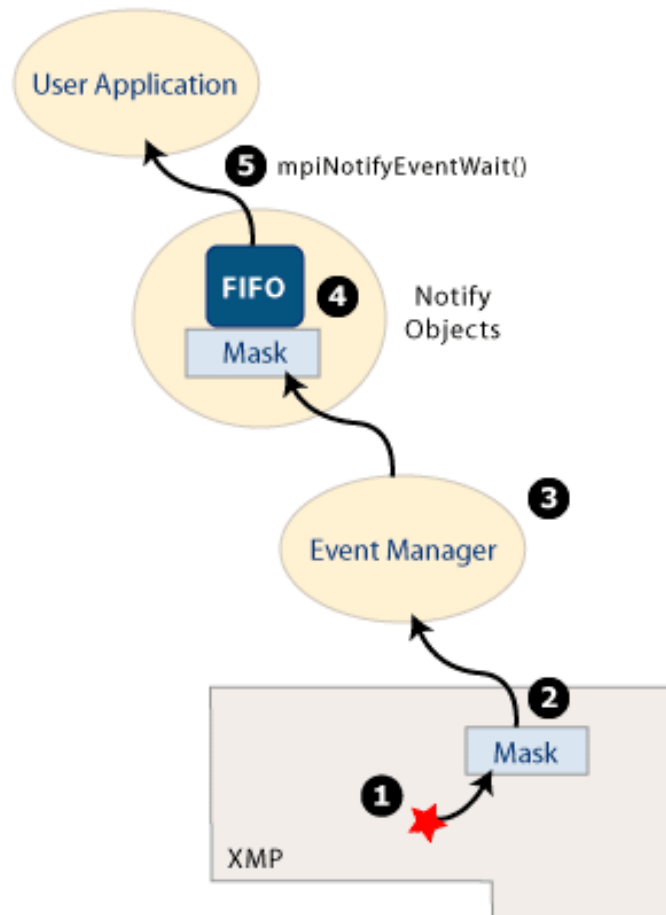
Every type of CANOpen node can transmit an emergency message. These messages are designed to report errors and warnings, as well as fatal problems on a node. The contents of these emergency messages are very dependent upon the node manufacturer and node type. To interpret this data, you will need to 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.

# 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.

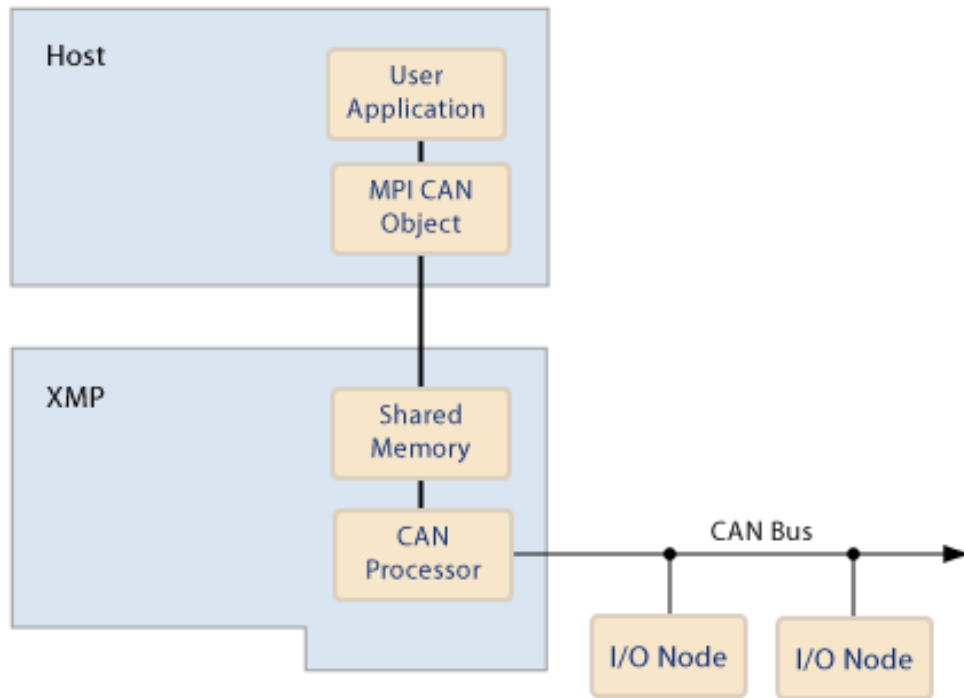


1. The CANOpen firmware detects one of the CAN events.
2. There is a mask within the XMP firmware that allows only a specified set of events to reach the host. This mask is interrogated and modified with the [meiCanEventNotifyGet](#) and [meiCanEventNotifySet](#) functions.
3. Like all other events in 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).



# XMP Overview

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.

# 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](#).

## Methods

### Create, Delete, Validate Methods

|                                    |                         |
|------------------------------------|-------------------------|
| <a href="#">mpiCommandCreate</a>   | Create Command object   |
| <a href="#">mpiCommandDelete</a>   | Delete Command object   |
| <a href="#">mpiCommandValidate</a> | Validate Command object |

### Configuration and Informational Methods

|                                  |                              |
|----------------------------------|------------------------------|
| <a href="#">mpiCommandLabel</a>  | Get pointer to Command label |
| <a href="#">mpiCommandParams</a> | Get Command parameters       |
| <a href="#">mpiCommandType</a>   | Return Command type          |

### Other Methods

|                                       |                                                       |
|---------------------------------------|-------------------------------------------------------|
| <a href="#">mpiCommandAxisListGet</a> | Get the axisCount and axisList from a Command object. |
|---------------------------------------|-------------------------------------------------------|

## Data Types

[MPICommandAddress](#)  
[MPICommandConstant](#)  
[MPICommandExpr](#)  
[MPICommandMessage](#)  
[MPICommandMotion](#)  
[MPICommandOperator](#)  
[MPICommandParams](#)  
[MPICommandType](#)

## See Also

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

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

|                                 |                                                                                                  |           |
|---------------------------------|--------------------------------------------------------------------------------------------------|-----------|
| <b>MPICommandTypeWAIT_EVENT</b> | Delays execution of the next command until a particular event occurs.                            | waitEvent |
| <b>MPICommandTypeWAIT_IO</b>    | 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](#)

# 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                                       |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| <b>assign</b> | Assign a value to a particular controller address:<br><b>*dst = value</b><br>assign.control is currently not supported and is reserved for future use. | MPICommandTypeASSIGN<br>MPICommandTypeASSIGN_FLOAT |



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

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

See Also

[MPICommand](#) | [MPICommandType](#) | [mpiCommandCreate](#) | [mpiCommandParams](#)

# 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.

## Methods

### Create, Delete, Validate Methods

[mpiCaptureCreate](#)

Create Capture object

[mpiCaptureDelete](#)

Delete Capture object

[mpiCaptureValidate](#) Validate Capture object

## Configuration and Information Methods

[mpiCaptureConfigGet](#) Get Capture configuration

[mpiCaptureConfigSet](#) Set Capture configuration

[mpiCaptureStatus](#) Get status of Capture

[mpiCaptureConfigReset](#)

## Action Methods

[mpiCaptureArm](#) Arm capture object

## Memory Methods

[mpiCaptureMemory](#) Set address to Capture memory

[mpiCaptureMemoryGet](#) Copy Capture memory to application memory

[mpiCaptureMemorySet](#) Copy application memory to Capture memory

## Relational Methods

[mpiCaptureControl](#)

[mpiCaptureNumber](#) Get index of Capture (for Control list)

## Data Types

[MPICaptureConfig](#)

[MPICaptureEdge](#)

[MPICaptureMessage](#) / [MEICaptureMessage](#)

[MPICaptureSource](#)

[MPICaptureState](#)

[MPICaptureStatus](#)

[MPICaptureTrigger](#)

[MPICaptureTriggerGlobal](#)

[MPICaptureType](#)

## Constants

[MPICaptureNOT\\_MAPPED](#)

# 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.

|                |                                          |
|----------------|------------------------------------------|
| <b>control</b> | a handle to a Control object             |
| <b>number</b>  | An index to the encoder's capture block. |

| Return Values        |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to a Capture object                |
| <b>MPIHandleVOID</b> | 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

|                     |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CaptureDelete</i> successfully deletes the Capture object and invalidates its handle |
|---------------------|--------------------------------------------------------------------------------------------|

## 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.

|                |                              |
|----------------|------------------------------|
| <b>capture</b> | a handle to a Capture object |
|----------------|------------------------------|

### Return Values

|                     |                                           |
|---------------------|-------------------------------------------|
| <b>MPIMessageOK</b> | if Capture is a handle to a valid object. |
|---------------------|-------------------------------------------|

## 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* either points to a structure of type **MEICaptureConfig{}** or is NULL.

### Return Values

**MPIMessageOK**

if CaptureConfigGet successfully writes the Capture object's configuration to the structure(s)

## 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(...)`.

## Remarks

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

### Return Values

**MPIMessageOK**

if *CaptureConfigSet* successfully sets the Capture object's configuration using data from the structure(s)

## 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.

|                  |                                                   |
|------------------|---------------------------------------------------|
| <b>capture</b>   | a handle to a Capture object                      |
| <b>*status</b>   | a pointer to MPIStatus structure                  |
| <b>*external</b> | a pointer to an implementation-specific structure |

| Return Values                |                                                                                                |
|------------------------------|------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>CaptureStatus</i> successfully writes the status of a Capture object to the structure(s) |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL.                                                          |

## 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(...).

|                |                              |
|----------------|------------------------------|
| <b>capture</b> | a handle to a Capture object |
|----------------|------------------------------|

### Return Values

|                     |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CaptureConfigReset</i> successfully returns the capture object to its unmapped state. |
|---------------------|---------------------------------------------------------------------------------------------|

## See Also

[mpiCaptureConfigGet](#) | [mpiCaptureConfigSet](#) | [MPICaptureConfig](#)

# mpiCaptureArm

## Declaration

```
long mpiCaptureArm(MPICapture capture,
                   long arm) /* TRUE/FALSE */
```

Required Header: stdmpi.h

## Description

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

| Value of "arm" | Action of mpiCaptureArm                                                            |
|----------------|------------------------------------------------------------------------------------|
| FALSE          | Disarms <i>capture</i> and sets the state of <i>capture</i> to MPICaptureStateIDLE |
| TRUE           | Arms <i>capture</i> and sets the state of <i>capture</i> to MPICaptureStateARMED   |

| Return Values |                                                         |
|---------------|---------------------------------------------------------|
| MPIMessageOK  | if the Capture object is successfully armed or disarmed |

## 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**

if *CaptureMemory* successfully writes the Capture object's memory address to the contents of *memory*

## See Also

[mpiCaptureMemoryGet](#) | [mpiCaptureMemorySet](#)

# mpiCaptureMemoryGet

## Declaration

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

Required Header: stdmpi.h

## 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       |                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CaptureMemoryGet</i> successfully copies data from Capture memory to application memory |

## See Also

[mpiCaptureMemory](#) | [mpiCaptureMemorySet](#)

# mpiCaptureMemorySet

## Declaration

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

Required Header: stdmpi.h

## 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       |                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CaptureMemorySet</i> successfully copies count bytes of application memory to Capture memory |

## 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.

|                |                              |
|----------------|------------------------------|
| <b>capture</b> | a handle to a Capture object |
|----------------|------------------------------|

| Return Values        |                            |
|----------------------|----------------------------|
| <b>MPIControl</b>    | Handle to a Control object |
| <b>MPIHandleVOID</b> | 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.

|                |                                |
|----------------|--------------------------------|
| <b>capture</b> | a handle to a capture object   |
| <b>*number</b> | pointer to the capture number. |

| Return Values       |                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CaptureNumber</i> successfully writes the index of a Capture object to the contents of <b>number</b> |

## 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

|                                                         |                                                                                                                                                                                                 |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>source</b> [ <a href="#">MPICaptureSourceCOUNT</a> ] | An array of capture trigger source inputs. The capture can be configured to trigger from one or more sources. See <a href="#">MPICaptureTrigger</a> and <a href="#">MPICaptureSourceCOUNT</a> . |
| <b>edge</b>                                             | An enumerated index to the trigger edge type. The capture can be configured to trigger from a variety of logic. See <a href="#">MPICaptureEdge</a> .                                            |
| <b>global</b>                                           | A structure to configure the global capture, to chain capture block triggering. See <a href="#">MPICaptureTriggerGlobal</a> .                                                                   |
| <b>type</b>                                             | Specifies either position-based or time-based capture. Use <a href="#">MPICaptureTypePOSITION</a> for position-based capture and <a href="#">MPICaptureTypeTIME</a> for time-based capture.     |
| <b>captureMotorNumber</b>                               | The number of the motor whose "source" (MPICaptureTrigger) is used to capture position.                                                                                                         |
| <b>feedbackMotorNumber</b>                              | The number of the motor whose position is being returned from the capture event. (It must be the same as captureMotorNumber for position capture).                                              |
| <b>encoder</b>                                          | 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.

|                              |                                                  |
|------------------------------|--------------------------------------------------|
| <b>MPICaptureEdgeRISING</b>  | Triggers on a 0 to 1 transition.                 |
| <b>MPICaptureEdgeFALLING</b> | Triggers on a 1 to 0 transition.                 |
| <b>MPICaptureEdgeEITHER</b>  | 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

mpiCaptureConfigSet(...) --> config.captureMotorNumber is not valid. It's either greater than maxMotors or == MPICaptureNOT.MAPPED.

### MEICaptureMessageCAPTURE\_TYPE\_INVALID

mpiCaptureConfigSet(...) --> config.Type == MPICaptureNOT.MAPPED.

### 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 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,
} MEICaptureMessage;
```

## 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 `mpiCatureCreate(...)` 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 [MPCaptureState](#).

### 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.

**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.

When using one of the MPICaptureSourceMOTOR\_IO values in MPICaptureSource, you can determine which MPICaptureSourceMOTOR\_IO to use by referencing the appropriate node module. Look in *NodeMotorIoConfig* (replacing *Node* with your node name) in the appropriate node module. Add the appropriate *NodeMotorIoConfig* value to MPICaptureSourceMOTOR\_IO\_0.

### Example: RMB-10V

Let's say you are using an MEI RMB-10V and want to find the trigger for XCVR\_C.

Look in RMBMotorIoConfig in *mei\_rmb.h*. You will find that the appropriate value for XCVR\_C is RMBMotorIoConfigXCVR\_C. RMBMotorIoConfigXCVR\_C is the third value in RMBMotorIoConfig. This means that the value to use in MPICaptureSource is MPICaptureSourceMOTOR\_IO\_2 (the third MPICaptureSourceMOTOR\_IO value).

A better way of making this conversion in your program is to add the MPICaptureSourceMOTOR\_IO\_0 to the nodeMotorIoConfig value you want to use. In the above example, it would be (MPICaptureSourceMOTOR\_IO\_0 + RMBMotorIoConfigXCVR\_C).



**Example: Trust TA800**

To trigger off of hall A on a Trust TA800 node, you would use (MPICaptureSourceMOTOR\_IO\_0 + TA800MotorIoConfigHALL\_A). Remember that you will need to look in *trust\_ta800.h* (the node module) to find TA800MotorIoConfigHALL\_A.

|                                     |                                                                                                                                              |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPICaptureSourceMOTOR_IO_0</b>   | a capture trigger source is the 0 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_1</b>   | a capture trigger source is the 1 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_2</b>   | a capture trigger source is the 2 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_3</b>   | a capture trigger source is the 3 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_4</b>   | a capture trigger source is the 4 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_5</b>   | a capture trigger source is the 5 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_6</b>   | a capture trigger source is the 6 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceMOTOR_IO_7</b>   | a capture trigger source is the 7 bit in the motor's configurable I/O.                                                                       |
| <b>MPICaptureSourceHOME</b>         | a capture trigger source is the HOME input in the dedicated I/O input.                                                                       |
| <b>MPICaptureSourceINDEX</b>        | a capture trigger source is the encoder INDEX input in the dedicated I/O input.                                                              |
| <b>MPICaptureSourceLIMIT_HW_NEG</b> | a capture trigger source is the Hardware Negative Limit input in the dedicated I/O input.                                                    |
| <b>MPICaptureSourceLIMIT_HW_POS</b> | a capture trigger source is the Hardware Positive Limit input in the dedicated IO word. Please see <a href="#">MPIMotorInfoDedicatedIn</a> . |
| <b>MPICaptureSourceGLOBAL</b>       | a capture trigger source is the Global capture signal found on the node. Please see <a href="#">MPICaptureTriggerGlobal</a> .                |

|                                        |                                                                                                                                                                                                      |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPICaptureSourceINDEX_SECONDARY</b> | 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. |
| <b>MPICaptureSourceCOUNT</b>           | Total number of possible input sources for a capture.                                                                                                                                                |

## See Also

[MPICaptureTrigger](#)

# MPICaptureState

## Definition

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

## Description

|                                |                                                                                                                                                                               |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPICaptureStateIDLE</b>     | Capture is not armed. This is the default state.                                                                                                                              |
| <b>MPICaptureStateARMED</b>    | Capture is armed, but has not triggered yet.                                                                                                                                  |
| <b>MPICaptureStateCAPTURED</b> | Capture triggered and position data is valid.                                                                                                                                 |
| <b>MPICaptureStateCLEAR</b>    | 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](#)

# MPICaptureTrigger

## Definition

```
typedef struct MPICaptureTrigger {  
    long enabled;    /* TRUE/FALSE */  
    long invert;     /* TRUE = invert, FALSE = normal */  
} MPICaptureTrigger;
```

## Description

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

|                |                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------|
| <b>enabled</b> | Enables or disables the trigger. A value of TRUE enables the trigger, FALSE disables the trigger.                  |
| <b>invert</b>  | Normal or inverted trigger polarity. A value of FALSE indicates normal polarity, TRUE indicates inverted polarity. |

## See Also

[MPICaptureSource](#)

# MPICaptureTriggerGlobal

## Definition

```
typedef struct MPICaptureTriggerGlobal {  
    long    enabled;    /* TRUE/FALSE */  
} MPICaptureTriggerGlobal;
```

## Description

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

|                |                                                                                                                |
|----------------|----------------------------------------------------------------------------------------------------------------|
| <b>enabled</b> | 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

|                               |                                                                                                             |
|-------------------------------|-------------------------------------------------------------------------------------------------------------|
| <b>MPICaptureTypePOSITION</b> | An actual position is captured by the Node from its feedback source.                                        |
| <b>MPICaptureTypeTIME</b>     | 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](#).

## Methods

### Create, Delete, Validate Methods

|                                    |                         |
|------------------------------------|-------------------------|
| <a href="#">mpiCommandCreate</a>   | Create Command object   |
| <a href="#">mpiCommandDelete</a>   | Delete Command object   |
| <a href="#">mpiCommandValidate</a> | Validate Command object |

### Configuration and Informational Methods

|                                  |                              |
|----------------------------------|------------------------------|
| <a href="#">mpiCommandLabel</a>  | Get pointer to Command label |
| <a href="#">mpiCommandParams</a> | Get Command parameters       |
| <a href="#">mpiCommandType</a>   | Return Command type          |

### Other Methods

|                                       |                                                       |
|---------------------------------------|-------------------------------------------------------|
| <a href="#">mpiCommandAxisListGet</a> | Get the axisCount and axisList from a Command object. |
|---------------------------------------|-------------------------------------------------------|

## Data Types

[MPICommandAddress](#)  
[MPICommandConstant](#)  
[MPICommandExpr](#)  
[MPICommandMessage](#)  
[MPICommandMotion](#)  
[MPICommandOperator](#)  
[MPICommandParams](#)  
[MPICommandType](#)

## See Also



[MPISequence](#)

# mpiCommandCreate

## Declaration

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

**Required Header:** stdmei.h

## Description

**mpiCommandCreate** 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        |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to a Command object                |
| <b>MPIHandleVOID</b> | 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

if *CommandDelete* successfully deletes the Command object and invalidates its handle

## See Also

[mpiCommandCreate](#) | [mpiCommandValidate](#)

# mpiCommandValidate

## Declaration

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

**Required Header:** stdmpi.h

## Description

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

|                |                                                      |
|----------------|------------------------------------------------------|
| <b>command</b> | a handle to the Command object                       |
| <b>*type</b>   | a pointer to a MPICommandType returned by the method |

### Return Values

|                     |                                                |
|---------------------|------------------------------------------------|
| <b>MPIMessageOK</b> | if the Command object and its handle are valid |
|---------------------|------------------------------------------------|

## 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.

|                |                                              |
|----------------|----------------------------------------------|
| <b>command</b> | a handle to the Command object               |
| <b>**label</b> | a pointer to a string returned by the method |

| Return Values       |                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointer</b>      | to a Command's ( <i><b>command</b></i> ) label (that is in the location pointed to by <i><b>label</b></i> )                              |
| <b>MPIMessageOK</b> | if <i>CommandLabel</i> successfully returns a pointer to the Command's label that is in the location pointed to by <i><b>**label</b></i> |

## 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                         |                                                                                                 |
|---------------------------------------|-------------------------------------------------------------------------------------------------|
| Command ( <i>command</i> ) parameters | in the structure pointed to by <i>params</i>                                                    |
| MPIMessageOK                          | if <i>CommandParams</i> successfully gets and writes the command parameters into <i>*params</i> |

## 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*.

|                |                                                      |
|----------------|------------------------------------------------------|
| <b>command</b> | a handle to the Command object                       |
| <b>*type</b>   | a pointer to a MPICommandType returned by the method |

| Return Values                              |                                                                                       |
|--------------------------------------------|---------------------------------------------------------------------------------------|
| <b>Command (<i>command</i>) parameters</b> | in the location pointed to by <i>type</i>                                             |
| <b>MPIMessageOK</b>                        | if <i>CommandType</i> successfully gets and writes the command type into <i>*type</i> |

## 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***.

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

| Return Values       |                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CommandAxisListGet</i> successfully gets the <b><i>axisCount</i></b> and <i>axisList</i> from a Command object. |

## 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.

|           |                                                           |
|-----------|-----------------------------------------------------------|
| <b>*l</b> | is used to access the long pointer of MPICommandAddress.  |
| <b>*f</b> | 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.

|          |                                                          |
|----------|----------------------------------------------------------|
| <b>l</b> | is used to access the long value of MPICommandConstant.  |
| <b>f</b> | 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 <b>oper</b> value |
|  | *address <b>oper</b> *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_ABORT,
    MPICommandMotionE_STOP_CMD_EQ_ACT,
    MPICommandMotionMODIFY,
    MPICommandMotionRESET,
    MPICommandMotionRESUME,
    MPICommandMotionSTART,
    MPICommandMotionSTOP,
} MPICommandMotion;
```

## 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.

|                                          |                                                                                                                                                                                                               |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPICommandMotionABORT</b>             | Commands an Abort action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , MPIActionABORT for details.                                                   |
| <b>MPICommandMotionE_STOP</b>            | Commands an E-Stop action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , MPIActionE_STOP for details.                                                 |
| <b>MPICommandMotionE_STOP_ABORT</b>      | Commands an E-Stop, then Abort action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , MPIActionE_STOP_ABORT for details.                               |
| <b>MPICommandMotionE_STOP_CMD_EQ_ACT</b> | Commands an E-Stop (command position = actual position) action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , MPIActionE_STOP_CMD_EQ_ACT for details. |

|                               |                                                                                                                                                                                                                                                                               |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPICommandMotionMODIFY</b> | Commands a Motion Modify on the motion supervisor associated with the motion object. Make sure to specify the <a href="#">MPIMotionType</a> and MPIMotionParams in the <a href="#">MPICommandParams{...}</a> structure. See <a href="#">mpiMotionModify(...)</a> for details. |
| <b>MPICommandMotionRESET</b>  | Commands a Reset action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , <a href="#">MPIActionRESET</a> for details.                                                                                                    |
| <b>MPICommandMotionRESUME</b> | Commands a Resume action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , <a href="#">MPIActionRESUME</a> for details.                                                                                                  |
| <b>MPICommandMotionSTART</b>  | Commands a Motion Start on the motion supervisor associated with the motion object. Make sure to specify the MPIMotionType and MPIMotionParams in the <a href="#">MPICommandParams{...}</a> structure. See <a href="#">mpiMotionStart(...)</a> for details.                   |
| <b>MPICommandMotionSTOP</b>   | Commands a Stop action on the motion supervisor associated with the motion object. See <a href="#">mpiMotionAction(...)</a> , <a href="#">MPIActionSTOP</a> 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              |                                                              |
|-----------------------------------|--------------------------------------------------------------|
| <b>MPICommandOperatorADD</b>      | Performs an addition. Equivalent to the C operator (+).      |
| <b>MPICommandOperatorSUBTRACT</b> | Performs a subtraction. Equivalent to the C operator (-).    |
| <b>MPICommandOperatorMULTIPLY</b> | Performs a multiplication. Equivalent to the C operator (*). |

|                                 |                                                           |
|---------------------------------|-----------------------------------------------------------|
| <b>MPICommandOperatorDIVIDE</b> | Performs a division. Equivalent to the C operator (/).    |
| <b>MPICommandOperatorAND</b>    | Performs a logical AND. Equivalent to the C operator (&). |
| <b>MPICommandOperatorOR</b>     | Performs a logical OR. Equivalent to the C operator ( ).  |
| <b>MPICommandOperatorXOR</b>    | Performs a logical XOR. Equivalent to the C operator (^). |

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

## See Also

[MPICommand](#) | [MPICommandExpr](#) | [MPICommandParams](#) | [MPIComparePosition](#) | [MPICompareParams](#)



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

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

|                                 |                                                                                                  |           |
|---------------------------------|--------------------------------------------------------------------------------------------------|-----------|
| <b>MPICommandTypeWAIT_EVENT</b> | Delays execution of the next command until a particular event occurs.                            | waitEvent |
| <b>MPICommandTypeWAIT_IO</b>    | 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](#)

# 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                                       |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| <b>assign</b> | Assign a value to a particular controller address:<br><b>*dst = value</b><br>assign.control is currently not supported and is reserved for future use. | MPICommandTypeASSIGN<br>MPICommandTypeASSIGN_FLOAT |

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

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



# 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](#)

## Methods

### Create, Delete, Validate Methods

[mpiCompensatorCreate](#)

Create Compensator object

[mpiCompensatorDelete](#)

Delete Compensator object

[mpiCompensatorValidate](#)

Validate Compensator object

### Configuration and Information Methods

[mpiCompensatorConfigGet](#)

Get Compensator configuration

[mpiCompensatorConfigSet](#)

Set Compensator configuration

[mpiCompensatorInfo](#)

Get Compensator information

[mpiCompensatorTableGet](#)

Get Compensator table

[mpiCompensatorTableSet](#)

Set Compensator table

## Memory Methods

[mpiCompensatorMemory](#)

Set address to be used to access Compensator memory

[mpiCompensatorMemoryGet](#)

Get bytes of Compensator memory and place it into application memory

[mpiCompensatorMemorySet](#)

Put (set) bytes of application memory into Compensator memory

## Relational Methods

[mpiCompensatorControl](#)

Return handle of Control object associated with Compensator

[mpiCompensatorNumber](#)

Get number of Compensator

## Data Types

[MPICompensatorConfig](#)

[MPICompensatorDimension](#)

[MPICompensatorInfo](#)

[MPICompensatorInputAxis](#)

[MPICompensatorMessage](#)

[MPICompensatorRange](#)

## Constants

[MPICompensatorDimensionsMAX](#)

# 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

if *CompensatorDelete* successfully deletes a Compensator object and invalidates its handle

## 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                                   |                                                                                                                                      |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                             | if Compensator is a handle to a valid object.                                                                                        |
| <b>MPICompensatorMessageCOMPENSATOR_INVALID</b> | If the <i>number</i> used in mpiCompensatorCreate() was not a valid number. Valid numbers range from 0 to MPIControlMAX_COMPENSATORS |
| <b>MPICompensatorMessageNOT_ENABLED</b>         | If the corresponding controller compensation object number is valid but is disabled on the controller.                               |

## 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                  |                                                                                                                                 |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>            | if <i>CompensatorConfigGet</i> successfully gets the configuration of a Compensator object and writes it into the structure(s). |
| <b>MPIMessagePARAM_INVALID</b> | If <i>*config</i> parameter is NULL, or if <i>*external</i> parameter is NOT NULL.                                              |

## 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                  |                                                                                                                                                           |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>            | if <i>CompensatorConfigSet</i> successfully writes the configuration of a Compensator object using data from the structure(s)                             |
| <b>MPIMessagePARAM_INVALID</b> | If <i>*config</i> parameter is NULL, or if <i>*external</i> parameter is NOT NULL                                                                         |
| <b>MPIMessageARG_INVALID</b>   | The axisOutNumber or any of the inputAxis[n].axisNumbers correspond to invalid Axis objects on the controller. See <a href="#">MPICompensatorConfig</a> . |

|                                                     |                                                                                                                                                                   |
|-----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPICompensatorMessageDIMENSION_NOT_SUPPORTED</b> | dimensionCount is out of range. See <a href="#">MPICompensatorConfig</a> .                                                                                        |
| <b>MPICompensatorMessageAXIS_NOT_ENABLED</b>        | The axisOutNumber or any of the inputAxis[n].axisNumbers correspond to disabled Axis objects on the controller. See <a href="#">MPICompensatorConfig</a> .        |
| <b>MPICompensatorMessagePOSITION_DELTA_INVALID</b>  | The <i>positionDelta</i> argument is either out of range or not a multiple of the range. See <a href="#">MPICompensatorConfig</a> .                               |
| <b>MPICompensatorMessageTABLE_SIZE_ERROR</b>        | The host compensation table will not fit within the controller's configured compensation table. See <a href="#">Determining Required Compensator Table Size</a> . |

## 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*.

|                    |                                                   |
|--------------------|---------------------------------------------------|
| <b>compensator</b> | a handle to the Compensator object.               |
| <b>*info</b>       | a pointer to a compensator information structure. |

| Return Values                              |                                                                                                                  |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                        | if <i>CompensatorInfo</i> successfully writes the Compensator object information into the <i>info</i> structure. |
| <b>MPICompensatorMessageNOT_CONFIGURED</b> | MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo.       |

## 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                              |                                                                                                            |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                        |                                                                                                            |
| <b>MPICompensatorMessageNOT_CONFIGURED</b> | MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo. |
| <b>MPIMessagePARAM_INVALID</b>             | If <i>*table</i> is NULL.                                                                                  |

## 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                              |                                                                                                            |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                        |                                                                                                            |
| <b>MPICompensatorMessageNOT_CONFIGURED</b> | MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo. |
| <b>MPIMessagePARAM_INVALID</b>             | If <i>*table</i> is NULL.                                                                                  |

## 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       |                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CompensatorMemory</i> successfully writes the Compensator's address to the contents of <i>memory</i> . |

## See Also

[mpiCompensatorMemoryGet](#) | [mpiCompensatorMemorySet](#)

# mpiCompensatorMemoryGet

## Declaration

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

Required Header: stdmpi.h

## 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  | if <i>CompensatorMemoryGet</i> successfully copies data from Compensator memory to application memory. |

## See Also

[mpiCompensatorMemorySet](#) | [mpiCompensatorMemory](#)

# mpiCompensatorMemorySet

## Declaration

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

Required Header: stdmpi.h

## Description

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

| Return Values       |                                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>CompensatorMemorySet</i> successfully copies data from application memory to Compensator memory. |

## 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.

|                    |                                    |
|--------------------|------------------------------------|
| <b>compensator</b> | a handle to the Compensator object |
|--------------------|------------------------------------|

| Return Values        |                                  |
|----------------------|----------------------------------|
| <b>MPIControl</b>    | handle to a Control object       |
| <b>MPIHandleVOID</b> | if <i>compensator</i> 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  | if <i>CompensatorNumber</i> successfully writes the index of a compensation object to the contents of <i>number</i> . |

## See Also

[mpiCompensatorCreate](#)



# MPICompensatorConfig

## Definition

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

## Description

|                         |                                                                                                                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>dimensionCount</b>   | 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 |
| <b>inputAxis</b>        | The substructure used to configure each input dimension of the Compensator object.                                                                                                   |
| <b>outputAxisNumber</b> | 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.

|                                 |                               |
|---------------------------------|-------------------------------|
| <b>MPICompensatorDimensionX</b> | First Compensating Dimension  |
| <b>MPICompensatorDimensionY</b> | Second Compensating Dimension |

## See Also

[MPICompensatorConfig](#) | [MPICompensatorInfo](#)

# MPICompensatorInfo

## Definition

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

## Description

|                        |                                                                                                                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>tableDimensions</b> | The dimensions along each axis of the table. This value is affected by the values set in the MPICompensatorConfig structure.                                                                                                                                                                                        |
| <b>tableSizeBytes</b>  | 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.                                                                                                                                           |
| positionDelete | Spacing between compensation positions on the compensating axis.<br><b>NOTE:</b> This value must be an exact multiple of the range (i.e. the difference between 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**

If the number used to in `mpiCompensatorCreate(...)` was not a valid number. Valid numbers range from 0 to `MPIControlMAX_COMPENSATORS`.

### **MPICompensatorMessageNOT\_CONFIGURED**

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

### **MPICompensatorMessageNOT\_ENABLED**

If the corresponding controller compensation object number is valid, but is disabled on the controller.

### **MPICompensatorMessageAXIS\_NOT\_ENABLED**

The `axisOutNumber` or any of the `inputAxis[n].axisNumbers` correspond to disabled Axis objects on the controller. See [MPICompensatorConfig{}](#).

### **MPICompensatorMessageTABLE\_SIZE\_ERROR**

The host compensation table will not fit within the controller's configured compensation table. See [Determining Required Compensation Table Size](#).

### **MPICompensatorMessagePOSITION\_DELTA\_INVALID**

The `positionDelta` argument is either out of range or is not a multiple of the range. See [MPICompensatorConfig{}](#).

### **MPICompensatorMessageDIMENSION\_NOT\_SUPPORTED**

`dimensionCount` is out of range. See [MPICompensatorConfig{}](#) for more information.

## See Also

# MPICompensatorRange

## Definition

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

## Description

|                    |                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------|
| <b>positionMin</b> | The minimum feedback position (counts) along the compensation axis where compensation will occur. |
| <b>positionMax</b> | 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:**  $\text{Points} = (\text{positionMax} - \text{positionMin}) / \text{positionDelta} + 1$

**2D Compensation:**  $\text{Points} = \text{PointsX} * \text{PointsY}$

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

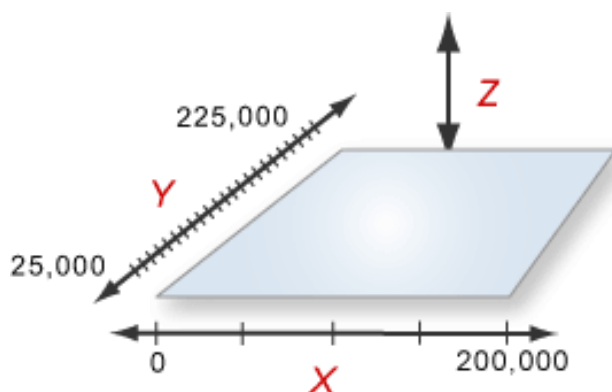
## Example: (taken from comp.c sample application)

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:

$\text{X\_DIM} = (200000 - 0) / 50000 + 1 = 5$



$$Y\_DIM = (225000-25000)/10000 + 1 = 21$$

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 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].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 previous section). Continuing with our original example lets assume that our measurements are as defined by the following table:

```
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 },
};
```

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](#).

## Methods

### Create, Delete, Validate Methods

|                                           |                         |
|-------------------------------------------|-------------------------|
| <a href="#"><u>mpiControlCreate</u></a>   | Create Control object   |
| <a href="#"><u>mpiControlDelete</u></a>   | Delete Control object   |
| <a href="#"><u>mpiControlValidate</u></a> | Validate Control object |

### Configuration and Information Methods

|                                                   |                                                              |
|---------------------------------------------------|--------------------------------------------------------------|
| <a href="#"><u>mpiControlAddress</u></a>          | Get original address of Control object (when it was created) |
| <a href="#"><u>mpiControlConfigGet</u></a>        | Get Control config                                           |
| <a href="#"><u>mpiControlConfigSet</u></a>        | Set Control config                                           |
| <a href="#"><u>meiControlExtMemAvail</u></a>      |                                                              |
| <a href="#"><u>mpiControlFlashConfigGet</u></a>   | Get Control flash config                                     |
| <a href="#"><u>mpiControlFlashConfigSet</u></a>   | Set Control flash config                                     |
| <a href="#"><u>meiControlFPGADefaultGet</u></a>   |                                                              |
| <a href="#"><u>meiControlFPGAFileOverride</u></a> |                                                              |
| <a href="#"><u>meiControlGateGet</u></a>          | Get the closed state (TRUE or FALSE)                         |
| <a href="#"><u>meiControlGateSet</u></a>          | Set the closed state (TRUE or FALSE)                         |
| <a href="#"><u>meiControlInfo</u></a>             |                                                              |
| <a href="#"><u>mpiControlIoGet</u></a>            |                                                              |

[mpiControlIoSet](#)[meiControlIoBitGet](#)[meiControlIoBitSet](#)[meiControlSampleCounter](#)[meiControlSampleRate](#)[meiControlSamplestoSeconds](#)

Converts samples to seconds

[meiControlSampleWait](#)[meiControlSecondstoSamples](#)

Converts seconds to samples

[mpiControlStatus](#)[mpiControlType](#)

Get type of Control object (used to create Command object)

## Event Methods

[mpiControlEventNotifyGet](#)[mpiControlEventNotifySet](#)[mpiControlEventReset](#)

## Memory Methods

[mpiControlMemory](#)

Get address of Control memory

[mpiControlMemoryAlloc](#)

Allocate bytes of firmware memory

[mpiControlMemoryCount](#)

Get number of bytes available in firmware

[mpiControlMemoryFree](#)

Free bytes of firmware memory

[mpiControlMemoryGet](#)

Copy count bytes of Control memory to application memory

[mpiControlMemorySet](#)

Copy count bytes of application memory to Control memory

## Relational Methods

[meiControlPlatform](#)

## Action Methods

[mpiControlCycleWait](#)

Wait for Control to execute count cycles

[mpiControlInit](#)

Initialize Control object

[mpiControlInterruptEnable](#)

Enable interrupts to Control object

[mpiControlInterruptWait](#)

Wait for controller interrupt

[mpiControlInterruptWake](#)

Wake all threads waiting for controller interrupt

[mpiControlReset](#)

Reset controller hardware

[meiControlSampleWait](#)

Specify how many samples the host waits for, while the XMP executes

[meiControlVersionMismatchOverride](#)

## Data Types



[MPIControlAddress](#)

[MPIControlConfig](#) / [MEIControlConfig](#)

[MPIControlFanStatusFlag](#)

[MPIControlFanStatusMask](#)

[MEIControlFPGA](#)

[MEIControlInfo](#)

[MEIControlInfoDriver](#)

[MEIControlInfoFirmware](#)

[MEIControlInfoFirmwareZMP](#)

[MEIControlInfoHardware](#)

[MEIControlInfoMpi](#)

[MEIControlInfoPld](#)

[MEIControlInfoRincon](#)

[MEIControlInput](#)

[MPIControlIo](#)

[MEIControlIoBit](#)

[MPIControlIoWords](#)

[MPIControlMessage](#) / [MEIControlMessage](#)

[MPIControlMemoryType](#)

[MEIControlOutput](#)

[MPIControlStatus](#)

[MEIControlTrace](#)

[MPIControlType](#)

## Constants

[MPIControlMAX\\_AXES](#)

[MPIControlMAX\\_COMPENSATORS](#)

[MPIControlMAX\\_RECORDERS](#)

[MPIControlMIN\\_AXIS\\_FRAME\\_COUNT](#)

[MEIControlSTRING\\_MAX](#)

## Macros

[mpiControlFanStatusMaskBIT](#)

# 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:

| <i>If the "type" parameter is</i> | <i>Then the form of the "address" parameter is</i> |
|-----------------------------------|----------------------------------------------------|
| 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 ( <b>NOTE:</b> address.number should be set to zero)                               |

# 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**

if *ControlDelete* successfully deletes a Control object and invalidates its handle

## 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

|                     |                                           |
|---------------------|-------------------------------------------|
| <b>MPIMessageOK</b> | if Control is a handle to a valid object. |
|---------------------|-------------------------------------------|

## 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**

if *ControlAddress* successfully writes the address (used when ***control*** was created) to the contents of ***address***

## 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

|                     |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlConfigGet</i> successfully gets the <b><i>control</i></b> configuration and writes it in the structure(s) |
|---------------------|------------------------------------------------------------------------------------------------------------------------|

## 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).

## Remarks

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

### Return Values

#### MPIMessageOK

if *ControlConfigSet* successfully writes the Control object's configuration using data from the structure(s)

## 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](#)

# 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.

|                |                                                                |
|----------------|----------------------------------------------------------------|
| <b>control</b> | a handle to the Control object                                 |
| <b>*size</b>   | a pointer to the available memory words returned by the method |

### Return Values

#### **MPIMessageOK**

if *ControlExtMemAvail* successfully gets and writes the available external memory words into *\*size*

## 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.

| Return Values |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>ControlFlashConfigGet</i> successfully gets the Control's flash configuration and writes it into the structure(s) |

## 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.

| Return Values       |                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlFlashConfigSet</i> successfully sets (writes) the Control's flash configuration using data from the structure(s) |

## 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*.

|                    |                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>control</b>     | a handle to the Control object                                                                                                                                         |
| <b>*socketInfo</b> | tells the function which type of FPGA is physically on the board.                                                                                                      |
| <b>*fpga</b>       | a pointer to a MEIControlFPGA object that contains a string that is the filename. To get the proper <i>fpga</i> , pass in <i>control</i> and valid <i>socketInfo</i> . |

| Return Values       |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlFPGADefaultGet</i> successfully gets creates a default FPGA filename. |

## 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.

|                      |                                                                                |
|----------------------|--------------------------------------------------------------------------------|
| <b>control</b>       | a handle to the Control object.                                                |
| <b>*fpga</b>         | a pointer to MEIControlFPGA struct that contains the current file name string. |
| <b>*overrideFile</b> | is a character string that contains a desired filename.                        |
| <b>*socketInfo</b>   | is a pointer to valid socket information.                                      |

| Return Values       |                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlFPGAOverride</i> successfully replaces the default FPGA file with the desired overrideFile. |

## See Also

# meiControlGateGet

## Declaration

```
long meiControlGateGet(MPIControl control,  
                        long      gate,  
                        long      *closed)
```

Required Header: stdmei.h

## Description

**meiControlGateGet** gets the closed state (TRUE or FALSE) from the specified control gate (0 to 31).

| Return Values       |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlGateGet</i> successfully gets (reads) the state from the control gate and puts it into closed. |

## See Also

[meiControlGateSet](#)

# meiControlGateSet

## Declaration

```
long meiControlGateSet(MPIControl control,  
                        long      gate,  
                        long      closed)
```

Required Header: stdmei.h

## Description

**meiControlGateSet** sets the closed state (TRUE or FALSE) for the specified control gate (0 to 31).

| Return Values       |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlGateSet</i> successfully sets (writes) the closed state into the control gate. |

## See Also

[meiControlGateGet](#)

# meiControlInfo

## Declaration

```
long  meiControlInfo(MPIControl      control,
                    MEIControlInfo  *info);
```

Required Header: stdmei.h

## Description

**meiControlInfo** retrieves information about an MEI motion controller.

|                |                                                                                              |
|----------------|----------------------------------------------------------------------------------------------|
| <b>control</b> | a handle to the Control object                                                               |
| <b>*info</b>   | a pointer to MEIControlInfo that gets completed with the appropriate controller information. |

| Return Values        |                                                                      |
|----------------------|----------------------------------------------------------------------|
| <b>MPIMessageOK</b>  | If meiControlInfo successfully retrieves the controller information. |
| <b>MPIHandleVOID</b> | if <i>control</i> is invalid                                         |

## See Also

# mpiControlIoGet

## Declaration

```
long mpiControlIoGet(MPIControl control,  
                    MPIControlIo *io);
```

Required Header: stdmpi.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.

|                |                                                                          |
|----------------|--------------------------------------------------------------------------|
| <b>control</b> | a handle to a Control object                                             |
| <b>*io</b>     | a pointer to a structure containing the digital input and output values. |

| Return Values                |                                                                                                                |
|------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>ControlIoGet</i> successfully gets the I/O bits from controller and puts (writes) them in the structure. |
| <b>MPIMessageARG_INVALID</b> | if the <i>io</i> pointer points to NULL.                                                                       |

## See Also

[mpiControlIoSet](#) | [MPIControlInput](#) | [MPIControlOutput](#)

# mpiControlIoSet

## Declaration

```
long mpiControlIoSet(MPIControl control,  
                    MPIControlIo *io);
```

Required Header: stdmpi.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.

|                |                                                                          |
|----------------|--------------------------------------------------------------------------|
| <b>control</b> | a handle to a Control object                                             |
| <b>*io</b>     | a pointer to a structure containing the digital input and output values. |

| Return Values       |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlIoSet</i> successfully writes the states of a controller's digital I/O |

## See Also

[mpiControlIoGet](#) | [MPIControlInput](#) | [MPIControlOutput](#)

# meiControlIoBitGet

## Declaration

```
long  meiControlIoBitGet(MPIControl      control,  
                        MEIControlIoBit bit,  
                        long          *value);
```

Required Header: stdmei.h

## 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

|                |                                                               |
|----------------|---------------------------------------------------------------|
| <b>control</b> | a handle to the Control object                                |
| <b>bit</b>     | an enumerated bit number                                      |
| <b>*value</b>  | a pointer to a long. The value contains the state of the bit. |

| Return Values       |                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlIoBitGet</i> successfully gets the Control configuration and writes it into the structure(s). |

## See Also

[meiControlIoBitSet](#) | [MEIControlIoBit](#)

# meiControlIoBitSet

## Declaration

```
long  meiControlIoBitSet(MPIControl      control,  
                        MEIControlIoBit bit,  
                        long      *value);
```

Required Header: stdmei.h

## 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

|                |                                                               |
|----------------|---------------------------------------------------------------|
| <b>control</b> | a handle to the Control object                                |
| <b>bit</b>     | an enumerated bit number                                      |
| <b>*value</b>  | a pointer to a long. The value contains the state of the bit. |

| Return Values       |                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlIoBitSet</i> successfully sets the Control configuration and writes it into the structure(s). |

## See Also

[meiControlIoBitGet](#) | [MEIControlIoBit](#)



# 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  | if the sample counter could be read |

## 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.

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <b>control</b>     | a handle to the Control object                                    |
| <b>*sampleRate</b> | pointer to a double where the current sample rate will be stored. |

| Return Values                                 |                                                                                                                                                             |
|-----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                           | if <i>ControlSampleRate</i> successfully writes the current sample rate (Hz) of the controller's processor to the address pointed to by <i>sampleRate</i> . |
| <b>MEIControlMessageFIRMWARE_VERSION_NONE</b> | The controller firmware version is zero.                                                                                                                    |
| <b>MEIControlMessageFIRMWARE_VERSION</b>      | The controller firmware version does not match the software 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       |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlSamplesToSeconds</i> successfully converts the samples to seconds. |

## See Also

[meiControlSecondstoSamples](#) | [meiControlSampleCounter](#)

# meiControlSampleWait

## Declaration

```
long meiControlSampleWait(MPIControl control,  
                           long count)
```

Required Header: stdmei.h

## Description

**meiControlSampleWait** waits for *count* samples while the XMP motion controller (associated with *control*) executes. While the host waits, the host gives up its time slice and continuously verifies that the XMP firmware is operational.

| Return Values       |                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlSampleWait</i> successfully waits for count samples while the XMP motion controller executes |

## See Also

[meiControlSampletoSeconds](#) | [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

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlSecondsToSamples</i> successfully converts the seconds to samples. |
|---------------------|---------------------------------------------------------------------------------|

## See Also

[meiControlSamplestoSeconds](#) | [meiControlSampleCounter](#) | [meiControlSampleWait](#)

# mpiControlStatus

## Structure

```
long mpiControlStatus(MPIControl control,  
                     MPIControlStatus *status,  
                     void *external);
```

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

## Description

**mpiControlStatus** gets a Control's (***control***) 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).

|                  |                                                    |
|------------------|----------------------------------------------------|
| <b>control</b>   | a handle to a Control object.                      |
| <b>*status</b>   | a pointer to MPIControlStatus structure.           |
| <b>*external</b> | a pointer to an implementation-specific structure. |

| Return Values                |                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>ControlStatus</i> successfully writes the Control's status to the structure(s). |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL or if the <i>external</i> pointer is not NULL    |

## See Also

[MPIControl](#) | [MPIControlStatus](#)

# 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

if *ControlType* successfully gets the type (used when ***control*** was created) to the contents of ***type***

## 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

**mpiControEventNotifyGet** 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.

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <b>control</b>    | a handle to the Control object                   |
| <b>*eventMask</b> | pointer to MPIEventMask structure.               |
| <b>*external</b>  | pointer to MEIEventNotifyData structure or NULL. |

| Return Values                |                                                                                                                                                                            |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>ControlEventNotifyGet</i> successfully fills in the <i>eventMask</i> with the data indicating which control events will cause the firmware to generate an interrupt. |
| <b>MPIMessageARG_INVALID</b> | if eventMask is NULL                                                                                                                                                       |

## 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.

|                  |                                                  |
|------------------|--------------------------------------------------|
| <b>control</b>   | a handle to the Control object                   |
| <b>eventMask</b> | MPIEventMask structure.                          |
| <b>*external</b> | pointer to MEIEventNotifyData structure or NULL. |

| Return Values                |                                                                                              |
|------------------------------|----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>ControlEventNotifySet</i> successfully configures the firmware to generate interrupts. |
| <b>MPIMessageARG_INVALID</b> | if eventMask is NULL                                                                         |

## 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

**mpiControEventReset** 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.

|                  |                                         |
|------------------|-----------------------------------------|
| <b>control</b>   | a handle to the Control object          |
| <b>eventMask</b> | <a href="#">MPIEventMask</a> structure. |

| Return Values                |                                                                      |
|------------------------------|----------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>ControlEventReset</i> successfully resets (clears) the events. |
| <b>MPIMessageARG_INVALID</b> | if eventMask is NULL.                                                |

## See Also

[mpiControlEventNotifyGet](#) | [mpiControlEventNotifySet](#) | [MPIEventMask](#)

# 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

if *ControlMemory* successfully writes the address(es) (used to access Control memory, and optionally to access another section of Control memory) to the contents of ***memory*** (and to ***external***, if ***external*** is not Null)

## 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  | if <i>ControlMemoryAlloc</i> successfully allocates firmware memory and writes the host address of that firmware memory to <i>memory</i> . |

## 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       |                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlMemoryCount</i> successfully writes the number of bytes of firmware memory (that are available to be allocated) to <i>count</i> . |

## 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       |                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlMemoryFree</i> successfully frees <i><b>count</b></i> bytes of firmware memory on a Control object |

## See Also

[mpiControlMemoryGet](#) | [mpiControlMemorySet](#) | [mpiControlMemoryAlloc](#) | [mpiControlMemoryCount](#) | [mpiControlMemory](#)

# mpiControlMemoryGet

## Declaration

```
long mpiControlMemoryGet(MPIControl control,
                        void *dst,
                        void *src,
                        long count)
```

**Required Header:** stdmpi.h

## 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

if *ControlMemoryGet* successfully gets *count* bytes of *control* memory and puts (writes) them in application memory

## 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,
                        void      *src,
                        long      count)
```

**Required Header:** stdmpi.h

## Description

**mpiControlMemorySet** sets (writes) *count* bytes of application memory (starting at address *src*) to *control* memory (starting at address *dst*).

### Return Values

#### MPIMessageOK

if *ControlMemorySet* successfully sets (writes) count bytes of application memory to control memory

## 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](#)

# 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.

|                |                                |
|----------------|--------------------------------|
| <b>control</b> | a handle to the Control object |
|----------------|--------------------------------|

| Return Values        |                              |
|----------------------|------------------------------|
| <b>MPIPlatform</b>   | handle to a Platform object  |
| <b>MPIHandleVOID</b> | if <i>control</i> is invalid |

## See Also

[mpiControlCreate](#)

# mpiControlCycleWait

## Declaration

```
long mpiControlCycleWait(MPIControl control,  
                        long count)
```

Required Header: stdmpi.h

## Description

**mpiControlCycleWait** 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       |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | after the motion controller successfully executes for <i><b>count</b></i> cycles |

## See Also

# mpiControlInit

## Declaration

```
long mpiControlInit(MPIControl control)
```

**Required Header:** stdmpi.h

## 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.

|                |                                |
|----------------|--------------------------------|
| <b>control</b> | a handle to the Control object |
|----------------|--------------------------------|

### Return Values

|                                                         |                                                                                  |
|---------------------------------------------------------|----------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                                     | if <i>ControlInit</i> successfully initializes the motion control device control |
| <a href="#">MPIControlMessageLIBRARY_VERSION</a>        | See Descriptions of Return Values.                                               |
| <a href="#">MPIControlMessageADDRESS_INVALID</a>        | See Descriptions of Return Values.                                               |
| <a href="#">MPIControlMessageCONTROL_INVALID</a>        | See Descriptions of Return Values.                                               |
| <a href="#">MPIControlMessageTYPE_INVALID</a>           | See Descriptions of Return Values.                                               |
| <a href="#">MPIControlMessageCONTROL_NUMBER_INVALID</a> | See Descriptions of Return Values.                                               |
| <a href="#">MEIControlMessageFIRMWARE_INVALID</a>       | See Descriptions of Return Values.                                               |
| <a href="#">MEIControlMessageSYNQNET_STATE</a>          | See Descriptions of Return Values.                                               |
| <a href="#">MEIPacketMessageADDRESS_INVALID</a>         | See Descriptions of Return Values.                                               |
| <a href="#">MEIPlatformMessageDEVICE_INVALID</a>        | See Descriptions of Return Values.                                               |
| <a href="#">MEIPlatformMessageDEVICE_MAP_ERROR</a>      | See Descriptions of Return Values.                                               |
| <a href="#">MEISynqNetMessageSTATE_ERROR</a>            | See Descriptions of Return Values.                                               |

## Sample Code

```
    MPIControl    control; /* motion controller object handle */

    control =
        mpiControlCreate(MPIControlTypeDEFAULT, NULL);
    mpiControlValidate(control);

/* Initialize motion controller */
returnValue =
    mpiControlInit(control);
```

## See Also

[mpiControlCreate](#) | [mpiControlDelete](#)

# 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**

if *ControlInterruptEnable* successfully enables (or disables) interrupts from the motion controller

## See Also

[mpiControlInteruptWait](#) | [mpiControlInteruptWake](#)



# mpiControlInterruptWait

## Declaration

```
long mpiControlInterruptWait(MPIControl control,  
                             long      *interrupted,  
                             MPIWait  timeout)
```

Required Header: stdmpi.h

## 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            |                                                                                       |
|--------------------------|---------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>      | if <i>ControlInterruptWait</i> waits for an interrupt from the motion controller      |
| <b>MPIMessageTIMEOUT</b> | if <i>ControlInterruptWait</i> did not receive an interrupt within <i>timeout</i> ms. |

## 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**

if *ControlInterruptWake* successfully wakes all threads waiting for an interrupt from the motion controller

## See Also

[mpiControlInterruptWait](#) | [mpiControlInteruptEnable](#)

# mpiControlReset

## Declaration

```
long mpiControlReset(MPIControl control)
```

**Required Header:** stdmpi.h

## Description

**mpiControlReset** resets the motion controller (***control***) board.

### Return Values

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ControlReset</i> successfully resets the motion controller board |
|---------------------|------------------------------------------------------------------------|

## See Also

# mpiControlVersionMismatchOverride

## Declaration

```
long meiControlVersionMismatchOverride(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

| Return Values       |                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if the motion controller successfully overrides the version mismatch between the MPI and the controller. |

## See Also

# MPIControlAddress

## Definition

```
typedef struct MPIControlAddress {
    long    number;    /* controller number */

    union {
        void                *mapped;    /* memory address */
        unsigned long       ioPort;    /* I/O port number */
        char                *device;    /* device driver name */
        struct {
            char            *name;    /* image file name */
            MPIControlFileType type;    /* image file type */
        } file;
        struct {
            char    *server;    /* IP address: host.domain.com */
            long    port;    /* socket number */
        } client;
    } type;
} MPIControlAddress;
```

## 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.

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <b>number</b> | The controller number in the computer                                    |
| <b>type</b>   | 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    cmdDacCount;
    long    auxDacCount;
    long    filterCount;
    long    motionCount;
    long    motorCount;
    long    recorderCount;
    long    recordCount[MPIControlMAX_RECORDERS];
    long    sequenceCount;
    long    userVersion;
    long    sampleRate;
} MPIControlConfig;
```

## 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.

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>axisCount</b>             | 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 <i>axisCount</i> equal to the <i>motorCount</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>axisFrameCount[]</b>      | 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. The axis frame buffer size must be a power of 2 (16, 32, 64, etc). 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 16 to the available memory. Use <a href="#">meiControlExtMemAvail(...)</a> to determine the controller's available memory. Be sure to leave some free memory for potential future features.                              |
| <b>captureCount</b>          | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>compareCount</b>          | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>compensatorCount</b>      | This value defines the number of enabled compensators.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>compensatorPointCount</b> | <p>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 <a href="#">Determining Required Compensator Table Size</a> for more information.</p> <p>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 <i>compensatorPointCount</i>. When using compensator objects, see <a href="#">Determining Required Compensator Table Size</a> for more information on a proper value for the point count.</p> |
| <b>cmdDacCount</b>           | Number of command DACs (digital to analog converter) enabled for the controller. The controller's cmdDac transmits and scales a torque demand value to a SynqNet servo drive or a physical DAC circuit. There is one cmdDac per motor. Normally, the cmdDacCount should be equal to the motorCount.                                                                                                                                                                                                                                                                                                                                                                                                                                     |

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>auxDacCount</b>   | Number of auxilliary DACs (digital to analog converter) enabled for the controller. The controller's auxDac transmits and scales a torque demand value to a SynqNet servo drive or a physical DAC circuit. Auxilliary DACs can be used for sinusoidal motor commutation, where the cmdDac and auxDac provide the commutation phases. Or, auxilliary DACs can be used for general purpose analog outputs. There is one auxDac per motor.                                                                                                                                                                                                                                                                              |
| <b>filterCount</b>   | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>motionCount</b>   | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>motorCount</b>    | 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).                                                                                                                                                                                                                                                                                                                                                                                |
| <b>recorderCount</b> | 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.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>recordCount</b>   | 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 <a href="#">meiControlExtMemAvail(.)</a> 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. |
| <b>sequenceCount</b> | Number of sequence objects enabled for the controller. The controller's sequence object executes and manages a sequence of pre-compiled controller commands.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>userVersion</b>   | 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

## Definition: MEIControlConfig

```
typedef struct MEIControlConfig {
    long          preFilterCount;
    long          TxTime;
    long          syncInterruptPeriod;
    MEIXmpPreFilter  PreFilter[MEIXmpMAX_PreFilters];
    MEIXmpUserBuffer UserBuffer;
} MEIControlConfig;
```

**Change History:** Modified in the 03.02.00

## Description

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>preFilterCount</b>      | This value defines the number of enabled pre-filters.                                                                                                                                                                                                                                                                                                                                                                         |
| <b>TxTime</b>              | 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. |
| <b>syncInterruptPeriod</b> | samples/interrupt. Configures the controller to send a hardware interrupt to host computer every $n$ controller samples.<br><br>0 = disabled, 1 = every sample, 2 = every other sample, etc...                                                                                                                                                                                                                                |
| <b>PreFilter</b>           | This array defines the configuration for each pre-filter.                                                                                                                                                                                                                                                                                                                                                                     |
| <b>UserBuffer</b>          | 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](#) | [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.

|                                                    |                                                                           |
|----------------------------------------------------|---------------------------------------------------------------------------|
| <b>MPIControlFanStatusFlagSTATUS_NOT_AVAILABLE</b> | Value specifies that the fan status is not available for your controller. |
| <b>MPIControlFanStatusFlagFAN_OK</b>               | Value specifies that the fan is fine.                                     |
| <b>MPIControlFanStatusFlagFAN_ERROR</b>            | Value specifies there is a fan error.                                     |
| <b>MPIControlFanStatusFlagOVER_TEMP_LIMIT</b>      | 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.

|                                                    |                                                                                 |
|----------------------------------------------------|---------------------------------------------------------------------------------|
| <b>MPIControlFanStatusMaskNONE</b>                 | Bit mask containing none of the ControlStatusFlags set.                         |
| <b>MPIControlFanStatusMaskSTATUS_NOT_AVAILABLE</b> | Fan status is not available or supported by hardware.                           |
| <b>MPIControlFanStatusMaskFAN_OK</b>               | Fan status is supported and there are no fan errors or temperature over limits. |

|                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIControlFanStatusMaskFAN_ERROR</b>       | <p>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.</p> <p>The cause of a FAN_ERROR is hardware dependent.</p> <p><b>For ZMP-Series using an ADM1030 fan controller, possible causes are:</b><br/>ALARM_SPEED,<br/>FAN_FAULT, and/or<br/>REMOTE_DIODE_ERROR Flags are set.</p> <p>Please refer to the ADM1030 specifications for more information.</p>                                                                                   |
| <b>MPIControlFanStatusMaskOVER_TEMP_LIMIT</b> | <p>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.</p> <p>The cause of an OVER_TEMP_LIMIT is hardware dependent.</p> <p><b>For ZMP-Series using an ADM1030 fan controller, possible causes are:</b><br/>REMOTE_TEMP_HIGH,<br/>LOCAL_TEMP_HIGH, and/or<br/>OVER_TEMP_LIMIT Flags are set.</p> <p>Please refer to the ADM1030 specifications for more information.</p> |
| <b>MPIControlFanStatusMaskALL</b>             | <p>Bit mask containing all of the ControlStatusFlags set.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

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.

|                 |                 |
|-----------------|-----------------|
| <b>Filename</b> | character array |
|-----------------|-----------------|

## See Also

[meiControlFPGADefaultGet](#) | [meiControlFPGAFileOverride](#)



# MEIControlInfo

## Definition

```
typedef struct MEIControlInfo {
    MEIControlInfoMpi      mpi;
    MEIControlInfoFirmware firmware;
    MEIControlInfoPld      pld;
    MEIControlInfoRincon   rincon;
    MEIControlInfoHardware hardware;
    MEIControlInfoDriver   driver;
}MEIControlInfo;
```

## Description

**MEIControlInfo** contains the information about the motion controller being used.

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <b>mpi</b>      | Information about the MPI software located on the host computer.                          |
| <b>firmware</b> | Information about the Firmware running on the controller.                                 |
| <b>pld</b>      | Information about the PLD located in the controller.                                      |
| <b>rincon</b>   | Information about the Rincon FPGA located on the controller.                              |
| <b>hardware</b> | Production information about the hardware stored in the controller.                       |
| <b>driver</b>   | Information about the Driver, running on the host, used to interface with 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.

| <b>version</b> |
|----------------|
|----------------|

|                                                                           |
|---------------------------------------------------------------------------|
| 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.

|                    |                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>version</b>     | The major version number for the controller's firmware. To be compatible with the MPI library, this number must match the fwVersion in the <a href="#">MEIControlInfoMpi</a> structure.                                                                                                                                                                                         |
| <b>option</b>      | 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.                                                                                                                                                                                                                  |
| <b>revision</b>    | The minor version number for the controller's firmware. Indicates a minor change or bug fix to the firmware code.                                                                                                                                                                                                                                                               |
| <b>subRevision</b> | The micro version value for the controller's firmware. Indicates a very minor change or bug fix to the firmware code.                                                                                                                                                                                                                                                           |
| <b>branchId</b>    | 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. |
| <b>zmp</b>         | ZMP-only information. Contains versions and revision info for boot0 and zboot code.                                                                                                                                                                                                                                                                                             |

## See Also

# 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

# MEIControlInfoHardware

## Definition

```
typedef struct MEIControlInfoHardware {
    char    modelName[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.

|                     |                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------|
| <b>modelName</b>    | The Controller's model number or t-level number (ex: T001-0001) which is stored on the hardware. |
| <b>serialNumber</b> | The Controller's serial number, which is unique to each controller.                              |
| <b>type</b>         | The type of Controller (XMP or ZMP).                                                             |

## See Also

# MEIControlInfoMpi

## Definition

```
typedef struct MEIControlInfoMpi {
    char        version[MEIControlSTRING_MAX];
    long        fwVersion;
    long        fwOption;
} MEIControlInfoMpi;
```

## Description

**MEIControlInfoMpi** is a structure that contains read-only version information for the MPI.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>version</b>   | 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.                                                                                                                                                                                                                                                                                                                         |
| <b>fwVersion</b> | 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. |
| <b>fwOption</b>  | 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.

|                |                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>version</b> | 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.                                                                                                                                                                                                                                       |
| <b>option</b>  | 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 | <p>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.</p> <p>Existing types are:</p> <p><b>9201</b> - Rincon for XMP, XC2S100, PQ208 package</p> <p><b>9601</b> - Rincon for XMP, XC2S100, FG256 package</p> <p><b>A102</b> - RinconZ for ZMP, XC2S300E, FT256 package</p> <p><b>A301</b> - RinconZ for ZMP, XC3S200, FT256 package</p> <p>The package and version data can be used to create the FPGA filename. For example, 221_9201.fpg is Rincon type 9201, version 221.</p> |

## 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

[mpiControlloGet](#) | [mpiControlloSet](#) | [MEIControlOutput](#)

# MPIControlInfo

## Definition

```
typedef struct MPIControlIo {  
    unsigned long    input[MPIControlIoWords];  
    unsigned long    output[MPIControlIoWords]  
} MPIControlIo;
```

## Description

**MPIControlInfo** 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.

|               |                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------|
| <b>input</b>  | An array of digital input values. Each bit mask is defined by the MEIControlInput enumeration.   |
| <b>output</b> | An array of digital output values. Each bit mask is defined by the MEIControlOutput enumeration. |

## See Also

[MEIControlOutput](#) | [MEIControlInput](#) | [mpiControlIoGet](#) | [mpiControlIoSet](#)

# 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.

|                                  |                                                                                                                                                                                                     |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIControlIoBitUSER_0_IN</b>  | controller's local input, bit number 0                                                                                                                                                              |
| <b>MEIControlIoBitUSER_1_IN</b>  | controller's local input, bit number 1                                                                                                                                                              |
| <b>MEIControlIoBitUSER_2_IN</b>  | controller's local input, bit number 2                                                                                                                                                              |
| <b>MEIControlIoBitUSER_3_IN</b>  | controller's local input, bit number 3                                                                                                                                                              |
| <b>MEIControlIoBitUSER_4_IN</b>  | controller's local input, bit number 4                                                                                                                                                              |
| <b>MEIControlIoBitUSER_5_IN</b>  | controller's local input, bit number 5                                                                                                                                                              |
| <b>MEIControlIoBitXESTOP</b>     | controller's local input, External Emergency Stop Input.<br><br><b>NOTE:</b> The XESTOP bit does not have any special functionality. The bit number and name were kept for backwards compatibility. |
| <b>MEIControlIoBitUSER_0_OUT</b> | controller's local output, bit number 0                                                                                                                                                             |

|                                  |                                         |
|----------------------------------|-----------------------------------------|
| <b>MEIControlIoBitUSER_1_OUT</b> | controller's local output, bit number 1 |
| <b>MEIControlIoBitUSER_2_OUT</b> | controller's local output, bit number 2 |
| <b>MEIControlIoBitUSER_3_OUT</b> | controller's local output, bit number 3 |
| <b>MEIControlIoBitUSER_4_OUT</b> | controller's local output, bit number 4 |
| <b>MEIControlIoBitUSER_5_OUT</b> | controller's local output, bit number 5 |

## See Also

[meiControlIoBitGet](#)

# MPIControlIoWords

## 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](#)

# 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,
    MPIControlMessageCMDDAC_COUNT_INVALID,
    MPIControlMessageAUXDAC_COUNT_INVALID,
    MPIControlMessageFILTER_COUNT_INVALID,
    MPIControlMessageMOTION_COUNT_INVALID,
    MPIControlMessageMOTOR_COUNT_INVALID,
    MPIControlMessageSAMPLE_RATE_TO_LOW,
    MPIControlMessageRECORDER_COUNT_INVALID,
    MPIControlMessageCOMPENSATOR_COUNT_INVALID,
    MPIControlMessageAXIS_RUNNING,
    MPIControlMessageRECORDER_RUNNING,
} MPIControlMessage;
```

## Description

**MPIControlMessage** is an enumeration of Motor 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\_Compare.

**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 1000. SynqNet does not support cyclic data rates below 1kHz. The controller's sample rate specifies the SynqNet cyclic rate.

**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.



## 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**

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 [meiControlIoBitGet\(...\)](#) or [meiControlIoBitSet\(...\)](#) 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.

|                                    |                                                                                               |
|------------------------------------|-----------------------------------------------------------------------------------------------|
| <b>MPIControlMemoryTypeUSER</b>    | The dynamic portion of the controller's external memory that is not in use by the controller. |
| <b>MPIControlMemoryTypeDEFAULT</b> | 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

[mpiControlIoGet](#) | [mpiControlIoSet](#) | [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](#)

# MEIControlTrace

## Definition

```
typedef enum {  
    MEIControlTraceDYNA_ALLOC = MEIControlTraceFIRST << 0,  
} MEIControlTrace;
```

## Description

**MEIControlTrace** is an enumeration of control object trace bits to enable debug tracing.

|                                  |                                                                                       |
|----------------------------------|---------------------------------------------------------------------------------------|
| <b>MEIControlTraceDYNA_ALLOC</b> | 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](#)

# 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 (8)
```

## 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 the minimum allowed value for which `MPIControlConfig.axisFrameCount` can be set.

## See Also

[MPIControlConfig](#) | [mpiControlConfigGet](#) | [mpiControlConfigSet](#)

# MEIControlSTRING\_MAX

## Definition

```
#define MEIControlSTRING_MAX (16)
```

## 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** defines the

## See Also

## *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](#)

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        |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to a Control object                |
| <b>MPIHandleVOID</b> | if the object could not be created |

## Sample Code

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);
```

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`.

## See Also

[MPIControl](#) | [MPIControlAddress](#) | [MPIControlType](#) | [mpiControlValidate](#)  
[mpiControlInit](#) | [mpiControlDelete](#)

# DriveMap Objects

## Introduction

A **DriveMap** object is used to access information in a drive map file.

## Methods

### Create, Delete, Validate Methods

[meiDriveMapCreate](#)

[meiDriveMapDelete](#)

[meiDriveMapValidate](#)

### Configuration and Information Methods

[meiDriveMapParamCount](#)

[meiDriveMapParamList](#)

[meiDriveMapConfigCount](#)

[meiDriveMapConfigList](#)

## Data Types

[MEIDriveMapMessage](#)

[MEIDriveMapParamAccess](#)

[MEIDriveMapParamInfo](#)

[MEIDriveMapParamType](#)

[MEIDriveMapParamValue](#)

## Constants

[MEIDriveMapParamMAX\\_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.

|                  |                     |
|------------------|---------------------|
| <b>*filename</b> | the drive map file. |
|------------------|---------------------|

| Return Values        |                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------|
| <b>handle</b>        | to a DriveMap object. After creating a DriveMap object, it must be validated using meiDriveMapValidate(). |
| <b>MPIHandleVOID</b> | 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.

|                 |                                                                                       |
|-----------------|---------------------------------------------------------------------------------------|
| <b>driveMap</b> | a handle of the DriveMap object to delete in the reverse order to avoid memory leaks. |
|-----------------|---------------------------------------------------------------------------------------|

| Return Values       |                                                           |
|---------------------|-----------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>DriveMapDelete</i> successfully deleted the object. |

## 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.

|                 |                                |
|-----------------|--------------------------------|
| <b>driveMap</b> | a handle to a DriveMap object. |
|-----------------|--------------------------------|

| Return Values       |                                                   |
|---------------------|---------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>driveMap</i> is a handle to a valid object. |

## 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.

|                        |                                                                                                                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>driveMap</b>        | a handle to a DriveMap object.                                                                                                                                                   |
| <b>nodeName</b>        | the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retrieved by calling meiSqNodeInfo. See <a href="#">MEISqNodeInfo</a> . |
| <b>firmwareVersion</b> | The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See <a href="#">MEISqNodeDriveInfo</a> . |
| <b>*paramsCount</b>    | pointer to the variable that will be set by this function.                                                                                                                       |

| Return Values       |                                                                                                                                                                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiDriveMapParamCount</i> successfully scans the drive map file for a drive entry that matches this node on the network and returns the number of drive parameters that need to be preserved for the configuration of the drive. |

## 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.

|                         |                                                                                                                                                                                  |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>driveMap</b>         | a handle to a DriveMap object.                                                                                                                                                   |
| <b>*nodeName</b>        | the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retrieved by calling meiSqNodeInfo. See <a href="#">MEISqNodeInfo</a> . |
| <b>*firmwareVersion</b> | The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See <a href="#">MEISqNodeDriveInfo</a> . |
| <b>paramCount</b>       | the number of drive parameter information records that can be written to the driveParamInfo list.                                                                                |
| <b>*driveParamInfo</b>  | pointer to the list of drive parameter information records that will be filled in by this function.                                                                              |

| Return Values       |                                                                                                                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiDriveMapParamList</i> successfully scans the drive map file for an entry that matches the node on the network and writes the information about each of the drive parameters to the <i>driveParamInfo</i> list. |

## 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 retrieved by calling meiSqNodeInfo. See <a href="#">MEISqNodeInfo</a> . |
| firmwareVersion | The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See <a href="#">MEISqNodeDriveInfo</a> . |
| *configCount    | pointer to the variable that will be set by this function.                                                                                                                       |

| Return Values |                                                                                                                                                                                                                                         |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>meiDriveMapConfigCount</i> successfully scans the drive map file for a drive entry that matches this node on the network and returns the number of drive parameters that need to be preserved for the configuration of the drive. |

## 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.

|                         |                                                                                                                                                                                  |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>driveMap</b>         | a handle to a DriveMap object.                                                                                                                                                   |
| <b>*nodeName</b>        | the product/manufacturing text string of the node to search for. The nodeName of an SqNode object can be retrieved by calling meiSqNodeInfo. See <a href="#">MEISqNodeInfo</a> . |
| <b>*firmwareVersion</b> | The firmware version of the drive to search for. This information can be retrieved from an SqNode object by calling meiSqNodeDriveInfo. See <a href="#">MEISqNodeDriveInfo</a> . |
| <b>configCount</b>      | the number of drive parameter information records that can be written to the configList list.                                                                                    |
| <b>*configList</b>      | pointer to the list of drive parameters that make up the drive configuration that will be filled in by this function.                                                            |

| Return Values       |                                                                                                                                                                                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiDriveMapConfigList</i> successfully scans the drive map file for a drive entry that matches this node on the network and returns the list of drive parameters that need to be preserved for the configuration of the drive. |



## 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 MEISqNodeDriveParamInfo {
    unsigned long      parameter;
    char              name[MEISqNodeDriveParamMAX_STRING_LENGTH];
    MEIDriveMapParamAccess access; /* 0=>rw, 1=>ro */
    MEIDriveMapParamType type;
    char              validValues[MEISqNodeDriveParamMAX_STRING_LENGTH];
    long              defaultValue;
    char              help[MEISqNodeDriveParamMAX_STRING_LENGTH];
} MEIDriveMapParamInfo;
```

## Description

**MEIDriveMapParamInfo** holds a set of information describing a drive parameter. This structure is read from the drives map file "drives.dm."

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>parameter</b>    | the number used to address this drive parameter.                                |
| <b>name</b>         | a null terminated string giving a name for this drive parameter.                |
| <b>access</b>       | defines if this drive parameter is read-only or read-write.                     |
| <b>type</b>         | the data type for this drive parameter. (ex: signed32, unsigned32, float, etc.) |
| <b>validValues</b>  | a string describing the possible valid values for this drive parameter.         |
| <b>defaultValue</b> | The factory default value this drive parameter.                                 |
| <b>help</b>         | A string describing this drive parameter.                                       |

## See Also

[MEIDriveMapParamType](#)

# MEIDriveMapParamType

## Definition

```
typedef enum MEIDriveParamType {
    MEISqNodeDriveParamTypeSIGNED32,
    MEISqNodeDriveParamTypeSIGNED16,
    MEISqNodeDriveParamTypeSIGNED8,
    MEISqNodeDriveParamTypeUNSIGNED32,
    MEISqNodeDriveParamTypeUNSIGNED16,
    MEISqNodeDriveParamTypeUNSIGNED8,
    MEISqNodeDriveParamTypeHEX,
    MEISqNodeDriveParamTypeENUMERATED,
    MEISqNodeDriveParamTypeMASK,
    MEISqNodeDriveParamTypeCHARACTER,
    MEISqNodeDriveParamTypeSTRING,
    MEISqNodeDriveParamTypeSINGLE,
    MEISqNodeDriveParamTypeACTION,
} MEIDriveMapParamType;
```

## Description

**MEIDriveMapParamType** is an enumeration that indicates which data format should be used with the drive parameter.

|                                          |                                                                     |
|------------------------------------------|---------------------------------------------------------------------|
| <b>MEISqNodeDriveParamTypeSIGNED32</b>   | A signed 32bit integer.                                             |
| <b>MEISqNodeDriveParamTypeSIGNED16</b>   | A signed 16bit integer.                                             |
| <b>MEISqNodeDriveParamTypeSIGNED8</b>    | A signed 8bit integer.                                              |
| <b>MEISqNodeDriveParamTypeUNSIGNED32</b> | An unsigned 32bit integer.                                          |
| <b>MEISqNodeDriveParamTypeUNSIGNED16</b> | An unsigned 16bit integer.                                          |
| <b>MEISqNodeDriveParamTypeUNSIGNED8</b>  | An unsigned 8bit integer.                                           |
| <b>MEISqNodeDriveParamTypeHEX</b>        | An integer representred in hexadecimal notation.                    |
| <b>MEISqNodeDriveParamTypeENUMERATED</b> | An integer where each value can be represented by a different name. |
| <b>MEISqNodeDriveParamTypeMASK</b>       | An integer where each bit has a unique name.                        |
| <b>MEISqNodeDriveParamTypeCHARACTER</b>  | An single ASCII character.                                          |
| <b>MEISqNodeDriveParamTypeSTRING</b>     | A null terminated string of characters.                             |

|                               |                                                                                                    |
|-------------------------------|----------------------------------------------------------------------------------------------------|
| MEISqNodeDriveParamTypeSINGLE | A single precision floating point number.                                                          |
| MEISqNodeDriveParamTypeACTION | 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;  
    float           single;  
} MEIDriveMapParamValue;
```

## 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](#)

# MEIDriveMapParamMAX\_STRING\_LENGTH

## Declaration

```
#define MEIDriveMapParamMAX_STRING_LENGTH (256)
```

**Required Header:** stdmpi.h

## Description

**MEIDriveMapParamMAX\_STRING\_LENGTH** macro defines the maximum length of a string that can be read from or written to a drive parameter.

## 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.

## Methods

### Configuration and Information Methods

|                                          |                  |
|------------------------------------------|------------------|
| <a href="#"><u>mpiEventStatusGet</u></a> | Get Event status |
| <a href="#"><u>mpiEventStatusSet</u></a> | Set Event status |

## Data Types

[MPIEventMessage](#)  
[MEIEventNotifyData](#)  
[MPIEventStatus](#)  
[MEIEventStatusInfo](#)  
[MPIEventType](#) / [MEIEventType](#)

## Constants

|                                                     |                                                      |
|-----------------------------------------------------|------------------------------------------------------|
| <a href="#"><u>MPIEventStatusINFO_COUNT_MAX</u></a> | 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

if *EventStatusGet* successfully gets the status of an Event object and writes it into the structure

## See Also

[mpiEventStatusSet](#) | [meiEventStatusInfo](#)

# 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       |                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventStatusSet</i> successfully sets (writes) the status of event using data from the structure |

## See Also

[mpiEventStatusGet](#) | [meiEventStatusInfo](#)

# 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(...)<sup>†</sup>.

The address array contains host-based XMP addresses, the contents of which are returned in MEIEventStatusInfo{}.data.

<sup>†</sup> **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.

|                |                                                                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>    | identifies the type of event that was generated.                                                                                                                                                                                       |
| <b>*source</b> | identifies what the source of the event was. source will either be a handle to an MPI object or a host pointer. Use <code>mpiObjectModuleId()</code> to identify what source points to.                                                |
| <b>info</b>    | Contains information on what generated the event and the conditions under which it was generated. <code>MEIEventStatusInfo</code> simplifies decoding this array. Sample code is shown on the <a href="#">MEIEventStatusInfo</a> 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 actualPosition;
        } axis;
        struct {
            /* Data associated with the CAN event. */
            long data[4];
        } can;
        struct {
            long sampleCounter;
            long encoderPosition;
        } motor;
        long word[MEIXmpSignalUserData];
    } data;
} MEIEventStatusInfo;
```

## Description

**MEIEventStatusInfo** is an information structure that tells the XMP what the data in MPIEventStatus.info holds.

|                                   |                                                                                                        |
|-----------------------------------|--------------------------------------------------------------------------------------------------------|
| <b>type</b>                       | A union that specifies the object handle, motion number, or external ID value that generated the event |
| <b>type.handle</b>                | A generic object handle. Used by MPIRecorder and MPIMotor events                                       |
| <b>type.axis</b>                  | An axis object handle. Used by MPIAxis events                                                          |
| <b>type.node</b>                  | The CAN Node number of the MEICan object that generated the event.                                     |
| <b>type.number</b>                | The motion number of the MPIMotion object that generated the event                                     |
| <b>type.value</b>                 | An ID value used to identify what external source or MPISequence event was generated                   |
| <b>signalID</b>                   | Specifies what type of object actually generated the event                                             |
| <b>data</b>                       | A union that contains extra data about the event that was generated                                    |
| <b>data.sampleCounter</b>         | The value of the sampleCounter when the event was generated                                            |
| <b>data.motion</b>                | A union that contains extra data about the motion event that was generated                             |
| <b>data.motion.sampleCounter</b>  | The value of the sampleCounter when the motion event was generated                                     |
| <b>data.axis</b>                  | A union that contains extra data about the axis event that was generated                               |
| <b>data.axis.sampleCounter</b>    | The value of the sampleCounter when the axis.event was generated                                       |
| <b>data.axis.actualPosition</b>   | The value of the axis' actual position when the event was generated                                    |
| <b>data.can.data</b>              | A union that contains extra data about the CAN event that was generated.                               |
| <b>data.motor</b>                 | A union that contains extra data about the motor event that was generated                              |
| <b>data.motor.sampleCounter</b>   | The value of the sampleCounter when the motor event was generated                                      |
| <b>data.motor.encoderPosition</b> | The value of the motor's ecoder position when the event was generated                                  |
| <b>data.word[]</b>                | 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.

|                                 |                                                                                              |
|---------------------------------|----------------------------------------------------------------------------------------------|
| <b>MPIEventTypeNONE</b>         | This event type indicates no event was generated.                                            |
| <b>MPIEventTypeAMP_FAULT</b>    | This event type indicates an Amp Fault event was generated from a Motor object.              |
| <b>MPIEventTypeHOME</b>         | This event type indicates a Home event was generated from a Motor object.                    |
| <b>MPIEventTypeLIMIT_ERROR</b>  | This event type indicates a position Error Limit was generated from a Motor object.          |
| <b>MPIEventTypeLIMIT_HW_NEG</b> | This event type indicates a Negative Hardware Limit event was generated from a Motor object. |

|                                       |                                                                                                  |
|---------------------------------------|--------------------------------------------------------------------------------------------------|
| <b>MPIEventTypeLIMIT_HW_POS</b>       | This event type indicates a Positive Hardware Limit event was generated from a Motor object.     |
| <b>MPIEventTypeLIMIT_SW_NEG</b>       | This event type indicates a Negative Software Limit event was generated from a Motor object.     |
| <b>MPIEventTypeLIMIT_SW_POS</b>       | This event type indicates a Positive Software Limit event was generated from a Motor object.     |
| <b>MPIEventTypeENCODER_FAULT</b>      | This event type indicates an Encoder Fault event was generated from a Motor object.              |
| <b>MPIEventTypeAMP_WARNING</b>        | This event type indicates an Amp Warning event was generated from a Motor object.                |
| <b>MPIEventTypeMOTION_DONE</b>        | This event type indicates a Motion Done event was generated from a Motion Supervisor object.     |
| <b>MPIEventTypeMOTION_AT_VELOCITY</b> | This event type indicates an At Velocity event was generated from a Motion Supervisor object.    |
| <b>MPIEventTypeRECORDER_HIGH</b>      | This event type indicates that the controller's recorded data exceeded the buffer's high limit.  |
| <b>MPIEventTypeRECORDER_FULL</b>      | This event type indicates that the controller's recorded data has filled the buffer.             |
| <b>MPIEventTypeRECORDER_DONE</b>      | This event type indicates that the controller has recorded the number of requested data records. |
| <b>MPIEventTypeEXTERNAL</b>           | This event type indicates an External event was generated from an external source.               |

## Definition: MEIEventType

```
typedef enum {
    /* System events */
    MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED =
                                                MPIEventTypeLAST, /* 16 */
    /* 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,
MEIEventTypeMOTION_RESERVED0,

/* Axis events */
MEIEventTypeIN_POSITION_COARSE,
MEIEventTypeIN_POSITION_FINE,
MEIEventTypeSETTLED
MEIEventTypeAT_TARGET,
MEIEventTypeFRAME,
MEIEventTypeAXIS_RESERVED0,
MEIEventTypeAXIS_RESERVED1,

/* 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,

/* 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 03.02.00

## Description

**MEIEventType** is used by the MPIEventMask macros to help generate event masks.

|                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED</b> | 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 <b>SynqInterrupt</b> feature and sets the <b>ProcessFlag</b> at the beginning of the foreground cycle, the firmware checks to see if the user cleared the <b>ProcessFlag</b> by the time SynqNet data is transmitted. If the ProcessFlag has not been cleared, the event occurs. |
| <b>MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED</b> | <p>This is an event that can occur when the on-board fan controller detects an error (overheating, fan failure, etc...).</p> <p><b>NOTE:</b> This is for the ZMP only and will not occur on an XMP.</p>                                                                                                                                                                                                                                                                       |
| <b>MEIEventTypeCONTROL_FAN</b>                        | <p>This is an event that can occur when the on-board fan controller detects an error (overheating, fan failure, etc...).</p> <p><b>NOTE:</b> This is for the ZMP only and will not occur on an XMP.</p>                                                                                                                                                                                                                                                                       |
| <b>MEIEventTypeLIMIT_USER0</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 0.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeLIMIT_USER1</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 1.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeLIMIT_USER2</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 2.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeLIMIT_USER3</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 3.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeLIMIT_USER4</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 4.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeLIMIT_USER5</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 5.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeLIMIT_USER6</b>                        | This event type indicates a User Limit event was generated from a Motor object. User Limit number 6.                                                                                                                                                                                                                                                                                                                                                                          |

|                                         |                                                                                                                                                                      |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIEventTypeLIMIT_USER7</b>          | This event type indicates a User Limit event was generated from a Motor object. User Limit number 7.                                                                 |
| <b>MEIEventTypeLIMIT_USER8</b>          | This event type indicates a User Limit event was generated from a Motor object. User Limit number 8.                                                                 |
| <b>MEIEventTypeLIMIT_USER9</b>          | This event type indicates a User Limit event was generated from a Motor object. User Limit number 9.                                                                 |
| <b>MEIEventTypeLIMIT_USER10</b>         | This event type indicates a User Limit event was generated from a Motor object. User Limit number 10.                                                                |
| <b>MEIEventTypeLIMIT_USER11</b>         | This event type indicates a User Limit event was generated from a Motor object. User Limit number 11.                                                                |
| <b>MEIEventTypeLIMIT_USER12</b>         | This event type indicates a User Limit event was generated from a Motor object. User Limit number 12.                                                                |
| <b>MEIEventTypeLIMIT_USER13</b>         | This event type indicates a User Limit event was generated from a Motor object. User Limit number 13.                                                                |
| <b>MEIEventTypeLIMIT_USER14</b>         | This event type indicates a User Limit event was generated from a Motor object. User Limit number 14.                                                                |
| <b>MEIEventTypeLIMIT_USER15</b>         | This event type indicates a User Limit event was generated from a Motor object. User Limit number 15.                                                                |
| <b>MEIEventTypeMOTION_OUT_OF_FRAMES</b> | This event type indicates a Motion Done event was generated from a Motion Supervisor object.                                                                         |
| <b>MEIEventTypeMOTION_RESERVED0</b>     | This event type indicates a Reserved Motion event was generated from a Motion Supervisor object. This event type is reserved for future use or custom motion events. |
| <b>MEIEventTypeIN_POSITION_COARSE</b>   | This event type indicates an In Coarse Position event was generated from an Axis object.                                                                             |
| <b>MEIEventTypeIN_POSITION_FINE</b>     | This event type indicates that an In Fine Position event was generated from an Axis object.                                                                          |

|                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetMessageREADY_TIMEOUT</b> | 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.                                        |
| <b>MEIEventTypeSETTLED</b>            | Equivalent to MEIEventTypeIN_POSITION_FINE.                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>MEIEventTypeAT_TARGET</b>          | Reserved Frame Event.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>MEIEventTypeFRAME</b>              | <b>This event type is currently not supported and is reserved for future use.</b>                                                                                                                                                                                                                                                                                                                                                          |
| <b>MEIEventTypeAXIS_RESERVED0</b>     | This event type indicates a Reserved Axis event was generated from an Axis object.<br><b>This event type is currently not supported and is reserved for future use or custom axis events.</b>                                                                                                                                                                                                                                              |
| <b>MEIEventTypeAXIS_RESERVED1</b>     | This event type indicates that a Reserved Axis event was generated from an Axis object.<br><b>This event type is currently not supported and is reserved for future use or custom axis events.</b>                                                                                                                                                                                                                                         |
| <b>MEIEventTypeSYNQNET_DEAD</b>       | 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 <a href="#">meiSynqNetEventReset(.)</a> to clear the status/event bit.                            |
| <b>MEIEventTypeSYNQNET_RX_FAILURE</b> | 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 |

|                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                         | <a href="#">meiSynqNetEventReset(.)</a> to clear the status/event bit.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>MEIEventTypeSYNQNET_TX_FAILURE</b>   | <p>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 <a href="#">meiSynqNetEventReset(.)</a> to clear the status/event bit.</p>      |
| <b>MEIEventTypeSYNQNET_NODE_FAILURE</b> | <p>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 <a href="#">meiSqNodeConfigSet(.)</a>. Use <a href="#">meiSynqNetStatus(.)</a> 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 <a href="#">meiSynqNetEventReset(.)</a> to clear the status/event bit. To recover from a node failure, the network must be shutdown and reinitialized.</p> <p><b>See Also:</b> <a href="#">SynqNet Node Failure</a></p> |
| <b>MEIEventTypeSYNQNET_RECOVERY</b>     | <p>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 <a href="#">meiSqNodeConfigSet(.)</a>. SYNQNET_RECOVERY is latched by the controller. Use <a href="#">meiSynqNetEventReset(.)</a> to clear the status/event bit.</p>                                                                                                                                                                                                                                                                        |



|                                              |                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIEventTypeSQNODE_IO_ABORT</b>           | 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 <a href="#">MEISqNodeConfigIoAbort{.}</a> for more details.                   |
| <b>MEIEventTypeSQNODE_NODE_DISABLE</b>       | 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 <a href="#">meiSqNodeEventReset(.)</a> to clear the status/event and the hardware latch.                                                                                                                                                      |
| <b>MEIEventTypeSQNODE_NODE_ALARM</b>         | 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 <a href="#">meiSqNodeEventReset(.)</a> to clear the status/event and the hardware latch.                                                             |
| <b>MEIEventTypeSQNODE_ANALOG_POWER_FAULT</b> |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>MEIEventTypeSQNODE_USER_FAULT</b>         | 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 <a href="#">meiSqNodeEventReset(.)</a> to clear the status/event bit.                                                                             |
| <b>MEIEventTypeSQNODE_NODE_FAILURE</b>       | 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 <a href="#">meiSqNodeConfigSet(.)</a> .<br>SQNODE_NODE_FAILURE is latched by the controller, use <a href="#">meiSqNodeEventReset(.)</a> to clear the status/event bit. To recover from a node failure, the network must be shutdown and reinitialized. |
| <b>MEIEventTypeCAN_BUS_STATE</b>             | The BusState has changed. Data[0] contains the new bus state.                                                                                                                                                                                                                                                                                                                                                           |
| <b>MEIEventTypeCAN_RECEIVE_OVERRUN</b>       | The CAN hardware detected a receive overrun.                                                                                                                                                                                                                                                                                                                                                                            |

|                                      |                                                                                                                                               |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIEventTypeCAN_EMERGENCY</b>     | 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. |
| <b>MEIEventTypeCAN_NODE_BOOT</b>     | A node boot message was received from a node. Data[0] contains the node number.                                                               |
| <b>MEIEventTypeCAN_HEALTH</b>        | The health of a node has changed. Data[0] contains the node number. Data[1] contains the new node health.                                     |
| <b>MEIEventTypeCAN_DIGITAL_INPUT</b> | A digital input event was received from a node. Data[0] contains the node number. Data[1 to 4] contains the new input state.                  |
| <b>MEIEventTypeCAN_ANALOG_INPUT</b>  | 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](#) | [Special Note](#) on the use of MPIEventTypeENCODER\_FAULT

# MPIEventStatusINFO\_COUNT\_MAX

## Definition

```
#define MPIEventStatusINFO_COUNT_MAX (16)
```

## Description

**MPIEventStatusINFO\_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

[mpiEventMaskALL](#) / [meiEventMaskALL](#)

Set all events within the event mask.

[mpiEventMaskAND\\_ASSIGN](#)

Assign dst all events masked by both *src* and *dst*.

[mpiEventMaskASSIGN](#)

Assign the value of event mask *src* to the event mask *dst*.

[mpiEventMaskAXIS](#) / [meiEventMaskAXIS](#)

Set all MPIAxis events within the event mask.

[mpiEventMaskBIT](#)

Set mask to only handle events of type *type*.

[mpiEventMaskBitGET](#)

Reports if a mask is set to handle events of type *type*.

[mpiEventMaskBitSET](#)

Sets mask to handle events of type *type*.

[mpiEventMaskBIT\\_POSITION](#)

Returns the bit number that is associated with MPI/MEI event type.

[mpiEventMaskBIT\\_POSITION\\_MASK](#)

Returns an element's bit-mask for the specified event type.

[meiEventMaskCAN](#)

Set all MPICan events within the event mask.

[mpiEventMaskCLEAR](#)

Set mask to handle no events.

[mpiEventMaskCOMPLEMENT](#)

Change the value of every bit within the event mask.

[mpiEventMaskCONTROL](#)

Set external events within the event mask.

[mpiEventMaskGET](#)

Reports if a mask is set to handle events of type *type*.

[mpiEventMaskIS\\_CLEAR](#)

Tests the equality of two event masks.

[mpiEventMaskIS\\_EQUAL](#)

[mpiEventMaskMOTION](#) / [meiEventMaskMOTION](#)

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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|            |                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>dst</b> | An array of unsigned longs. Use MPIEventMask to declare the dst. Each bit in the array represents a mask for a particular event. |
| <b>src</b> | 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.

|            |                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>dst</b> | An array of unsigned longs. Use MPIEventMask to declare the dst. Each bit in the array represents a mask for a particular event. |
| <b>src</b> | 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 mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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 mpiObjectEventNotifySet(...) method, which configures the controller to generate events.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event. |
| <b>type</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event. |
| <b>bit</b>  | 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.

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <b>type</b> | 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.

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <b>type</b> | 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.<br>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***.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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 associated 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event. |
| <b>type</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|              |                                                                                                                                   |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask1</b> | An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event. |
| <b>mask2</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|            |                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>dst</b> | An array of unsigned longs. Use MPIEventMask to declare the dst. Each bit in the array represents a mask for a particular event. |
| <b>src</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event. |
| <b>type</b> | 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | An array of unsigned longs. Use MPIEventMask to declare the mask. Each bit in the array represents a mask for a particular event. |
| <b>type</b> | An enumerated event type. Use an enumerated value from MPIEventType or MEIEventType.                                              |

## See Also

[MPIEventMask](#) | [MPIEventType](#) | [MEIEventType](#) | [mpiEventMaskGET](#) | [mpiEventMaskSET](#)

# meiEventMaskSYNQNQT

## Declaration

```
#define meiEventMaskSYNQNQT(mask)
```

**Required Header:** stdmei.h

## Description

**meiEventMaskSYNQNQT** 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.

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b> | 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.

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <b>type</b> | 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.

## Methods

### Create, Delete, Validate Methods

|                                            |                          |
|--------------------------------------------|--------------------------|
| <a href="#"><u>mpiEventMgrCreate</u></a>   | Create EventMgr object   |
| <a href="#"><u>mpiEventMgrDelete</u></a>   | Delete EventMgr object   |
| <a href="#"><u>mpiEventMgrValidate</u></a> | Validate EventMgr object |

### Configuration and Information Methods

|                                                    |                                                                      |
|----------------------------------------------------|----------------------------------------------------------------------|
| <a href="#"><u>mpiEventMgrConfigGet</u></a>        | Get EventMgr config                                                  |
| <a href="#"><u>mpiEventMgrConfigSet</u></a>        | Set EventMgr config                                                  |
| <a href="#"><u>mpiEventMgrEvent</u></a>            | Request event notification for all Notify objects on EventMgr's list |
| <a href="#"><u>mpiEventMgrServiceConfigGet</u></a> | Get processes that EventMgr will service                             |
| <a href="#"><u>mpiEventMgrServiceConfigSet</u></a> | Set processes that EventMgr will service                             |

### Action Methods

|                                           |                                             |
|-------------------------------------------|---------------------------------------------|
| <a href="#"><u>mpiEventMgrFlush</u></a>   | Flush pending EventMgr events               |
| <a href="#"><u>mpiEventMgrService</u></a> | Get list of all pending asynchronous events |

### Relational Methods

#### List Methods- for Control Objects

|                                                   |                                                                        |
|---------------------------------------------------|------------------------------------------------------------------------|
| <a href="#"><u>mpiEventMgrControl</u></a>         | Return handle of indexth Control object in list                        |
| <a href="#"><u>mpiEventMgrControlAppend</u></a>   | Append Control's handle to list                                        |
| <a href="#"><u>mpiEventMgrControlCount</u></a>    | Count the number of Control objects associated with EventMgr (in list) |
| <a href="#"><u>mpiEventMgrControlFirst</u></a>    | Return handle to first Control object in list                          |
| <a href="#"><u>mpiEventMgrControlIndex</u></a>    | Return the index of a Control object in list                           |
| <a href="#"><u>mpiEventMgrControlInsert</u></a>   | Insert Control handle into list                                        |
| <a href="#"><u>mpiEventMgrControlLast</u></a>     | Get handle to last Control object in list                              |
| <a href="#"><u>mpiEventMgrControlListGet</u></a>  | Get list of Control objects associated with EventMgr                   |
| <a href="#"><u>mpiEventMgrControlListSet</u></a>  | Create a list of Control objects associated with EventMgr              |
| <a href="#"><u>mpiEventMgrControlNext</u></a>     | Get handle to next Control object in list                              |
| <a href="#"><u>mpiEventMgrControlPrevious</u></a> | Get handle to previous Control object in list                          |
| <a href="#"><u>mpiEventMgrControlRemove</u></a>   | Remove a Control object's handle from list                             |

#### List Methods- for Notify Objects

|                                                |                                                           |
|------------------------------------------------|-----------------------------------------------------------|
| <a href="#"><u>mpiEventMgrNotify</u></a>       | Return handle to a Notify object associated with EventMgr |
| <a href="#"><u>mpiEventMgrNotifyAppend</u></a> | Append Notify object to list                              |
| <a href="#"><u>mpiEventMgrNotifyCount</u></a>  | Return number of Notify objects in list                   |
| <a href="#"><u>mpiEventMgrNotifyFirst</u></a>  | Get first Notify object in list                           |

[mpiEventMgrNotifyIndex](#)

Get index value for a Notify object in list

[mpiEventMgrNotifyInsert](#)

Place a Notify object after another Notify object in list

[mpiEventMgrNotifyLast](#)

Get handle to the Notify object that is last on the list

[mpiEventMgrNotifyListGet](#)

Get a list of Notify objects

[mpiEventMgrNotifyListSet](#)

Create a list of Notify objects

[mpiEventMgrNotifyNext](#)

Get the Notify object just after notify in list

[mpiEventMgrNotifyPrevious](#)

Get the Notify object just before notify in list

[mpiEventMgrNotifyRemove](#)

Remove a Notify object from list

## Data Types

[MPIEventMgrMessage](#)

[MEIEventMgrServiceConfig](#)

# 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 sources for which no prior call to *mpiEventMgrEnable()* has been made are discarded. Events generated by enabled sources are returned in First-In/First-Out (FIFO) order each time a thread calls **mpiEventMgrWait()**.

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 " <i>control</i> " is | Then                                                                                                   |
|--------------------------|--------------------------------------------------------------------------------------------------------|
| MPIHandleVOID            | events will be obtained from all motion controllers on the EventMgr's ( <i>eventMgr</i> ) control list |
| valid                    | <i>control</i> must also be present on the EventMgr's ( <i>eventMgr</i> ) control list                 |

| Return Values |                                                                                                           |
|---------------|-----------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>EventMgrService</i> successfully obtains all pending asynchronous events from the motion controller |

## See Also

[meiEventMgrServiceConfigGet](#) | [meiEventMgrServiceConfigSet](#)

# 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

|                      |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to an EventMgr object              |
| <b>MPIHandleVOID</b> | 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**

if *EventMgrDelete* successfully deletes an EventMgr object and invalidates its handle

## 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

|                     |                                            |
|---------------------|--------------------------------------------|
| <b>MPIMessageOK</b> | if EventMgr is a handle to a valid object. |
|---------------------|--------------------------------------------|

## 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  | if <i>EventMgrConfigGet</i> successfully gets the EventMgr's configuration and writes it into the structure(s) |

## 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**

if *EventMgrConfigSet* successfully sets (writes) the EventMgr's configuration using data from the structure(s)

## 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       |                                                                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrEvent</i> successfully requests that the EventMgr call <code>mpiNotifyEvent(notify, status)</code> for all Notify objects on that Event Manager's list |

## 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  | if <i>EventMgrServiceConfigGet</i> successfully gets the EventMgr's configuration and writes it into the structure(s). |

## 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       |                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrServiceConfigSet</i> successfully sets (writes) the EventMgr's flash configuration using data from the structure(s) |

## 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

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrFlush</i> successfully flushes any pending EventMgr events |
|---------------------|--------------------------------------------------------------------------|

## 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                   |                                                                                                                                                  |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                          | to the <i>index</i> th motion controller (Control) associated with an EventMgr ( <i>eventMgr</i> )                                               |
| MPIHandleVOID                   | if <i>eventMgr</i> is invalid<br>if <i>index</i> is less than 0<br>if <i>index</i> is greater than or equal to <b>mpiEventMgrCount(eventMgr)</b> |
| MPIMessageARG_INVALID           | <i>index</i> is a negative number.                                                                                                               |
| MEIListMessageELEMENT_NOT_FOUND | <i>index</i> is greater than or equal to the number of elements in the list.                                                                     |
| MPIMessageHANDLE_INVALID        | <i>eventMgr</i> is an invalid handle.                                                                                                            |

## 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.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>control</b>  | a handle to a Control object.    |

| Return Values                   |                                                                                                                            |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>             | if <i>EventMgrControlAppend</i> successfully appends control to the list of motion controllers associated with an EventMgr |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i><b>eventMgr</b></i> or <i><b>control</b></i> is an invalid handle.                                               |
| <b>MPIMessageOBJECT_ON_LIST</b> | <i><b>control</b></i> is already on the list.                                                                              |
| <b>MPIMessageNO_MEMORY</b>      | Not enough memory was available.                                                                                           |

## 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 ( <i>eventMgr</i> ) |
| -1            | if <i>eventMgr</i> is an invalid handle                               |
| 0             | if <i>eventMgr</i> 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 conjunction with `mpiEventMgrControlNext()` in order to iterate through the list. `MPIHandleVOID` is returned if the list is empty.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

| Return Values                   |                                                                                          |
|---------------------------------|------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the first motion controller (Control) associated with an EventMgr ( <i>eventMgr</i> ) |
| <b>MPIHandleVOID</b>            | if <i>eventMgr</i> is invalid<br>if <i>eventMgr</i> has no associated motion controllers |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>eventMgr</i> is an invalid handle.                                                 |

## 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.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>control</b>  | a handle to a Control object.    |

| Return Values |                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------|
| <b>index</b>  | of <i>control</i> in the list of motion controllers associated with an EventMgr ( <i>eventMgr</i> ) |
| <b>-1</b>     | if <i>eventMgr</i> is invalid<br>if <i>control</i> 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***.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

### Return Values

|                     |                                                                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrControlInsert</i> successfully inserts the Control object ( <i><b>insert</b></i> ) after another Control object ( <i><b>control</b></i> ) 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 conjunction with `mpiEventMgrControlPrevious()` in order to iterate through the list backwards.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

| Return Values                   |                                                                                          |
|---------------------------------|------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the last motion controller (Control) associated with an EventMgr ( <i>eventMgr</i> )  |
| <b>MPIHandleVOID</b>            | if <i>eventMgr</i> is invalid<br>if <i>eventMgr</i> has no associated motion controllers |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>eventMgr</i> is an invalid handle.                                                 |

## 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*.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

| Return Values       |                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrControlListGet</i> successfully returns the list of Control objects associated with an EventMgr object |

## 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**

if *EventMgrControlListSet* successfully creates a list of Control objects using the Control handles

## 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 conjunction with mpiEventMgrControlFirst() in order to iterate through the list.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>control</b>  | a handle to a Control object.    |

| Return Values                   |                                                                                                                                                   |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the motion controller following <i><b>control</b></i> in the list of motion controllers associated with an EventMgr ( <i><b>eventMgr</b></i> ) |
| <b>MPIHandleVOID</b>            | if <i><b>eventMgr</b></i> is invalid<br>if <i><b>control</b></i> is the last motion controller                                                    |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i><b>eventMgr</b></i> or <i><b>control</b></i> is an invalid handle.                                                                      |

## 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 conjunction with mpiEventMgrControlLast() in order to iterate through the list backwards.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>control</b>  | a handle to a Control object.    |

| Return Values                   |                                                                                                                                     |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the motion controller preceding <b>control</b> in the list of motion controllers associated with an EventMgr ( <b>eventMgr</b> ) |
| <b>MPIHandleVOID</b>            | if <b>eventMgr</b> is invalid<br>if <b>control</b> is the first motion controller                                                   |
| <b>MPIMessageHANDLE_INVALID</b> | Either <b>eventMgr</b> or <b>control</b> is an invalid handle.                                                                      |

## 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***.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>control</b>  | a handle to a Control object.    |

### Return Values

|                     |                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrControlRemove</i> successfully removes a Control object from the list of Control objects associated with an EventMgr ( <i><b>eventMgr</b></i> ) |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 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*.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>index</b>    | a position in the list.          |

| Return Values                          |                                                                                                                                                  |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                          | to the <i>index</i> th Notify object associated with an EventMgr ( <i>eventMgr</i> )                                                             |
| <b>MPIHandleVOID</b>                   | if <i>eventMgr</i> is invalid<br>if <i>index</i> is less than 0<br>if <i>index</i> is greater than or equal to <b>mpiEventMgrCount(eventMgr)</b> |
| <b>MPIMessageARG_INVALID</b>           | <i>index</i> is a negative number.                                                                                                               |
| <b>MEIListMessageELEMENT_NOT_FOUND</b> | <i>index</i> is greater than or equal to the number of elements in the list.                                                                     |
| <b>MPIMessageHANDLE_INVALID</b>        | <i>eventMgr</i> is an invalid handle.                                                                                                            |

## 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***).

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>notify</b>   | a handle to a Notify object.     |

| Return Values                   |                                                                                                                                    |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>             | if <i>EventMgrNotifyAppend</i> successfully appends a Notify object to the list of Notify objects maintained by an EventMgr object |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i><b>eventMgr</b></i> or <i><b>notify</b></i> is an invalid handle.                                                        |
| <b>MPIMessageOBJECT_ON_LIST</b> | if <i><b>notify</b></i> is already on the list.                                                                                    |
| <b>MPIMessageNO_MEMORY</b>      | if not enough memory was available.                                                                                                |

## See Also

# mpiEventMgrNotifyCount

## Declaration

```
long mpiEventMgrNotifyCount(MPIEventMgr eventMgr)
```

Required Header: stdmpi.h

## Description

**mpiEventMgrNotifyCount** returns the number of elements on the list.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

| Return Values |                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------|
| <b>number</b> | of Notify objects in the list (of Notify objects) maintained by an EventMgr ( <i>eventMgr</i> ) |
| <b>-1</b>     | if <i>eventMgr</i> is invalid                                                                   |
| <b>0</b>      | 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 conjunction with `mpiEventMgrNotifyNext()` in order to iterate through the list.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

| Return Values                   |                                                                                                          |
|---------------------------------|----------------------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the first Notify object in the list (of Notify objects) maintained by an EventMgr ( <i>eventMgr</i> ) |
| <b>MPIHandleVOID</b>            | if <i>eventMgr</i> is invalid<br>if the list (of Notify objects) is empty                                |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>eventMgr</i> is an invalid handle.                                                                 |

## 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.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>notify</b>   | a handle to the EventMgr object. |

| Return Values |                                                                                                |
|---------------|------------------------------------------------------------------------------------------------|
| <b>index</b>  | of <i>notify</i> in the list (of Notify objects) maintained by an EventMgr ( <i>eventMgr</i> ) |
| <b>-1</b>     | if eventMgr is invalid<br>if <i>notify</i> 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       |                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrNotifyInsert</i> successfully places a Notify object after another Notify object in the list of Notify objects |

## 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 conjunction with `mpiEventMgrNotifyPrevious()` in order to iterate through the list backwards.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
|-----------------|----------------------------------|

| Return Values                   |                                                                                     |
|---------------------------------|-------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the last Notify object in the list maintained by an EventMgr ( <i>eventMgr</i> ) |
| <b>MPIHandleVOID</b>            | if <i>eventMgr</i> is invalid<br>if the list (of Notify objects) is empty           |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>eventMgr</i> is an invalid handle.                                            |

## 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**

if *EventMgrNotifyListGet* successfully returns the list of Notify objects for an EventMgr object

## 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  | if <i>EventMgrNotifyListSet</i> successfully creates a list of Notify objects using the Notify handles specified by <i><b>notifyList</b></i> |

## 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 conjunction with mpiEventMgrNotifyFirst() in order to iterate through the list.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>notify</b>   | a handle to a Notify object.     |

| Return Values                   |                                                                                                                                                                |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the Notify object after another Notify object ( <i><b>notify</b></i> ) in the list (of Notify objects) maintained by an EventMgr ( <i><b>eventMgr</b></i> ) |
| <b>MPIHandleVOID</b>            | if <i><b>eventMgr</b></i> is invalid<br>if <i><b>notify</b></i> is the last notify object in the list                                                          |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i><b>eventMgr</b></i> or <i><b>notify</b></i> is an invalid handle.                                                                                    |

## 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 conjunction with `mpiEventMgrNotifyLast()` in order to iterate through the list backwards.

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>notify</b>   | a handle to a Notify object.     |

| Return Values                   |                                                                                                                                                                      |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the Notify object just before another Notify object ( <i><b>notify</b></i> ) in the list (of Notify objects) maintained by an EventMgr ( <i><b>eventMgr</b></i> ) |
| <b>MPIHandleVOID</b>            | if <i><b>eventMgr</b></i> is invalid<br>if <i><b>notify</b></i> is the first Notify object in the list                                                               |
| <b>MPIMessageHANDLE_INVALID</b> | either <i><b>eventMgr</b></i> or <i><b>notify</b></i> 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***).

|                 |                                  |
|-----------------|----------------------------------|
| <b>eventMgr</b> | a handle to the EventMgr object. |
| <b>notify</b>   | a handle to a Notify object.     |

| Return Values       |                                                                                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>EventMgrNotifyRemove</i> successfully removes a Notify object from the list of Notify objects maintained by an EventMgr object |

## 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
{ long allProcesses; /* TRUE => collect
                      events from all processes,
                      else EventMgr process only */
} MEIEventMgrServiceConfig;
```

## Description

| <b>allProcesses</b> |                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                     | 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.

## Methods

### Create, Delete, Validate Methods

|                                          |                        |
|------------------------------------------|------------------------|
| <a href="#"><u>mpiFilterCreate</u></a>   | Create Filter object   |
| <a href="#"><u>mpiFilterDelete</u></a>   | Delete Filter object   |
| <a href="#"><u>mpiFilterValidate</u></a> | Validate Filter object |

### Configuration and Information Methods

|                                                |                                    |
|------------------------------------------------|------------------------------------|
| <a href="#"><u>mpiFilterConfigGet</u></a>      | Get Filter configuration           |
| <a href="#"><u>mpiFilterConfigSet</u></a>      | Set Filter configuration           |
| <a href="#"><u>mpiFilterFlashConfigGet</u></a> | Get flash configuration for Filter |
| <a href="#"><u>mpiFilterFlashConfigSet</u></a> | Set flash configuration for Filter |
| <a href="#"><u>mpiFilterGainGet</u></a>        | Get gain coefficients              |
| <a href="#"><u>mpiFilterGainSet</u></a>        | Set current gain index             |
| <a href="#"><u>mpiFilterGainIndexGet</u></a>   | Get current gain index             |
| <a href="#"><u>mpiFilterGainIndexSet</u></a>   | Set current gain index             |

### Memory Methods

|                                           |                                                    |
|-------------------------------------------|----------------------------------------------------|
| <a href="#"><u>mpiFilterMemory</u></a>    | Get address to Filter memory                       |
| <a href="#"><u>mpiFilterMemoryGet</u></a> | Copy data from Filter memory to application memory |
| <a href="#"><u>mpiFilterMemorySet</u></a> | Copy data from application memory to Filter memory |

### Relational Methods

|                                            |                                                         |
|--------------------------------------------|---------------------------------------------------------|
| <a href="#"><u>mpiFilterAxisMapGet</u></a> | Get object map of axes associated with Filter           |
| <a href="#"><u>mpiFilterAxisMapSet</u></a> | Set axes associated with Filter                         |
| <a href="#"><u>mpiFilterControl</u></a>    | Return handle of Control that is associated with Filter |

|                                                    |                                                 |
|----------------------------------------------------|-------------------------------------------------|
| <a href="#"><u>mpiFilter<b>MotorMapGet</b></u></a> | Get object map of Motors associated with Filter |
| <a href="#"><u>mpiFilter<b>MotorMapSet</b></u></a> | Set Motors to be associated with Filter         |
| <a href="#"><u>mpiFilter<b>Number</b></u></a>      | Get index of Filter (for Control list)          |

## Action Methods

|                                                         |                                  |
|---------------------------------------------------------|----------------------------------|
| <a href="#"><u>mpiFilter<b>IntergratorReset</b></u></a> | Reset the integrators of filter. |
|---------------------------------------------------------|----------------------------------|

## Postfilter Methods

|                                                             |                                        |
|-------------------------------------------------------------|----------------------------------------|
| <a href="#"><u>meiFilter<b>PostfilterGet</b></u></a>        | Reads postfilter information.          |
| <a href="#"><u>meiFilter<b>PostfilterSet</b></u></a>        | Writes postfilter information.         |
| <a href="#"><u>meiFilter<b>PostfilterSectionGet</b></u></a> | Reads postfilter section information.  |
| <a href="#"><u>meiFilter<b>PostfilterSectionSet</b></u></a> | 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        |                                           |
|----------------------|-------------------------------------------|
| <b>handle</b>        | to an Filter object                       |
| <b>MPIHandleVOID</b> | 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

|                     |                                                                                        |
|---------------------|----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterDelete</i> successfully deletes a Filter object and invalidates its handle |
|---------------------|----------------------------------------------------------------------------------------|

## 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**

if Filter is a handle to a valid object.

## 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**

if *FilterConfigGet* successfully writes the Filter's configuration to the structure(s)

## See Also

[mpiFilterConfigSet](#) | [MEIFilterConfig](#)



# mpiFilterConfigSet

## Declaration

```
long mpiFilterConfigSet(MPIFilter filter,
                       MPIFilterConfig *config,
                       void *external)
```

Required Header: stdmpi.h

## Description

**mpiFilterConfigSet** 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**

if *FilterConfigSet* successfully sets the Filter's configuration using data from the structure(s)

## 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.

| Return Values       |                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterFlashConfigGet</i> successfully writes the Filter's flash configuration to the structure(s)<br><i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |

## 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.

| Return Values       |                                                                                                                                                                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterFlashConfigSet</i> successfully sets the Filter's flash configuration using data from the structure(s)<br><i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |

## 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***.

### Return Values

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterGainGet</i> successfully writes the gain coefficients to the structure |
|---------------------|------------------------------------------------------------------------------------|

## 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

[mpiFilterGainSet](#)

# 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*.

### Return Values

**MPIMessageOK**

if *FilterGainSet* successfully sets the gain coefficients of a Filter using data from the structure

## 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

[mpiFilterGainGet](#)

# 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

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterGainIndexGet</i> successfully writes the gain index to the location |
|---------------------|---------------------------------------------------------------------------------|

## 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       |                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterGainIndexSet</i> successfully sets the current gain index to <i>gainIndex</i> |

## 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**

if *FilterMemory* successfully writes the Filter's memory address to the contents of *memory*

## See Also

[mpiFilterMemoryGet](#) | [mpiFilterMemorySet](#)

# mpiFilterMemoryGet

## Declaration

```
long mpiFilterMemoryGet(MPIFilter filter,  
                        void      *dst,  
                        void      *src,  
                        long      count)
```

**Required Header:** stdmpi.h

## 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**

if *FilterMemoryGet* successfully copies data from Filter memory to application memory

## See Also

[mpiFilterMemorySet](#) | [mpiFilterMemory](#)

# mpiFilterMemorySet

## Declaration

```
long mpiFilterMemorySet(MPIFilter filter,  
                        void      *dst,  
                        void      *src,  
                        long      count)
```

**Required Header:** stdmpi.h

## 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**

if *FilterMemorySet* successfully copies data from application memory to Filter memory

## 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**

if *FilterAxisMapGet* successfully writes the object map of Axes to the structure

## 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

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FilterAxisMapSet</i> successfully sets the Axes using the object map |
|---------------------|----------------------------------------------------------------------------|

## 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        |                                                             |
|----------------------|-------------------------------------------------------------|
| <b>handle</b>        | to a Control object that a Filter object is associated with |
| <b>MPIHandleVOID</b> | 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**

if *FilterMotorMapGet* successfully writes the object map of the Motors to the structure

## 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**

if *FilterMotorMapGet* successfully sets the Motors using data from the object map

## 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  | if <i>FilterNumber</i> successfully writes the index of a Filter to the contents of <b>number</b> |

## See Also

# mpiFilterIntergratorReset

## Declaration

```
long mpiFilterIntegratorReset(MPIFilter filter)
```

Required Header: stdmpi.h

## Description

**mpiFilterIntergratorReset** resets the integrators of filter.

| Return Values                            |                                                                                           |
|------------------------------------------|-------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                      | if <i>mpiFilterIntegratorReset</i> successfully clears the integrators of <i>filter</i> . |
| <b>MPIFilterMessageINVALID_ALGORITHM</b> | if the <i>filter</i> 's current algorithm does not use integrators.                       |

## 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](#)

# mpiFilterPosfilterGet

## 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.

|                      |                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------|
| <b>filter</b>        | the handle of the MPIFilter object whose postfilter configuration is to be read.             |
| <b>*sectionCount</b> | the data location where the postfilter's current section count will be written.              |
| <b>*sections</b>     | the data location where the postfilter's current section configuration data will be written. |

| Return Values                              |                                                                                                                                                                                                                                           |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                        | if <i>PostfilterGet</i> successfully retrieves postfilter information.                                                                                                                                                                    |
| <b>MPIFilterMessageCONVERSION_DIV_BY_0</b> | Returned when meiFilterPostfilterGet(...) 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. |
| <b>MPIFilterMessageINVALID_FILTER_FORM</b> | Returned when meiFilterPostfilterGet(...) cannot interpret the current postfilter's form (when the form is something other than NONE, IIR, or BIQUAD).                                                                                    |

## 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 (sections[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 <i>filter</i> object.                      |
| *sections      | a pointer to an array of MEIPostfilterSection data structures to be set in <i>filter</i> . |

| Return Values |                                                                     |
|---------------|---------------------------------------------------------------------|
| MPIMessageOK  | if <i>PostfilterSet</i> successfully writes postfilter information. |

## 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**<sup>th</sup> 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.

|                      |                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------|
| <b>filter</b>        | the handle of the MPIFilter object whose postfilter section configuration is to be read.     |
| <b>sectionNumber</b> | the index of the postfilter section whose configuration is to be read.                       |
| <b>section</b>       | the data location where the postfilter's current section configuration data will be written. |

| Return Values                              |                                                                                                                                                                                                                                     |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                        | if <i>PostfilterSectionGet</i> successfully reads postfilter section information.                                                                                                                                                   |
| <b>MPIFilterMessageCONVERSION_DIV_BY_0</b> | Returned when meiFilterPostfilterSectionGet(...) cannot convert digital coefficients to analog coefficients. When this error occurs, the section will report its type as MEIFilterTypeUNKNOWN and will not contain any analog data. |
| <b>MPIFilterMessageSECTION_NOT_ENABLED</b> | Returned when no postfilter sections are enabled.                                                                                                                                                                                   |
| <b>MPIFilterMessageINVALID_FILTER_FORM</b> | Returned when meiFilterPostfilterSectionGet(...) cannot interpret the current postfilter's form (when the form is something other than NONE, IIR, or BIQUAD).                                                                       |

## 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, &section);
        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*<sup>th</sup> 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.

|                      |                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------|
| <b>filter</b>        | the handle of the MPIFilter object whose postfilter section configuration is to be set. |
| <b>sectionNumber</b> | the index of the postfilter section whose configuration is to be set.                   |
| <b>*section</b>      | the data location of the section configuration to copy to the controller.               |

| Return Values       |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>PostfilterSectionSet</i> successfully writes postfilter section information. |

## 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, sion);

    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.

|          |                   |
|----------|-------------------|
| <b>f</b> | float coefficient |
| <b>l</b> | 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

|                  |                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------|
| <b>gainIndex</b> | Gain table index. Gain tables number 0 to MPIFilterGainCOUNT_MAX -1 (MPIFilterGainCOUNT_MAX = 5). |
| <b>gain</b>      | See <a href="#">MPIObjectMap</a>                                                                  |
| <b>axisMap</b>   | See <a href="#">MPIObjectMap</a>                                                                  |
| <b>motorMap</b>  | See <a href="#">MPIObjectMap</a>                                                                  |

## Definition: MEIFilterConfig

```
typedef struct MEIFilterConfig {
    MEIXmpAlgorithm      Algorithm;

    MEIXmpAxisInput      Axis[MEIXmpFilterAxisInputs];

    long                  *VelPositionPtr;

    MEIXmpSwitchType      GainSwitchType;
    float                  GainDelay;
    long                  GainWindow;
    MEIXmpSwitchType      PPISwitchType;
    MEIXmpPPIMode          PPIMode;
    float                  PPIDelay;
    long                  PPIWindow;
    MEIXmpIntResetConfig   ResetIntegratorConfig;
    float                  ResetIntegratorDelay;

    MEIXmpFilterForm       PostFilterForm;
    MEIXmpPostFilter       PostFilter;
} MEIFilterConfig;
```

## 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.

|                                            |                                                                                                                                                                                                                                                                                                                                  |                    |               |                    |               |                     |                      |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|---------------|--------------------|---------------|---------------------|----------------------|
| <b>Algorithm</b>                           | <p>This value defines the algorithm that the filter is executing every servo cycle. The most common values are:</p> <table> <tr> <td>MEIXmpAlgorithmPID</td><td>PID algorithm</td></tr> <tr> <td>MEIXmpAlgorithmPIV</td><td>PIV algorithm</td></tr> <tr> <td>MEIXmpAlgorithmNONE</td><td>No control algorithm</td></tr> </table> | MEIXmpAlgorithmPID | PID algorithm | MEIXmpAlgorithmPIV | PIV algorithm | MEIXmpAlgorithmNONE | No control algorithm |
| MEIXmpAlgorithmPID                         | PID algorithm                                                                                                                                                                                                                                                                                                                    |                    |               |                    |               |                     |                      |
| MEIXmpAlgorithmPIV                         | PIV algorithm                                                                                                                                                                                                                                                                                                                    |                    |               |                    |               |                     |                      |
| MEIXmpAlgorithmNONE                        | No control algorithm                                                                                                                                                                                                                                                                                                             |                    |               |                    |               |                     |                      |
| <b>Axis</b><br>[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.                                                                                           |                    |               |                    |               |                     |                      |
| <b>VelPositionPtr</b>                      | Velocity position pointer to an encoder input for algorithms that require a velocity encoder position input (such as the PIV algorithm).                                                                                                                                                                                         |                    |               |                    |               |                     |                      |
| <b>AuxInput</b><br>[MEIXmpFilterAuxInputs] | <p>This array is a place holder for additional filter inputs from analog sources.</p> <p><b>This is currently not supported and is reserved for future use.</b></p>                                                                                                                                                              |                    |               |                    |               |                     |                      |
| <b>GainSwitchType</b>                      | <p>Value to define the gain table switch type.</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                           |                    |               |                    |               |                     |                      |
| <b>GainDelay</b>                           | <p>Custom Delay</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                                                          |                    |               |                    |               |                     |                      |
| <b>GainWindow</b>                          | <p>Custom Delay</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                                                          |                    |               |                    |               |                     |                      |
| <b>PPISwitchType</b>                       | <p>Value to define the gain switch type for PPI mode.</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                    |                    |               |                    |               |                     |                      |
| <b>PPIMode</b>                             | <p>Value to define the PPI switch mode.</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                                  |                    |               |                    |               |                     |                      |
| <b>PPIDelay</b>                            | <p>Custom Delay</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                                                          |                    |               |                    |               |                     |                      |
| <b>PPIWindow</b>                           | <p>Custom Window</p> <p><b>Not implemented in standard firmware.</b></p>                                                                                                                                                                                                                                                         |                    |               |                    |               |                     |                      |
| <b>ResetIntegratorConfig</b>               | <p>Value to define the integrator's reset configuration.</p> <p><b>Not supported in standard firmware.</b></p>                                                                                                                                                                                                                   |                    |               |                    |               |                     |                      |
| <b>ResetIntegratorDelay</b>                | <p>Value to define the integrator's reset delay.</p> <p><b>Not supported in standard firmware.</b></p>                                                                                                                                                                                                                           |                    |               |                    |               |                     |                      |



|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>PostFilterForm</b> | <p>This value defines the form for postfilters when they are configured using <code>mpiFilterConfigGet/Set()</code>.</p> <p>Supported values are:</p> <ul style="list-style-type: none"> <li>• <b>MEIXmpFilterFormIIR</b>,<br/>IIR Filter</li> <li>• <b>MEIXmpFilterFormBIQ</b>,<br/>Bi-Quad Filter</li> <li>• <b>MEIXmpFilterFormSS_BIQ</b>,<br/>State Space form of Bi-Quad Filter</li> <li>• <b>MEIXmpFilterFormINT_BIQ</b>,<br/>Integer (64-bit) Bi-Quad Filter</li> <li>• <b>MEIXmpFilterFormINT_SS_BIQ</b>,<br/>Integer State Space form of Bi-Quad Filter</li> </ul> <p>Though the postfilter may be configured through this parameter, it is strongly recommended that users use the <code>meiFilterPostfilter()</code> methods instead for a more intuitive and user-friendly interface.</p> |
| <b>PostFilter</b>     | <p>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 through this parameter, it is strongly recommended that users use the <code>meiFilterPostfilter()</code> methods instead for a more intuitive interface.</p> <p>Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.</p>                                                                                                                                                                                                                                                                                                   |

## 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.

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIFilterFormIIR</b>        | <b><i>Deprecated.</i></b> Cascaded biquad sections offer better precision and better calculation performance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>MEIFilterFormBIQUAD</b>     | <p>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:</p> <p>The XMP's representation of this filter is:</p> <p>w0: Intermediate result<br/> u(k): filter input<br/> a1, a2, b0, b1, and b2: discrete biquad coefficients<br/> y(k):filter output<br/> x1k and x2k: filter states</p>                                                                                                                                                                                                                                                                                                            |
| <b>MEIFilterFormSS_BIQUAD</b>  | <p>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:</p> <p>u(k): filter input<br/> d1, c1, c2, a2, a1,b1: discrete biquad coefficients<br/> y(k):filter output<br/> p1k and p2k: filter states</p>                                                                                                                                                                                                                                                                                                                              |
| <b>MEIFilterFormINT_BIQUAD</b> | <p>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.</p> <p>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 <math>1=2^{30}</math>. 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.</p> |

**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

|              |                                    |
|--------------|------------------------------------|
| <b>coeff</b> | see <a href="#">MPIFilterCoeff</a> |
|--------------|------------------------------------|

## 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.

|                                    |                                                                                                                                                                                                                                                                      |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIFilterGainIndexNO_MOTION</b> | No commanded motion. Trajectory parameters Velocity, Acceleration, and Jerk equal zero.                                                                                                                                                                              |
| <b>MEIFilterGainIndexACCEL</b>     | Acceleration portion of the commanded move.                                                                                                                                                                                                                          |
| <b>MEIFilterGainIndexDECEL</b>     | Deceleration portion of the commanded move.                                                                                                                                                                                                                          |
| <b>MEIFilterGainIndexVELOCITY</b>  | Constant velocity portion of the commanded move. Gain switching is configured by setting the GainSwitichType, 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 {
    MEIFilterGainPIDCoeffINVALID = -1,

    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 {
    MEIFilterGainPIVCoeffINVALID = -1,

    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](#)



# MEIFilterGainTypeID

## Definition

```
static MEIDataType MEIFilterGainTypeID[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

**MEIFilterGainTypeID** is a static array that describes the data type of the coefficients for the PID algorithm. Specifically, an element of MEIFilterGainTypeID describes which member of the union MPIFilterCoeff to access when using the data structure MPIFilterCoeff.

MEIFilterGainTypeID 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 MEIFilterGainTypeID.

## 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.

|                                         |                                                                                                                                                                                       |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIFilterTypeUNITY_GAIN</b>          | A unity gain filter. This effectively performs no filtering.                                                                                                                          |
| <b>MEIFilterTypeSINGLE_ORDER</b>        | A single order filter                                                                                                                                                                 |
| <b>MEIFilterTypeLOW_PASS</b>            | A low pass filter                                                                                                                                                                     |
| <b>MEIFilterType_HIGH_PASS</b>          | A high pass filter.                                                                                                                                                                   |
| <b>MEIFilterTypeNOTCH</b>               | A notch filter                                                                                                                                                                        |
| <b>MEIFilterTypeRESONATOR</b>           | A resonator filter.                                                                                                                                                                   |
| <b>MEIFilterTypeLEAD_LAG</b>            | A lead or lag filter.                                                                                                                                                                 |
| <b>MEIFilterTypeZERO_GAIN</b>           | Zeros the output of a filter.                                                                                                                                                         |
| <b>MEIFilterTypeBIQUAD</b>              | An analog biquad filter. When reading postfilter data, this type means that the postfilter section could not be identified as a standard filter type.                                 |
| <b>MEIFilterTypeDIGITAL_BIQUAD</b>      | A digital biquad filter. This is only used for setting postfilter sections.                                                                                                           |
| <b>MEIFilterTypePOLES_ZERO</b>          | Analog poles and zeros filter (maximum of two poles and zeros) with unity zero-frequency amplitude. This is only used for setting postfilter sections.                                |
| <b>MEIFilterTypeDIGITAL_POLES_ZEROS</b> | Digital poles and zeros filter (maximum of two poles and zeros) with unity zero-frequency amplitude. This is only used for setting postfilter sections.                               |
| <b>MEIFilterTypeUNKNOWN</b>             | Returned by <code>meiFilterPostfilterGet(...)</code> and <code>meiFilterPostfilterSectionGet(...)</code> 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;

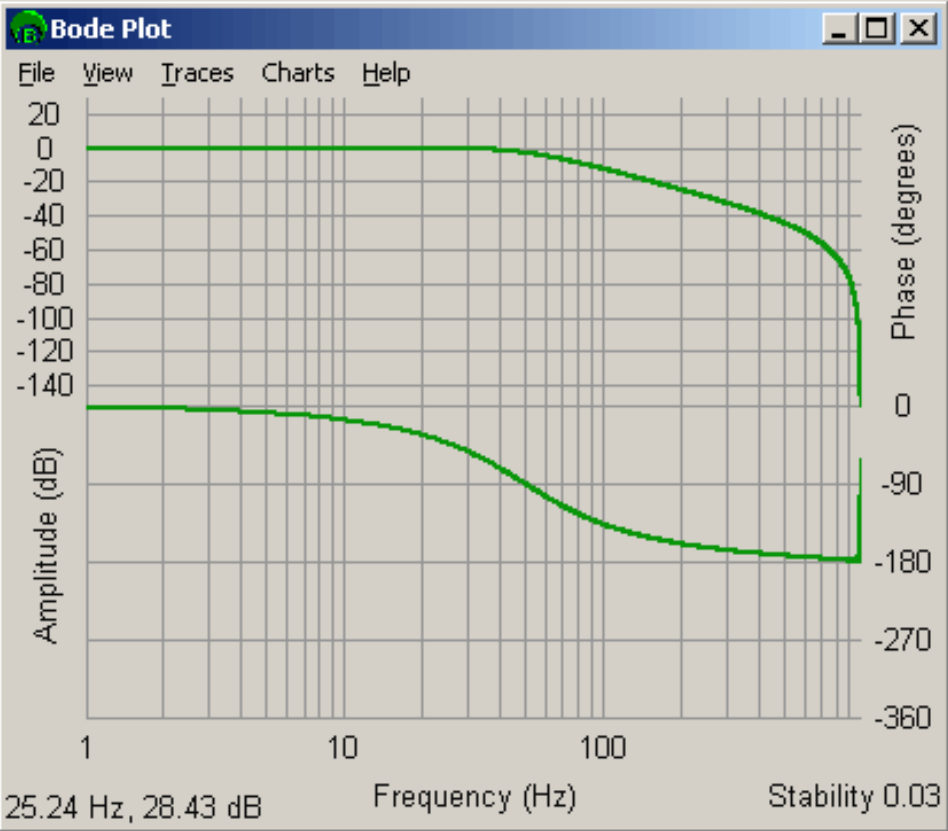
    struct {
        double d1;
        double c1;
        double c2;
        double a2;
        double a1;
        double b1;
    } stateSpaceBiquad;
} data;
} MEIPostfilterSection;

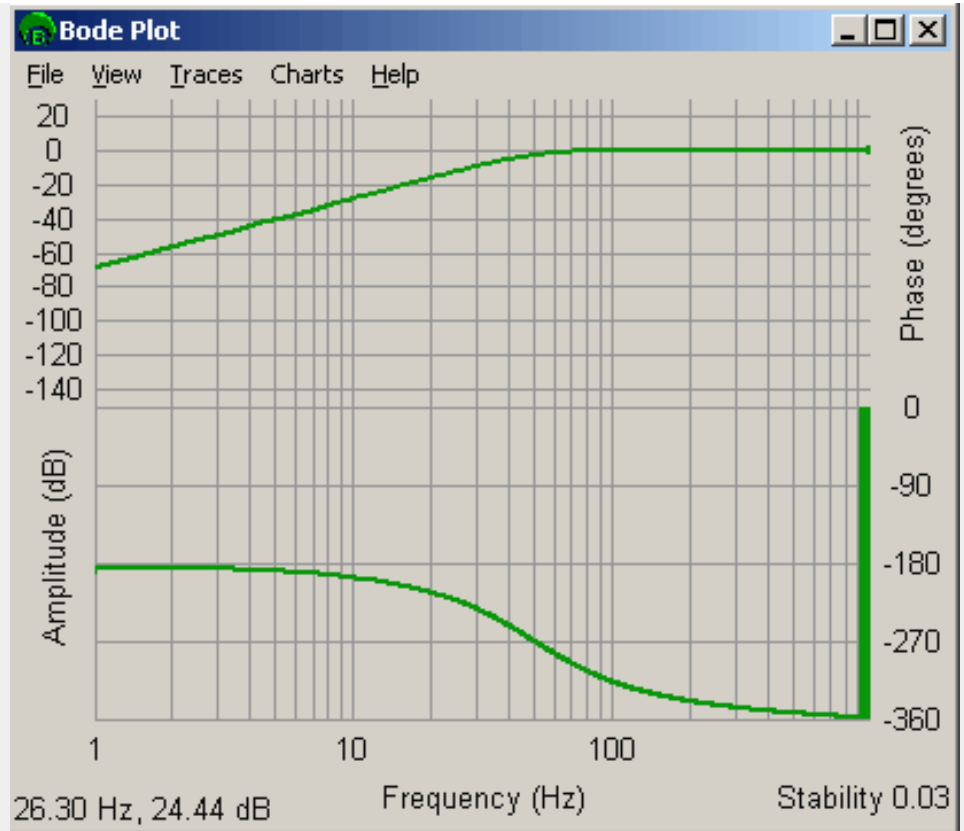
```

## 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.

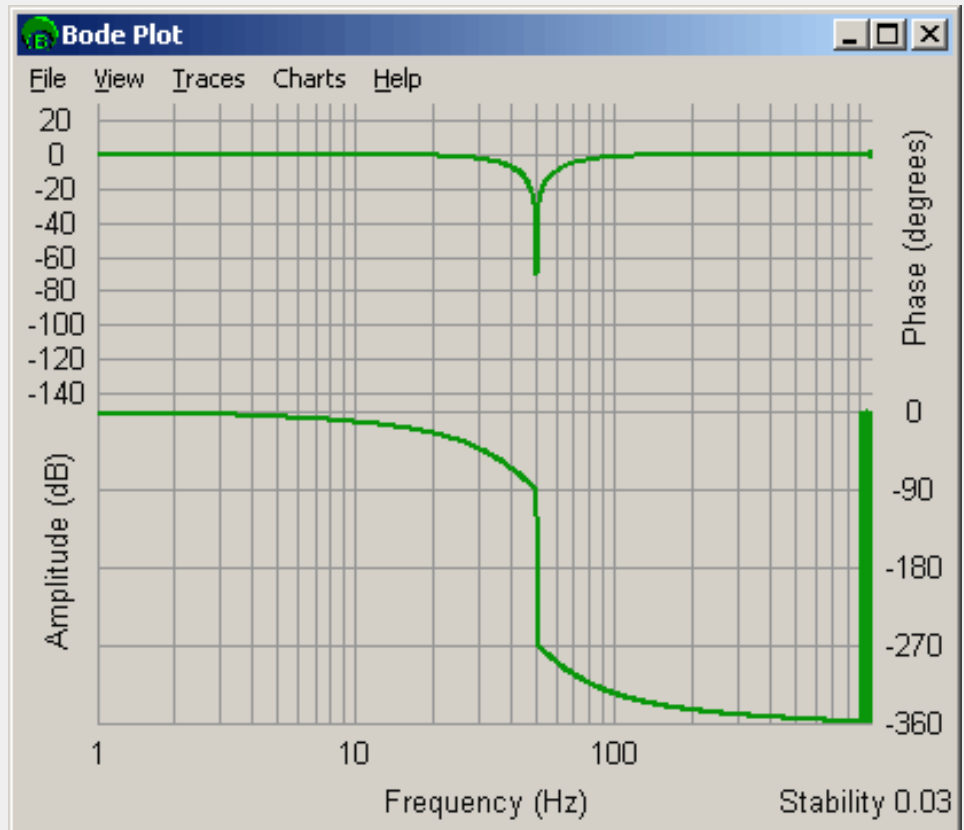
|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>                | <p>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 <a href="#">MEIFilterType</a> documentation.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>form</b>                | <p>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 <a href="#">MEIFilterForm</a> for more information.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>lowPass.breakpoint</b>  | <p>The break point (measured in Hertz) of a low pass postfilter section.</p> <div><p>The image shows a Bode Plot window titled 'Bode Plot'. It contains two subplots sharing a common x-axis representing Frequency in Hertz (Hz) on a logarithmic scale from 1 to 100. The top subplot shows the Phase in degrees, with a green curve starting at 0 degrees and decreasing to -180 degrees at approximately 50 Hz. The bottom subplot shows the Amplitude in dB, with a green curve starting at 28.43 dB and decreasing to -140 dB at approximately 50 Hz. The status bar at the bottom indicates '25.24 Hz, 28.43 dB' and 'Stability 0.03'.</p></div> <p>Example of a 50 Hz low pass filter.</p> |
| <b>highPass.breakpoint</b> | <p>The break point (measured in Hertz) of a high pass postfilter section.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |



Example of a 50 Hz High pass filter

**notch.centerFrequency**

The bandwidth (measured in Hertz) of a notch postfilter section.

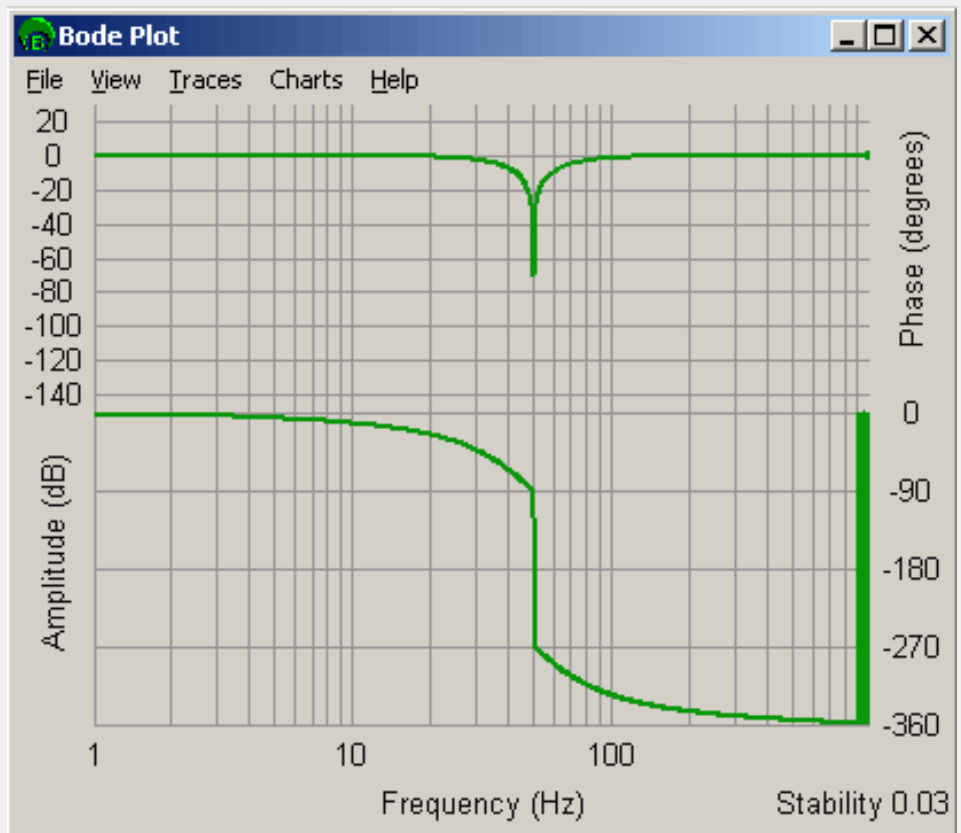


Example of a 50 Hz Center / 50 Hz Bandwidth Notch filter. Note that phase wrapping gives the illusion that phase drops 180 degrees after the

center frequency. The phase raises by 180 degrees.

**notch.bandwidth**

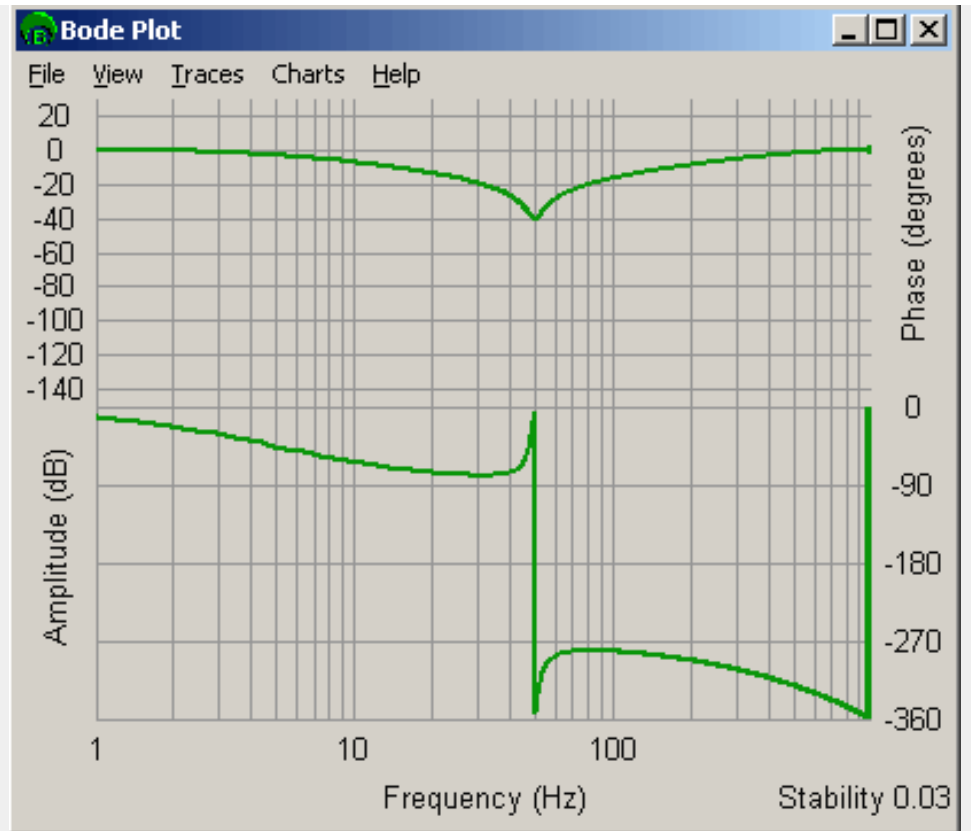
The center frequency (measured in Hertz) of a notch postfilter section.



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.

**resonator.centerFrequency**

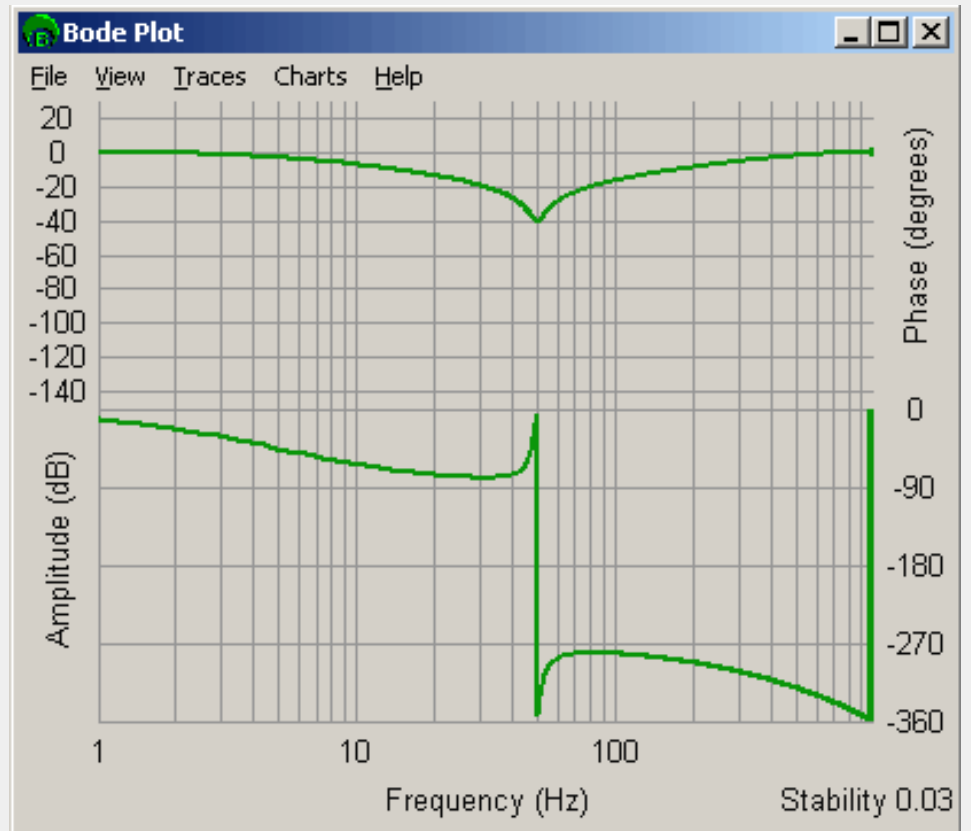
The center frequency (measured in Hertz) of a resonator postfilter section.



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

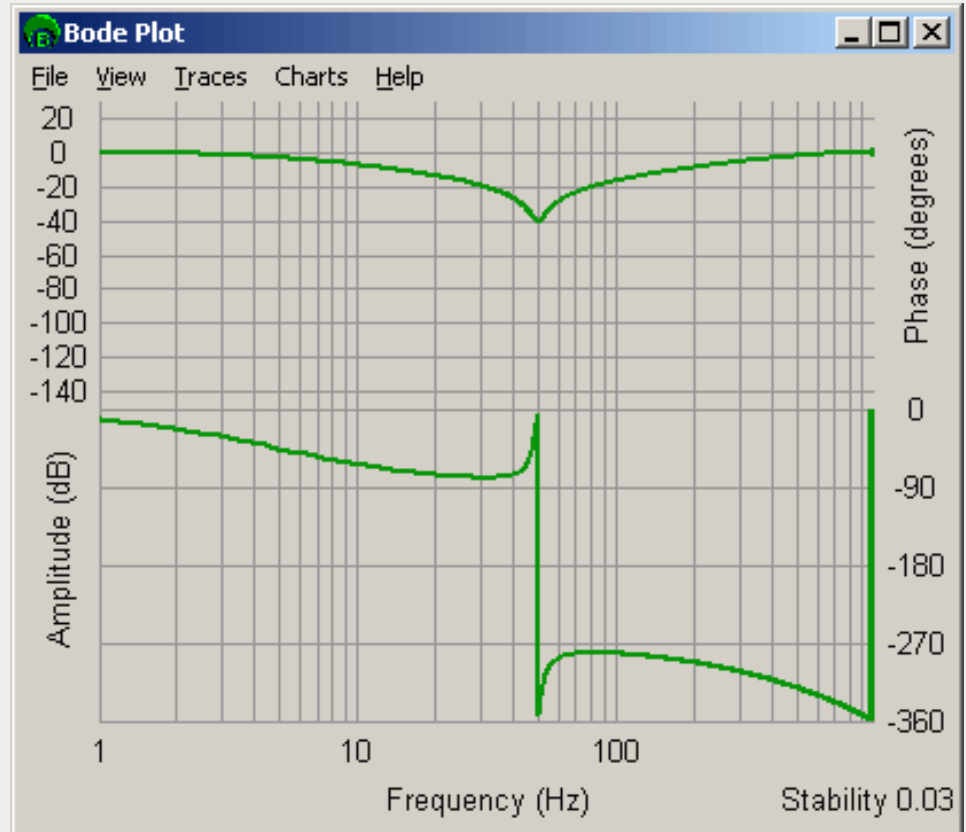
The bandwidth (measured in Hertz) of a resonator postfilter section.



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.

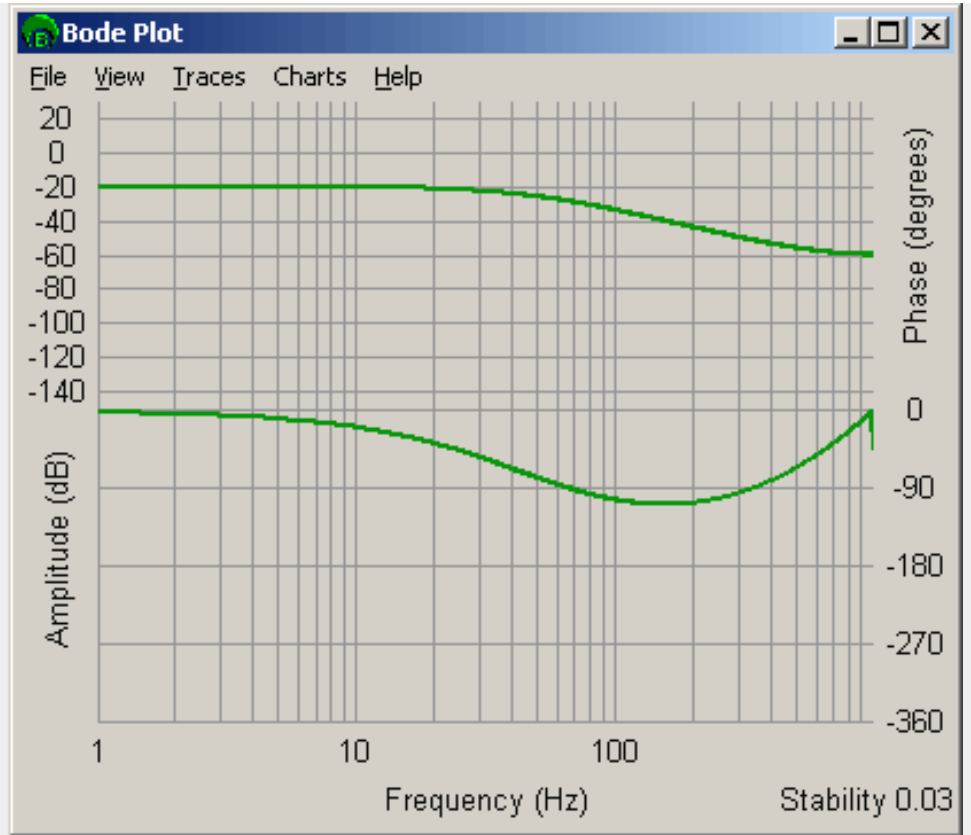


Example of a 50 Hz center / 50 Hz Bandwidth / -40 dB Gain Resonator filter. Note that phase wrapping gives the illusion that the phase drops 360 degrees after the center frequency.

### **leadLad.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:

$$centerFrequencyGain = 20 \cdot \log_{10} \left( \frac{10^{\frac{lowFrequencyGain}{20}} + 10^{\frac{highFrequencyGain}{20}}}{2} \right)$$

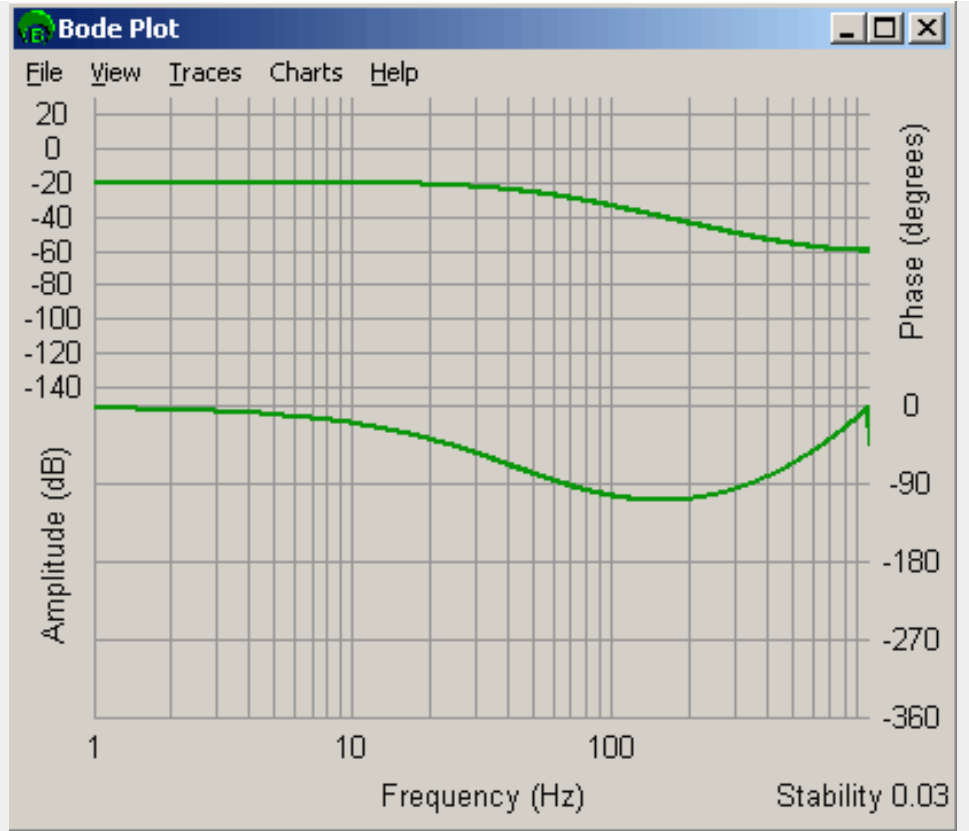


Example of a -20 dB low frequency gain / -60 dB high frequency gain / 50 Hz center lead lag filter.

### **leadLag.lowFrequencyGain**

The low frequency gain (measured in dB) of a lead or lag postfilter section.

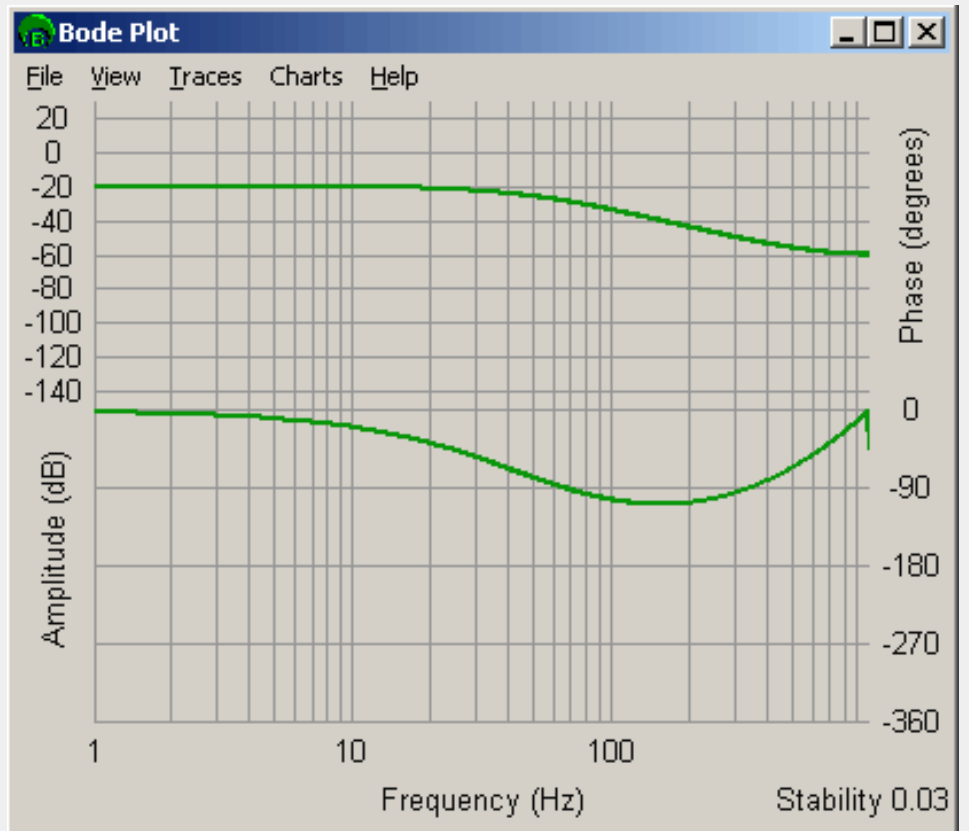




Example of a -20 dB low frequency gain / -60 dB high frequency gain / 50 Hz center lead lag filter.

### 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.                                                                                                                                                                                                                                                                                                                     |
| <b>biquad.a1</b>                   | <p>The analog coefficients of a single order or bi-quad postfilter section.</p> <p>Analog values of the postfilter coefficients are produced as parts of a Laplace Transform:</p> $H(s) = \frac{b_0 + b_1 \cdot s + b_2 \cdot s^2}{1 + a_1 \cdot s + a_2 \cdot s^2} \text{ where } s = j \cdot \omega_{warped}$ <p>and</p> $\omega_{warped} = 2 \cdot sampleRate \cdot \tan\left(\frac{\pi \cdot f}{sampleRate}\right)$ |
| <b>biquad.a2</b>                   |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>biquad.b0</b>                   |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>biquad.b1</b>                   |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>biquad.b2</b>                   |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.a1</b>            | The digital coefficients of a single order or bi-quad postfilter section.                                                                                                                                                                                                                                                                                                                                               |
| <b>digitalBiquad.a2</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.b0</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.b1</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.b2</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.d1</b>            | The digital coefficients of a state-space bi-quad postfilter section.                                                                                                                                                                                                                                                                                                                                                   |
| <b>digitalBiquad.c1</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.c2</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.a2</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.a1</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalBiquad.b1</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>polesZeros.poleCount</b>        | Analog poles and zeros.                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>polesZeros.zeroCount</b>        |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>polesZeros.pole[].real</b>      |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>polesZeros.pole[].imag</b>      |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>digitalPolesZeros.poleCount</b> | Digital poles and zeros.                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>digitalPolesZeros.zeroCount</b> |                                                                                                                                                                                                                                                                                                                                                                                                                         |

|                                      |                           |
|--------------------------------------|---------------------------|
| <b>digitalPolesZeros.pole[].real</b> |                           |
| <b>digitalpolesZeros.pole[].imag</b> |                           |
| <b>stateSpaceBiquad.d1</b>           | State space coefficients. |
| <b>stateSpaceBiquad.c1</b>           |                           |
| <b>stateSpaceBiquad.c2</b>           |                           |
| <b>stateSpaceBiquad.a2</b>           |                           |
| <b>stateSpaceBiquad.a1</b>           |                           |
| <b>stateSpaceBiquad.b1</b>           |                           |

## 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](#) Motion Engineering. 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

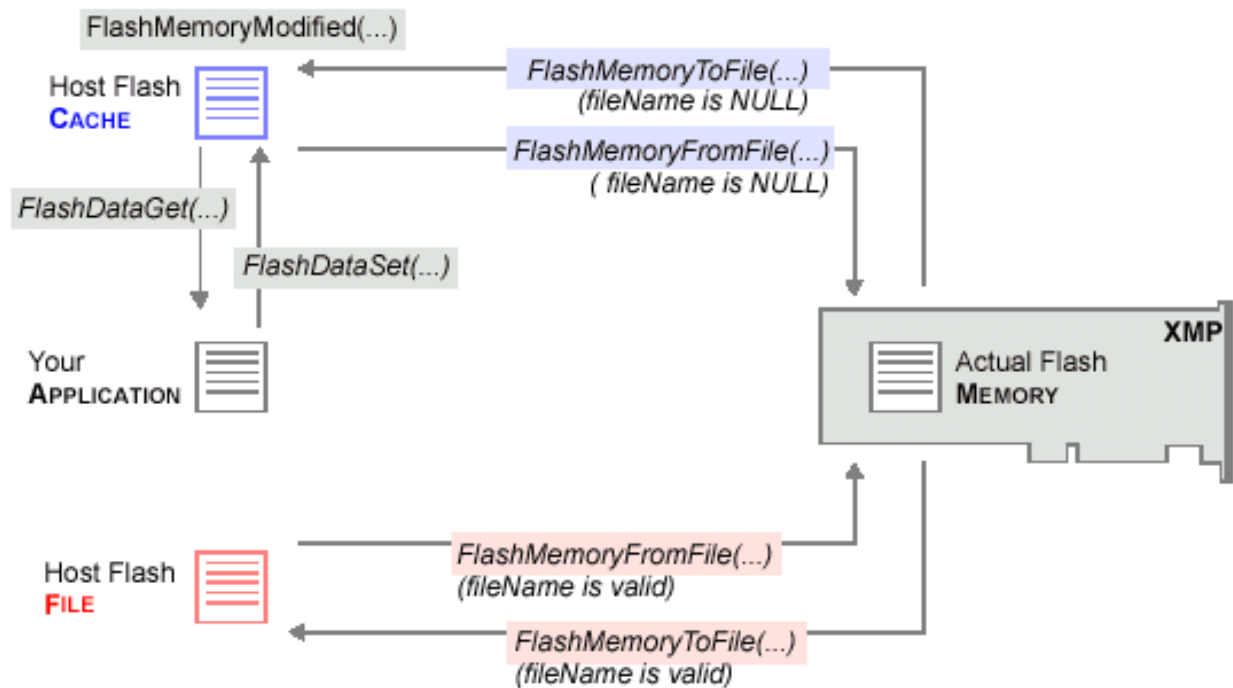
## Introduction

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 `meiFlashMemoryToFile(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 `meiFlashMemory/FromFile(...)` 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.



## Methods

### Create, Delete, Validate Methods

|                                  |                       |
|----------------------------------|-----------------------|
| <a href="#">meiFlashCreate</a>   | Create Flash object   |
| <a href="#">meiFlashDelete</a>   | Delete Flash object   |
| <a href="#">meiFlashValidate</a> | Validate Flash object |

### Configuration and Information Methods

|                                   |                                                                           |
|-----------------------------------|---------------------------------------------------------------------------|
| <a href="#">meiFlashConfigGet</a> | Copy flash config from cache to application memory                        |
| <a href="#">meiFlashConfigSet</a> | Copy flash config from application memory to cache                        |
| <a href="#">meiFlashDataGet</a>   | Get count bytes of flash data memory and write them in application memory |
| <a href="#">meiFlashDataSet</a>   | Set count bytes of flash data memory using application memory             |

### Memory Methods

|                                             |                                                                        |
|---------------------------------------------|------------------------------------------------------------------------|
| <a href="#">meiFlashMemoryFromFileImage</a> | Write actual flash memory using the binary image contained in filename |
| <a href="#">meiFlashMemoryFromFileType</a>  | Copy count bytes of flash memory to application memory                 |
| <a href="#">meiFlashMemoryGet</a>           | Determine if flash cache has been modified                             |
| <a href="#">meiFlashMemoryModified</a>      | Copy count bytes of application memory to flash memory                 |
| <a href="#">meiFlashMemorySet</a>           | Write actual flash memory to cache or to file                          |
| <a href="#">meiFlashMemoryFromFile</a>      | Save actual flash memory to cache or to file                           |
| <a href="#">meiFlashMemoryToFile</a>        |                                                                        |
| <a href="#">meiFlashMemoryVerify</a>        |                                                                        |

### Relational Methods

|                                 |                                                        |
|---------------------------------|--------------------------------------------------------|
| <a href="#">meiFlashControl</a> | Return handle of Control that is associated with Flash |
|---------------------------------|--------------------------------------------------------|



## Data Types

[MEIFlashConfig](#)

[MEIFlashFileMaxNameChars](#)

[MEIFlashFileMaxChars](#)

[MEIFlashFileMaxPathChars](#)

[MEIFlashFiles](#)

[MEIFlashFileType](#)

[MEIFlashMessage](#)

[MEIFlashSection](#)

# meiFlashCreate

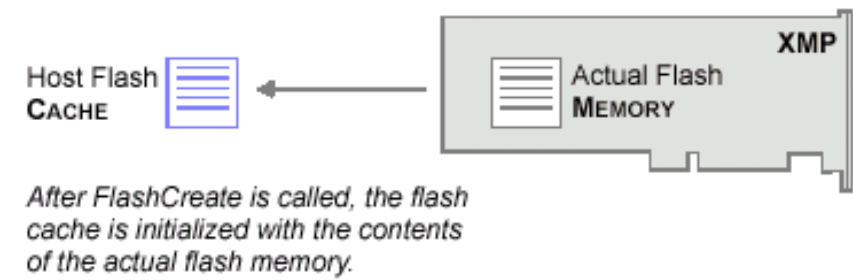
## Declaration

```
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.



| Return Values |                                    |
|---------------|------------------------------------|
| handle        | to a Flash object                  |
| MPIHandleVOID | if the object could not be created |

## See Also

[meiFlashDelete](#) | [meiFlashValidate](#)

# 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**

if *FlashDelete* successfully deletes a Flash object and invalidates its handle

## See Also

[meiFlashCreate](#) | [meiFlashValidate](#)

# meiFlashValidate

## Declaration

```
long meiFlashValidate(MEIFlash flash)
```

**Required Header:** stdmei.h

## Description

**meiFlashValidate** validates the Flash object and its handle (***flash***).

### Return Values

|                     |                                         |
|---------------------|-----------------------------------------|
| <b>MPIMessageOK</b> | if Flash is a handle to a valid object. |
|---------------------|-----------------------------------------|

## See Also

[meiFlashCreate](#) | [meiFlashDelete](#)

# meiFlashConfigGet

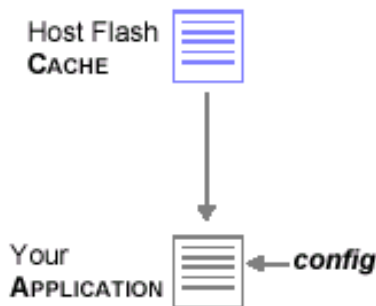
## Declaration

```
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

if *FlashConfigGet* successfully gets the flash configuration and writes it into the structure

## See Also

[meiFlashConfigSet](#)

# meiFlashConfigSet

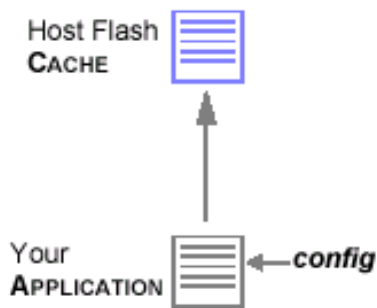
## Declaration

```
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**

if *FlashConfigSet* successfully reads the control configuration from the structure

## See Also

[meiFlashConfigGet](#)

# meiFlashDataGet

## Declaration

```
long meiFlashDataGet(MEIFlash flash,
                    void      *dst,
                    void      *src,
                    long      count)
```

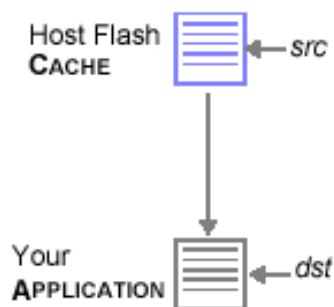
**Required Header:** stdmei.h

## 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

if *FlashDataGet* successfully gets count bytes of flash data memory writes them to application memory

## See Also

[mpiControlMemory](#) | [meiFlashMemoryGet](#) | [meiFlashDataSet](#)

# meiFlashDataSet

## Declaration

```
long meiFlashDataSet(MEIFlash flash,
                    void      *dst,
                    void      *src,
                    long      count)
```

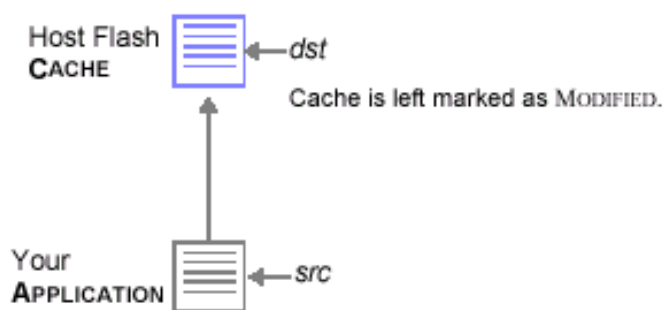
**Required Header:** stdmei.h

## 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 FlashDataSet method to access flash data memory.



### Return Values

**MPIMessageOK**

if *FlashDataSet* successfully writes to the flash cache

## See Also

[mpiControlMemory](#) | [meiFlashDataSet](#)



# 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 recommended that you use the [meiFlashMemoryFromFileType\(\)](#) method instead.

|                   |                                                    |
|-------------------|----------------------------------------------------|
| <b>flash</b>      | a handle to a flash object                         |
| <b>*fileImage</b> | a pointer to data in memory to be copied to flash. |
| <b>*fileType</b>  | where the fileImage will be copied to in flash.    |

| Return Values                           |                                                                                              |
|-----------------------------------------|----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                     | if <i>FlashMemoryFromFileImage</i> successfully erases/programs the controller flash memory. |
| <b>MEIFlashMessageFLASH_INVALID</b>     | if the flash handle is invalid.                                                              |
| <b>MPIMessageNO_MEMORY</b>              | the system could not allocate enough memory for temporary storage of the flash image.        |
| <b>MPIMessageTIMEOUT</b>                | the flash could not be programmed.                                                           |
| <b>MEIFlashMessageFLASH_WRITE_ERROR</b> | the programmed flash did not match the flash image in verification.                          |

## 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  | if <i>FlashMemoryFromFileType</i> successfully copies the data from the file to the flash. |

## See Also

[meiFlashCreate](#) | [meiFlashMemoryFromFile](#)

# meiFlashMemoryGet

## Declaration

```
long meiFlashMemoryGet(MEIFlash flash,
                       void      *dst,
                       void      *src,
                       long      count)
```

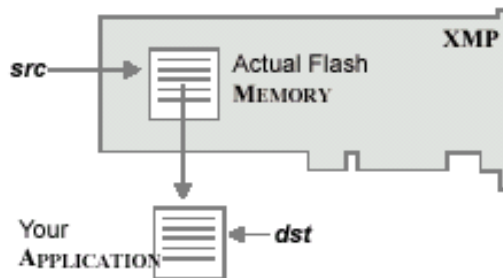
**Required Header:** stdmei.h

## 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**

if *FlashMemoryGet* successfully copies flash memory to application memory

## 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.

| <i>If the "flash cache"</i> | <i>Then</i>                                                                    |
|-----------------------------|--------------------------------------------------------------------------------|
| has been modified           | <i>FlashMemoryModified</i> writes True to the location pointed to by modified  |
| has not been modified       | <i>FlashMemoryModified</i> writes False to the location pointed to by modified |

| Return Values       |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>FlashMemoryModified</i> successfully determines if the flash cache has been modified |

## See Also

[meiFlashDataSet](#)

# meiFlashMemorySet

## Declaration

```
long meiFlashMemorySet(MEIFlash flash,
                        void      *dst,
                        void      *src,
                        long      count)
```

**Required Header:** stdmei.h

## 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**

if *FlashMemorySet* successfully copies application memory to flash memory

## See Also

[meiFlashMemoryGet](#)

# 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 <a href="#">MEIFlashFiles</a> .                       |
| *filesOut | a pointer to an array of flash filename structures that were successfully downloaded to the controller. See <a href="#">MEIFlashFiles</a> . |

| Return Values |                                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>FlashMemoryFromFile</i> successfully writes to actual flash memory using the file or using flash cache. |

## See Also

[MEIFlashFiles](#) | [mpiControlReset](#)

# meiFlashMemoryToFile

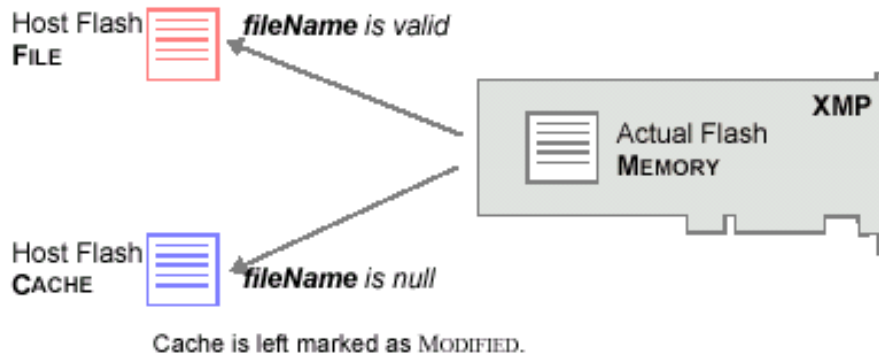
## Declaration

```
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**

if *FlashMemoryToFile* successfully saves actual flash memory to file or to cache

## See Also



# 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                    | if <i>FlashMemoryVerify</i> successfully verifies the Flash memory against an input file. |
| MEIFlashMessageFLASH_READ_ERROR | if the input file does not match what is in flash memory.                                 |

## See Also

# meiFlashControl

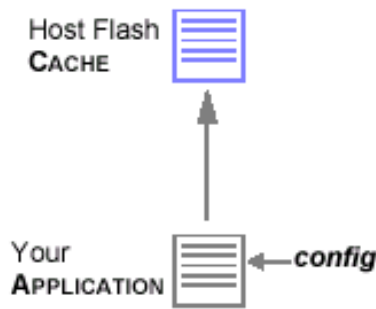
## Declaration

```
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.

|                   |                                                                                      |
|-------------------|--------------------------------------------------------------------------------------|
| <b>wordSize</b>   | The size of a flash word in bytes.                                                   |
| <b>sectorSize</b> | The size of a flash sector in bytes.                                                 |
| <b>extMemSize</b> | The size of the controller's external memory in bytes.                               |
| <b>all</b>        | The flash section parameters for the controller's code, data, and local FPGA images. |
| <b>code</b>       | The flash section parameters for the controller's boot and main code.                |
| <b>codeBoot0</b>  | The flash section parameters for the controller's boot and main code. (ZMP)          |
| <b>codeBoot</b>   | The flash section parameters for the controller's boot code.                         |
| <b>codeMain</b>   | The flash section parameters for the controller's main code.                         |

|                |                                                                         |
|----------------|-------------------------------------------------------------------------|
| <b>data</b>    | The flash section parameters for the controller's internal data memory. |
| <b>dataExt</b> | The flash section parameters for the controller's external data memory. |
| <b>synqNet</b> | This is for the controller's synqnet data memory.                       |
| <b>FPGA0</b>   | 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 FGAs
                               (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(...)`.

|                                    |                                              |
|------------------------------------|----------------------------------------------|
| <b>MEIFlashFileTypeCode</b>        | Controller processor code only.              |
| <b>MEIFlashFileTypeDataInt</b>     | Controller internal data only.               |
| <b>MEIFlashFileTypeDataExt</b>     | Controller external data only.               |
| <b>MEIFlashFileTypeSynqNet</b>     | Used for flashing the synqNet page of flash. |
| <b>MEIFlashFileTypeCodeAndData</b> | Controller processor code and data.          |
| <b>MEIFlashFileTypeFPGA0</b>       | Local FPGA image number 0.                   |
| <b>MEIFlashFileTypeALL</b>         | 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 initialization, 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.

|                    |                                      |
|--------------------|--------------------------------------|
| <b>*address</b>    | A pointer to a flash sector address. |
| <b>size</b>        | Length of the flash memory in bytes. |
| <b>sectorIndex</b> | 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

[`MPIAction` / `MEIAction`](#)

[`MEIDataType`](#)

[`MPIIoSource`](#)

[`MPIIoType`](#)

[`MPIIoTrigger`](#)

[`MEIMaxBiQuadSections`](#)

[`MPIModuleId` / `MEIModuleId`](#)

[`MEINetworkPort`](#)

[`MEINetworkType`](#)

[`MPIState`](#)

[`MPIStatus`](#)

[`MEIStatusFlag`](#)

[`MPIStatusMask` / `MEIStatusMask`](#)

[`MPITrajectory`](#)

[`MPIWait`](#)

## Macros

[`mpiStatusMaskBIT`](#)

# MPIAction / MEI Action

## Definition: MPIAction

```
typedef enum {
    MPIActionINVALID,

    MPIActionNONE,
    MPIActionSTOP,
    MPIActionE_STOP,
    MPIActionE_STOP_ABORT,
    MPIActionE_STOP_CMD_EQ_ACT,
    MPIActionABORT,

    MPIActionDONE,
    MPIActionSTART,
    MPIActionRESUME,
    MPIActionRESET,
    MPIActionCANCEL_REPEAT,
} MPIAction;
```

## 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.

|                        |                                                                                                                                                                                                                                             |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIActionNONE</b>   | Performs no action. Use with MPIMotorEventConfig to prevent a motor event from performing an action.                                                                                                                                        |
| <b>MPIActionSTOP</b>   | Makes a motion supervisor perform a stop. This action can be commanded with <code>mpiMotionAction(.)</code> or by a motor event on the controller. Please see <code>MPIMotionDecelTime</code> for more information about stop actions.      |
| <b>MPIActionE_STOP</b> | Makes a motion supervisor perform an e-stop. This action can be commanded with <code>mpiMotionAction(.)</code> or by a motor event on the controller. Please see <code>MPIMotionDecelTime</code> for more information about e-stop actions. |

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIActionE_STOP_ABORT</b>      | 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.                                                                                                                                                                                                                                  |
| <b>MPIActionE_STOP_CMD_EQ_ACT</b> | 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.                                                                                                                                                                                                   |
| <b>MPIActionABORT</b>             | Makes a motion supervisor perform an abort. This action can be commanded with mpiMotionAction(.) or by a motor event on the controller.                                                                                                                                                                                                                                                                                                                              |
| <b>MPIActionDONE</b>              | is currently not supported and is reserved for future use.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>MPIActionSTART</b>             | 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.                                                                                                                                                                                                                                                                                                          |
| <b>MPIActionRESUME</b>            | 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(.).                                                                                                                                                                                                                                             |
| <b>MPIActionRESET</b>             | 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(.). |
| <b>MPIActionCANCEL_REPEAT</b>     | 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 <a href="#">Repeating Cams</a> .                                                                                                                                                                                                                                                      |

## Remarks

An **MPIAction** can be generated from the host or the firmware. Below is a table where MPIActions originate (start):

| <b>MPIAction</b> | <b>Originating from Host</b> | <b>Originating from XMP Firmware</b> |
|------------------|------------------------------|--------------------------------------|
|------------------|------------------------------|--------------------------------------|

|        |                                            |       |
|--------|--------------------------------------------|-------|
| 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

|                     |                                                                                                                                                                          |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIActionMAP</b> | 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](#)



# MEIDataType

## Definition

```
typedef enum {
    MEIDataTypeINVALID = 0, /* this should stay zero - static
                             arrays init to zero by default */

    MEIDataTypeCHAR,
    MEIDataTypeSHORT,
    MEIDataTypeUSHORT,
    MEIDataTypeLONG,
    MEIDataTypeULONG,
    MEIDataTypeFLOAT,
    MEIDataTypeDOUBLE,
} MEIDataType;

static MEIDataType  MEIFilterGainTypeID[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 03.02.00

## Description

**MEIDataType** is an enumeration of data types for the filter coefficients.

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <b>MEIDataTypeCHAR</b>   | character filter data type                       |
| <b>MEIDataTypeSHORT</b>  | short integer filter data type                   |
| <b>MEIDataTypeUSHORT</b> | unsigned short integer filter data type          |
| <b>MEIDataTypeLONG</b>   | long integer filter data type                    |
| <b>MEIDataTypeULONG</b>  | unsigned long filter data type                   |
| <b>MEIDataTypeFLOAT</b>  | floating point filter data type                  |
| <b>MEIDataTypeDOUBLE</b> | double precision floating point filter data type |

## 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.

|              |                                                                                                                                       |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b> | Handle to a motor object that is the source for the I/O.                                                                              |
| <b>index</b> | 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,  
    MPIIoTypeUSER,  
} MPIIoType;
```

## Description

**MPIIoType** is an enumeration of data types for the filter coefficients.

|                       |                                                                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIIoTypeMotor</b> | Value specifies the I/O type as motor (the I/O source is a motor object).                                                                    |
| <b>MPIIoTypeUSER</b>  | Value specifies the I/O type as user ( <b>User I/O types are currently not supported. User I/O is available through the motor objects</b> ). |

## See Also

# MPIIoTrigger

## Definition

```
typedef struct MPIIoTrigger {  
    MPIIoType          type;  
    MPIIoSource       source;  
    unsigned long     mask;  
    unsigned long     pattern;  
} MPIIoTrigger;
```

## Description

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <b>type</b>    | See <a href="#">MPIIoType</a> .                                   |
| <b>source</b>  | See <a href="#">MPIIoSource</a> .                                 |
| <b>mask</b>    | Value that specifies the mask to be applied to the IO.            |
| <b>pattern</b> | Value that specifies the pattern to be compared to the masked IO. |

## 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](#)



# 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_TERMINATED,
    MEINetworkTypeRING,
} MEINetworkType;
```

## Description

**MEINetworkPort** enumerations list all network topologies supported by this MPI release.

**NOTE:** Due to the un-terminated nature of the MEINetworkTypeSTRING network type discovering a topology of this type requires a timeout period to detect the end of the string. Thus discovering a string topology will take a longer time than other networks types.

|                                 |                                                                                            |
|---------------------------------|--------------------------------------------------------------------------------------------|
| MEINetworkTypeINVALID           | No nodes were found on the network                                                         |
| MEINetworkTypeSTRING            | The network topology type is a string of nodes                                             |
| MEINetworkTypeSTRING_TERMINATED | The network topology type is a string of nodes. The last node on the string is terminated. |
| MEINetworkTypeRING              | The network topology type is a ring of nodes.                                              |

## See Also

[MEISynqNetInfo](#) | [meiSynqNetInfo](#)

# 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.

|                               |                                                                                                                                                                                                      |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIStateIDLE</b>           | The state of motion is idle and waiting to resume motion.                                                                                                                                            |
| <b>MPIStateMOVING</b>         | The state of the motion is moving.                                                                                                                                                                   |
| <b>MPIStateSTOPPING</b>       | 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.                     |
| <b>MPIStateSTOPPED</b>        | 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. |
| <b>MPIStateSTOPPING_ERROR</b> | 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.).                                                         |
| <b>MPIStateERROR</b>          | 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;  
  
    long            settled;  
    long            atTarget;  
  
    MPIStatusMask     statusMask;  
} MPIStatus;
```

## 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.

|                   |                                                                                                                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>state</b>      | Value that indicates the state of an xmp controller's motion supervisor.                                                                                                             |
| <b>action</b>     | Value that indicates the action to perform for a motion supervisor.                                                                                                                  |
| <b>eventMask</b>  | Array that defines the event mask bits. The array is defined as <code>typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS]</code> .                                   |
| <b>settled</b>    | Value that indicates if an axis associated with a motion supervisor has settled (is in fine position).                                                                               |
| <b>atTarget</b>   | 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). |
| <b>statusMask</b> | Value is an enumeration of bit masks for the MEIStatusFlags. The status masks represent the present condition for an object.                                                         |

## See Also

[MPIState](#) | [MPIStatusMask](#)

# MEIStatusFlag

## Definition

```
typedef enum {  
    MEIStatusFlagBROKEN_WIRE,           /* 1 */  
    MEIStatusFlagILLEGAL_STATE,         /* 2 */  
    MEIStatusFlagABS_ENCODER_FAULT,     /* 3 */  
    MEIStatusFlagABS_ENCODER_TIMEOUT,   /* 4 */  
    MEIStatusFlagBROKEN_WIRE_SECONDARY, /* 5 */  
    MEIStatusFlagILLEGAL_STATE_SECONDARY, /* 6 */  
} MEIStatusFlag;
```

## 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.

|                           |                                                                                 |
|---------------------------|---------------------------------------------------------------------------------|
| <b>MPIStatusMaskMOTOR</b> | Value specifies the motor's status mask.                                        |
| <b>MPIStatusMaskALL</b>   | 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.

|                                           |                                                                                                                                                                                                                                                                      |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIStatusMaskBROKEN_WIRE</b>           | 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.                 |
| <b>MEIStatusMaskILLEGAL_STATE</b>         | 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. |
| <b>MEIStatusMaskABS_ENCODER_FAULT</b>     | Absolute encoder initialization failure. Occurs when the hardware fails to read the absolute position information from an encoder.                                                                                                                                   |
| <b>MEIStatusMaskABS_ENCODER_TIMEOUT</b>   | Absolute encoder response timeout. Occurs when the encoder fails to respond to a request for absolute position data.                                                                                                                                                 |
| <b>MEIStatusMaskBROKEN_WIRE_SECONDARY</b> | 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.               |

|                                             |                                                                                                                                                                                                                                                                 |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIStatusMaskILLEGAL_STATE_SECONDARY</b> | 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. |
| <b>MEIStatusMaskMOTOR</b>                   | Bit mask containing all of the motor specific MEIStatusFlags set.                                                                                                                                                                                               |
| <b>MEIStatusMaskALL</b>                     | 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.

|                         |                                                                                                                                                                                                                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>velocity</b>         | 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.                                                                                                                                           |
| <b>acceleration</b>     | 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. |
| <b>deceleration</b>     | Rate of change of velocity. Specifies the final ramp to reach zero velocity for TRAPEZOIDAL and VELOCITY 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. Units are counts per second * second.                              |
| <b>jerkPercent</b>      | 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.                                                                                                                               |
| <b>accelerationJerk</b> | 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. 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.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIWaitFOREVER</b> | Makes MPI methods wait forever for an event to occur before returning.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>MPIWaitPOLL</b>    | 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 <code>MPIMessageTIMEOUT</code> .                                                                                                                                                                                                                                                                                                                                                                          |
| <b>MPIWaitMSEC</b>    | 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 <code>MPIWaitMSEC</code> to make MPI methods wait longer periods of time. For example, the following statement will make <code>mpiPlatformKey</code> wait 5 milliseconds for a user keystroke: <code>mpiPlatformKey( 5 * MPIWaitMSEC );</code> If an event does not occur within the specified time, MPI methods will generally return the value <code>MPIMessageTIMEOUT</code> . |

## 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

[mpiControllInterruptWait](#) | [mpiNotifyEventWait](#) | [mpiObjectTimeoutGet](#)  
[mpiObjectTimeoutSet](#) | [meiPlatformKey](#)

# mpiStatusMaskBIT

## Declaration

```
#define    mpiStatusMaskBIT(flag) (0x1 << (flag))
```

**Required Header:** stdmpi.h

## Description

**mpiStatusMaskBIT** converts the status flag into the status mask.

## See Also

[MPIStatusMask](#)

# 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](#)

[mpiMessageFunction](#)

Associate message text with a function

## Data Types

[MPIMessage / MEIMessage](#)

[MPIMessageFunction](#)

## Macros

[mpiMessageID](#)

[mpiMessageMODULE](#)

[mpiMessageNUMBER](#)

# 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                | <i>Message</i> returns a pointer to message text resident in the library                                                                                 |
| not NULL            | <i>Message</i> returns the pointer <i>messageText</i> , 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                                                                                                       |
| <b>Pointer to an empty string (" ")</b>    | if the message utility is not enabled or the message function pointer associated with <i>messageId</i> is NULL <sup>1</sup> . |
| <b>Pointer to a non-zero-length string</b> | if the mpiMessage successfully retrieves the message associated with <i>messageId</i> .                                       |

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**

if *MessageFunction* successfully registers ***function*** as the function to be called by **mpiMessage(...)**

## 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

|                                   |                        |
|-----------------------------------|------------------------|
| <a href="#">mpiMotionCreate</a>   | Create Motion object   |
| <a href="#">mpiMotionDelete</a>   | Delete Motion object   |
| <a href="#">mpiMotionValidate</a> | Validate Motion object |

### Configuration and Information Methods

|                                            |                                                   |
|--------------------------------------------|---------------------------------------------------|
| <a href="#">mpiMotionConfigGet</a>         | Get Motion configuration                          |
| <a href="#">mpiMotionConfigSet</a>         | Set Motion configuration                          |
| <a href="#">mpiMotionFlashConfigGet</a>    | Get Motion flash config                           |
| <a href="#">mpiMotionFlashConfigSet</a>    | Set Motion flash config                           |
| <a href="#">meiMotionFrameBufferStatus</a> | Get a Motion object (motion) frame buffer status. |
| <a href="#">meiMotionLogClose</a>          | Frees memory buffer for motion logging            |
| <a href="#">meiMotionLogOpen</a>           | Allocates a memory buffer for motion logging.     |
| <a href="#">meiMotionLogPrint</a>          | Writes motion log buffer data                     |
| <a href="#">mpiMotionParamsGet</a>         | Get Motion parameters                             |

[mpiMotionParamsSet](#)

Set Motion parameters

[mpiMotionParamsValidate](#)

Validate Motion parameters

[mpiMotionPositionGet](#)

Get position parameters of all axes associated with Motion

[mpiMotionPositionSet](#)

Set position parameters of all axes associated with Motion

[mpiMotionStatus](#)

Get Motion status

[mpiMotionTrajectory](#)

Get trajectories for all Axis associated with Motion

## Event Methods

[mpiMotionEventNotifyGet](#)

Get event mask

[mpiMotionEventNotifySet](#)

Set event mask

[mpiMotionEventReset](#)

Reset events specified in event mask

## Action Methods

[mpiMotionAction](#)

Perform an action on a Motion

[mpiMotionActionFunction](#)[mpiMotionModify](#)

Modify parameters of Motion while it is executing

[mpiMotionStart](#)

Start Motion (idle state &gt; moving state)

## Memory Methods

[mpiMotionMemory](#)

Set address to be used to access Motion memory

[mpiMotionMemoryGet](#)

Get bytes of Motion memory and place it into application memory

[mpiMotionMemorySet](#)

Put (set) bytes of application memory into Motion memory

## Relational Methods

[mpiMotionControl](#)

Return handle of Control object associated with Motion

[mpiMotionNumber](#)

Get index of Motion

[mpiMotionAxis](#)

Return handle of axis by index number

[mpiMotionAxisAppend](#)

Append axis to list

[mpiMotionAxisCount](#)

Return number of axes in list

[mpiMotionAxisFirst](#)

Return handle to first axis in list

[mpiMotionAxisIndex](#)

Return index value of an axis in list

[mpiMotionAxisInsert](#)

Insert axis into list associated with Motion

[mpiMotionAxisLast](#)

Return handle to last axis in list

[mpiMotionAxisListGet](#)

Get list of axes associated with Motion

[mpiMotionAxisListSet](#)

Create a list of axes associated with Motion

[mpiMotionAxisNext](#)

Get handle to next axis in list

[mpiMotionAxisPrevious](#)

Get handle to previous axis in list

[mpiMotionAxisRemove](#)

Remove handle to axis in list

# Data Types

[\\*MEIMotionActionFunction](#)

[MPIMotionAttr](#) / [MEIMotionAttr](#)

[MEIMotionAttrHold](#)

[MEIMotionAttrHoldSource](#)

[MEIMotionAttrHoldType](#)

[MPIMotionAttrMask](#) / [MEIMotionAttrMask](#)

[MEIMotionAttrOutput](#)

[MEIMotionAttrOutputType](#)

[MPIMotionAttributes](#) / [MEIMotionAttributes](#)

[MPIMotionBESSEL](#)

[MPIMotionBSPLINE](#)

[MPIMotionCam](#)

[MPIMotionConfig](#) / [MEIMotionConfig](#)

[MPIMotionDecelTime](#)

[MEIMotionFrame](#)

[MEIMotionFrameBufferStatus](#)

[MPIMotionMessage](#) / [MEIMotionMessage](#)

[MPIMotionParams](#) / [MEIMotionParams](#)

[MPIMotionPoint](#)

[MPIMotionPT](#)

[MPIMotionPTF](#)

[MPIMotionPVT](#)

[MPIMotionPVTF](#)

[MPIMotionSCurve](#)

[MPIMotionSCurveJerk](#)

[MPIMotionSPLINE](#)

[MEIMotionTrace](#)

[MPIMotionTrapezoidal](#)

[MPIMotionType](#) / [MEIMotionType](#)

[MPIMotionVelocity](#)

# Macros

[mpiMotionATTR](#)

[mpiMotionAttrMaskBIT](#)

[mpiMotionTYPE](#)

# 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

|                      |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | handle to a Motion object          |
| <b>MPIHandleVOID</b> | if the object could not be created |

## See Also

[mpiMotionDelete](#) | [mpiMotionValidate](#)

# mpiMotionDelete

## Declaration

```
long mpiMotionDelete(MPI\_Motion 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

| Return Values       |                                                                                        |
|---------------------|----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionDelete</i> successfully deletes a Motion object and invalidates its handle |

## 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

|                     |                                          |
|---------------------|------------------------------------------|
| <b>MPIMessageOK</b> | if Motion is a handle to a valid object. |
|---------------------|------------------------------------------|

## See Also

[mpiMotionCreate](#) | [mpiMotionDelete](#)

# mpiMotionConfigGet

## Declaration

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

**Required Header:** stdmpi.h

## Description

**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

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

## See Also

[mpiMotionConfigSet](#) | [MEIMotionConfig](#)

# mpiMotionConfigSet

## Declaration

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

**Required Header:** stdmpi.h

## Description

**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

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

## See Also

[mpiMotionConfigGet](#) | [MEIMotionConfig](#)

# mpiMotionFlashConfigGet

## Declaration

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

Required Header: stdmpi.h

## Description

**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.

| Return Values       |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionFlashConfigGet</i> successfully gets the Motion's flash configuration and writes it into the structure(s). |

## See Also

[mpiMotionFlashConfigSet](#)

# mpiMotionFlashConfigSet

## Declaration

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

Required Header: stdmpi.h

## Description

**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       |                                                                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionFlashConfigSet</i> successfully sets (writes) the Motion's flash configuration using data from the structure(s) |

## See Also

[MEIMotionConfig](#) | [MEIFlash](#) | [mpiMotionFlashConfigGet](#)

# mpiMotionFrameBufferStatus

## Declaration

```
long meiMotionFrameBufferStatus(MPI\_Motion motion,  
                                MEI\_MotionFrameBufferStatus *status)
```

Required Header: stdmei.h

## Description

**mpiMotionFrameBufferStatus** gets a Motion object (motion) frame buffer status and writes it to the structure pointed to by status.

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| <b>motion</b>  | a handle to the Motion object                                                |
| <b>*status</b> | a pointer to the motion frame buffer status structure returned by the method |

| Return Values       |                                                                             |
|---------------------|-----------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionFrameBufferStatus</i> successfully gets the frame buffer status |

## See Also

[MEI\\_MotionFrameBufferStatus](#)

# meiMotionLogClose

## Declaration

```
long meiMotionLogClose(MPI\_Motion motion);
```

**Required Header:** stdmei.h

## Description

**meiMotionLogClose** frees the memory buffer for motion logging.

|               |                               |
|---------------|-------------------------------|
| <b>motion</b> | a handle to the Motion object |
|---------------|-------------------------------|

| Return Values        |                                                                                   |
|----------------------|-----------------------------------------------------------------------------------|
| <b>MPI_MessageOK</b> | if <i>MotionLogClose</i> successfully frees the memory buffer for motion logging. |

## 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.

|                   |                                                                                      |
|-------------------|--------------------------------------------------------------------------------------|
| <b>motion</b>     | a handle to the Motion object                                                        |
| <b>maxEntries</b> | The maximum number of log entries. This value determines the log buffer memory size. |

| Return Values       |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionLogOpen</i> successfully allocates a memory buffer for motion logging. |

## See Also

[meiMotionLogClose](#) | [meiMotionLogPrint](#)



# meiMotionLogPrint

## Declaration

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

Required Header: stdmei.h

## Description

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

|                  |                                  |
|------------------|----------------------------------|
| <b>motion</b>    | a handle to the Motion object    |
| <b>*filename</b> | a pointer to a file name string. |

| Return Values       |                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionLogPrint</i> successfully writes the data into a file specified by the <i>fileName</i> . |

## See Also

[meiMotionLogOpen](#) | [meiMotionLogClose](#)

# mpiMotionParamsGet

## Declaration

```
long mpiMotionParamsGet(MPI\_Motion motion,
                        MPI\_MotionParams *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.

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <b>motion</b>  | a handle to the Motion object                                       |
| <b>*params</b> | a pointer to the motion parameters structure returned by the method |

| Return Values            |                                                                                                                                                               |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>      | if <i>MotionParamsGet</i> successfully returns the parameters associated with a Motion object                                                                 |
| <b>motion parameters</b> | that are associated with a Motion object ( <i><b>motion</b></i> ). To set these motion parameters, call either mpiMotionParamsSet(...) or mpiMotionStart(...) |

## See Also

[mpiMotionParamsSet](#) | [mpiMotionStart](#) | [meiMotionParamsValidate](#)

# mpiMotionParamsSet

## Declaration

```
long mpiMotionParamsSet(MPI\_Motion motion,
                        MPI\_Motion\_Params *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

|                     |                                                                               |
|---------------------|-------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionParamsSet</i> successfully sets the parameters of a Motion object |
|---------------------|-------------------------------------------------------------------------------|

## See Also

[mpiMotionStart](#) | [mpiMotionParamsGet](#)

# meiMotionParamsValidate

## Declaration

```
long meiMotionParamsValidate(MPI\_Motion motion,  
                             MPI\_MotionType type,  
                             MPI\_MotionParams *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 <i>params</i> is written to the location (pointed to by <i>points</i> ) |

| Return Values        |                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPI_MessageOK</b> | if <i>MotionParamsValidate</i> successfully validates the type-specific motion parameters using the coordinate system of <i>motion</i> |

## 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**

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

## 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**

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

## 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.

|                  |                                                   |
|------------------|---------------------------------------------------|
| <b>motion</b>    | a handle to a Motion object                       |
| <b>*status</b>   | a pointer to MPIStatus structure                  |
| <b>*external</b> | a pointer to an implementation-specific structure |

| Return Values                |                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>MotionStatus</i> successfully writes the Motion's status to the structure(s). |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL.                                               |

## 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**

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

## 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(MPI\_Motion motion,  
                             MPI\_EventMask *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 MPI\_EventMask bits associated with the desired MPI\_EventType values.  
Event notification is disabled for event types not specified in **eventmask**.

## Remarks

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

| Return Values       |                                                                                      |
|---------------------|--------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionEventNotifyGet</i> successfully writes the event mask to the location(s) |

## See Also

[MPI\\_EventType](#) | [MEI\\_EventNotifyData](#) | [MEI\\_EventStatusInfo](#) | [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**

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

**See Also**

[MPIEventType](#) | [MEIEventNotifyData](#) | [MEIEventStatusInfo](#) | [mpiEventMaskMOTION](#) | [mpiEventMaskAXIS](#) | [mpiMotionEventNotifyGet](#)

# mpiMotionEventReset

## Declaration

```
long mpiMotionEventReset(MPI\_Motion motion,
                        MPI\_EventMask 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       |                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionEventReset</i> successfully resets the event(s) that are specified in <b><i>eventMask</i></b> and generated by <b><i>motion</i></b> |

## See Also

# mpiMotionAction

## Declaration

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

Required Header: stdmpi.h

## Description

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

The deceleration stop and Estop times can be set with:

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

where the structure MPIMotionConfig, contains the elements:

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

In the 20030818 and later releases, mpiMotionAction(...) was optimized for the MPIActionRESET action type. To make the execution as fast as possible, service commands to clear node status, CRCs, and the packet error counters were removed. Service commands to clear amp and encoder faults are only sent to the node when an amp fault or encoder fault is active.

The method, [meiSqNodeStatusClear\(...\)](#) was added to clear node faults. It sends service commands to the node in order to clear the node status, CRCs, packet error counters, and hardware latches for the nodeDisable and powerFault.

| If "action" is                                                                        | Then                                                                                                                                                                                                                                                     |
|---------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIActionABORT</b> or<br><b>MPIActionE_STOP</b> or<br><b>MPIActionE_STOP_ABORT</b> | 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). |
| <b>MPIActionABORT</b>                                                                 | 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.                                        |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIActionE_STOP</b>       | MotionAction will decelerate the axis (or axes) to a stop in the time specified by the "eStop" time. After the Motion stops, the Motion will be set to the ERROR state.                                                                                                                                                                                                                                             |
| <b>MPIActionE_STOP_ABORT</b> | 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. |
| <b>MPIActionRESET</b>        | 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.                                   |
| <b>MPIActionRESUME</b>       | 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).                                                                                                                                                                                                     |
| <b>MPIActionSTOP</b>         | if the Motion is in the moving state (MPIStateMOVING), it will performing a stop on all axes and change to the idle state (MPIStateSTOPPING to MPIStateIDLE). MPIActionSTOP decelerates the axis (or axes) to a stop in the time specified by the "stop" time. After the motion stops, the Motion is set to the ERROR state.                                                                                        |
| <b>MEIActionMAP</b>          | 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

|                     |                                                                                        |
|---------------------|----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionAction</i> successfully performs the specified action on the Motion object |
|---------------------|----------------------------------------------------------------------------------------|

## See Also

# meiMotionActionFunction

## Declaration

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

Required Header: stdmei.h

## Description

### meiMotionActionFunction

|          |                               |
|----------|-------------------------------|
| motion   | a handle to the Motion object |
| function |                               |

| Return Values |                                             |
|---------------|---------------------------------------------|
| MPIMessageOK  | if <i>MotionActionFunction</i> successfully |

## 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).

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                     |                                                                                                                                                                                            |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>               | if <i>MotionModify</i> successfully modifies the parameters of a Motion object                                                                                                             |
| <b>MPIMotionMessageIDLE</b>       | if no Motion was in progress when <i>MotionModify</i> was called. In order for <i>MotionModify</i> to work, there must be a Motion in progress.                                            |
| <b>MPIMotionMessageAUTO_START</b> | if <i>MotionModify</i> was called when no motion was in progress and the Auto-Start attribute was specified. In this case, the MotionModify is automatically converted into a MotionStart. |



|                                              |                                                                                                                 |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionMessagePROFILE_ERROR</b>         | if the controller cannot generate the motion profile (based on the specified motion parameters and attributes). |
| For more Returns, <a href="#">click here</a> |                                                                                                                 |

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                                |                                                                                                 |
|----------------------------------------------|-------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                          | <i>MotionStart</i> successfully changes a Motion object from the idle state to the moving state |
| <b>MPIMotionMessageMOVING</b>                | if <i>MotionStart</i> was called when a Motion is in progress                                   |
| For more Returns, <a href="#">click here</a> |                                                                                                 |

## 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

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

## See Also

[mpiMotionMemoryGet](#) | [mpiMotionMemorySet](#)

# mpiMotionMemoryGet

## Declaration

```
long mpiMotionMemoryGet(MPI\_Motion motion,
                        void      *dst,
                        void      *src,
                        long      count)
```

Required Header: stdmpi.h

## Description

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

| Return Values       |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionMemoryGet</i> successfully copies <i>count</i> bytes of a Motion's memory to application memory |

## See Also

[mpiMotionMemorySet](#) | [mpiMotionMemory](#)

# mpiMotionMemorySet

## Declaration

```
long mpiMotionMemorySet(MPI\_Motion motion,  
                        void      *dst,  
                        void      *src,  
                        long      count)
```

**Required Header:** stdmpi.h

## 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

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

## 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.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
|---------------|--------------------------------|

| Return Values        |                             |
|----------------------|-----------------------------|
| <b>MPIControl</b>    | handle to a Control object  |
| <b>MPIHandleVOID</b> | if <i>motion</i> is invalid |

## See Also

[mpiMotionCreate](#) | [mpiControlCreate](#)

# mpiMotionNumber

## Declaration

```
long mpiMotionNumber(MPI\_Motion 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**

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

## 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*.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
| <b>index</b>  | a position in the list.        |

| Return Values                          |                                                                                                                                                |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                          | to the <i>index</i> th Axis of a Motion ( <i>motion</i> )                                                                                      |
| <b>MPIHandleVOID</b>                   | if <i>motion</i> is invalid<br>if <i>index</i> is less than 0<br>if <i>index</i> is greater than or equal to <b>mpiMotionAxisCount(motion)</b> |
| <b>MPIMessageARG_INVALID</b>           | <i>index</i> is a negative number.                                                                                                             |
| <b>MEIListMessageELEMENT_NOT_FOUND</b> | <i>index</i> is greater than or equal to the number of elements in the list.                                                                   |
| <b>MPIMessageHANDLE_INVALID</b>        | <i>motion</i> is an invalid handle.                                                                                                            |

## 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.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
| <b>axis</b>   | a handle to an Axis object.    |

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

## See Also

# mpiMotionAxisCount

## Declaration

```
long mpiMotionAxisCount(MPI\_Motion motion)
```

Required Header: stdmpi.h

## Description

**mpiMotionAxisCount** returns the number of elements on the list.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
|---------------|--------------------------------|

| Return Values |                                       |
|---------------|---------------------------------------|
| <b>number</b> | of Axes in a Motion ( <i>motion</i> ) |
| <b>-1</b>     | if <i>motion</i> is invalid           |
| <b>0</b>      | if <i>motion</i> 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 conjunction with mpiMotionAxisNext() in order to iterate through the list.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
|---------------|--------------------------------|

| Return Values                   |                                                          |
|---------------------------------|----------------------------------------------------------|
| <b>handle</b>                   | to the first Axis of a Motion ( <i>motion</i> )          |
| <b>MPIHandleVOID</b>            | if <i>motion</i> is invalid<br>if <i>motion</i> is empty |
| <b>MPIMessageHANDLE_INVALID</b> | <i>motion</i> is an invalid handle.                      |

## 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.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
| <b>axis</b>   | a handle to an Axis object.    |

| Return Values |                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------|
| <b>index</b>  | of an Axis ( <i>axis</i> ) in a Motion ( <i>motion</i> )                                        |
| <b>-1</b>     | if <i>motion</i> is invalid<br>if the Axis (axis) was not found in the Motion ( <i>motion</i> ) |

## 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***).

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
|---------------|--------------------------------|

### Return Values

|                     |                                                                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionAxisInsert</i> successfully inserts an Axis ( <i><b>insert</b></i> ) in a Motion ( <i><b>motion</b></i> ) following a specified Axis ( <i><b>axis</b></i> ) |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 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 conjunction with mpiMotionAxisPrevious() in order to iterate through the list backwards.

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
|---------------|--------------------------------|

| Return Values                   |                                                                         |
|---------------------------------|-------------------------------------------------------------------------|
| <b>handle</b>                   | to the last Axis of a Motion ( <i><b>motion</b></i> )                   |
| <b>MPIHandleVOID</b>            | if <i><b>motion</b></i> is invalid<br>if <i><b>motion</b></i> is empty  |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i><b>motion</b></i> or <i><b>axis</b></i> is an invalid handle. |

## 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***).

|               |                                |
|---------------|--------------------------------|
| <b>motion</b> | a handle to the Motion object. |
|---------------|--------------------------------|

| Return Values       |                                                                                        |
|---------------------|----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionAxisListGet</i> successfully gets the coordinate system of a Motion object |

## See Also

[mpiMotionAxisListSet](#) | [mpiMotionAxis](#)



# mpiMotionAxisListSet

## Declaration

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

**Required Header:** stdmpi.h

## Description

**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

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionAxisListSet</i> successfully creates a coordinate system |
|---------------------|----------------------------------------------------------------------|

## 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 conjunction with mpiMotionAxisFirst() in order to iterate through the list.

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

| Return Values            |                                                                                                                     |
|--------------------------|---------------------------------------------------------------------------------------------------------------------|
| handle                   | to the Axis just after the specified Axis ( <i>axis</i> ) in a Motion ( <i>motion</i> )                             |
| MPIHandleVOID            | if <i>motion</i> is invalid<br>if the specified Axis ( <i>axis</i> ) is the last axis in a Motion ( <i>motion</i> ) |
| MPIMessageHANDLE_INVALID | Either <i>motion</i> or <i>axis</i> is an invalid handle.                                                           |

## 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 conjunction with mpiMotionAxisLast() in order to iterate through the list backwards.

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

| Return Values            |                                                                                                                      |
|--------------------------|----------------------------------------------------------------------------------------------------------------------|
| handle                   | to the Axis preceding the specified Axis ( <i>axis</i> ) in a Motion ( <i>motion</i> )                               |
| MPIHandleVOID            | if <i>motion</i> is invalid<br>if the specified Axis ( <i>axis</i> ) is the first Axis in a Motion ( <i>motion</i> ) |
| MPIMessageHANDLE_INVALID | Either <i>motion</i> or <i>axis</i> is an invalid handle.                                                            |

## 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***).

| Return Values       |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotionAxisRemove</i> successfully removes the Axis from the Motion object |

## See Also

# MEIMotionActionFunction

## Definition

```
typedef long    /* Callback prototype */
    (*MEIMotionActionFunction)(MPI\_Motion      motion,
                                MPI\_MotionType   type,
                                MPI_XMPTAction    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.

|                  |                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------|
| <b>motion</b>    | The motion object handle used to call <code>mpiMotionAction()</code> will be passed to the callback function. |
| <b>type</b>      | The type of motion that the action was called to act upon will be passed to the callback function.            |
| <b>xmpAction</b> | 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`.

|                                |                                                                                                                                                                                                                                                                          |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionAttrDELAY</b>      | This bit enables a time delay (seconds) before the motion profile begins. This mask can be used with <code>mpiMotionStart(.)</code> . Motion with Modify is not supported with the DELAY attribute. Please see <a href="#">MPIMotionAttributes</a> for more information. |
| <b>MPIMotionAttrID</b>         | This bit enables an identification tag to be stored in the motion profile. Please see <a href="#">MPIMotionAttributes</a> for more information. This bit can be used with <code>mpiMotionStart(.)</code> and <code>mpiMotionModify(.)</code> .                           |
| <b>MPIMotionAttrELEMENT_ID</b> | This bit enables an identification tag to be stored in the path motion profiles. Please see <a href="#">MPIMotionAttributes</a> for more information.                                                                                                                    |
| <b>MPIMotionAttrRELATIVE</b>   | 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.                                          |
| <b>MPIMotionAttrSYNC_END</b>   | 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.                                                       |
| <b>MPIMotionAttrSYNC_START</b> | 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.                                                      |
| <b>MPIMotionAttrREPEAT</b>     | This attribute generates a repeating cam motion. If you use this attribute you need to fill in the repeatFrom field in the <a href="#">MPIMotionAttributes</a> structure. See <a href="#">Repeating Cams</a> .                                                           |



|                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionAttrMASTER_START</b>             | This attribute specifies 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 <a href="#">MPIMotionAttributes</a> structure. See <a href="#">Starting at Specific Master Positions</a> .                                                                                                                                                                          |
| <b>MPIMotionAttrREPEAT_NO_BACKTRACK</b>      | 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 <a href="#">Reversal of the Master - Backtracking</a> . |
| <b>MPIMotionAttrREPEAT_NO_BACKTRACK_HOLD</b> | 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 <a href="#">Reversal of the Master with NO_BACKTRACK_HOLD</a> .                                                                                             |

## 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.

|                                  |                                                                                                                                                                                                                                                                 |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIMotionAttrEVENT</b>        | This bit allows the user to specify an <code>MPIEventMask</code> during a motion.                                                                                                                                                                               |
| <b>MEIMotionAttrFINAL_VEL</b>    | This bit allows the user to specify a non-zero target velocity for point to point motion types.                                                                                                                                                                 |
| <b>MEIMotionAttrNO_REVERSAL</b>  | This bit prevents a motion profile from changing direction.                                                                                                                                                                                                     |
| <b>MEIMotionAttrHOLD</b>         | This bit prevents a motion profile from executing until the specified trigger conditions are met.<br><code>MPIMotionMessageATTRIBUTE_INVALID</code> will be returned if <code>MEIMotionAttrHOLD</code> is used with a <code>mpiMotionModify(...)</code> method. |
| <b>MEIMotionAttrHOLD_LESS</b>    | Motion attribute bit for less than or equal hold logic.<br><code>MPIMotionMessageATTRIBUTE_INVALID</code> will be returned if <code>MEIMotionAttrHOLD_LESS</code> is used with a <code>mpiMotionModify(...)</code> method.                                      |
| <b>MEIMotionAttrHOLD_GREATER</b> | Motion attribute bit for greater than or equal hold logic.<br><code>MPIMotionMessageATTRIBUTE_INVALID</code> will be returned if <code>MEIMotionAttrHOLD_GREATER</code> is used with a <code>mpiMotionModify(...)</code> method.                                |
| <b>MEIMotionAttrOUTPUT</b>       | 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 <a href="#">MEIMotionAttrHoldType</a> for more information.                                                        |
| source  | This value specifies the motion hold conditions. Please see <a href="#">MEIMotionAttrHoldSource</a> 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](#)

# 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

|                      |                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>gate</b>          | This value specifies the control gate number when MEIMotionAttrHoldTypeGATE is used. Valid values are between 0 and 31. See meiControlGateGet/Set(.) for more information.             |
| <b>input.input</b>   | This value specifies the input address when MEIMotionAttrHoldTypeINPUT is used.                                                                                                        |
| <b>input.mask</b>    | This value specifies the AND mask when MEIMotionAttrHoldTypeINPUT is used.                                                                                                             |
| <b>input.pattern</b> | 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. |
| <b>motor.number</b>  | This value specifies the motor number when MEIMotionAttrHoldTypeMOTOR is used.                                                                                                         |

|                      |                                                                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor.mask</b>    | This value specifies the AND mask when MEIMotionAttrHoldTypeMOTOR is used.                                                                                                                 |
| <b>motor.pattern</b> | 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.         |
| <b>axis.number</b>   | 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.                                                     |
| <b>axis.position</b> | A position comparison value. Must be specified when the AXIS_POSITION_COMMAND or AXIS_POSITION_ACTUAL motion hold types are used.                                                          |
| <b>user.address</b>  | A controller memory address. Must be specified when the USER_ADDRESS motion hold type is used.                                                                                             |
| <b>user.mask</b>     | A bitwise AND mask for the user specified controller memory address. Must be specified when the USER_ADDRESS motion hold type is used.                                                     |
| <b>user.pattern</b>  | 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

[MEIMotionAttrHoldType](#) | [meiControlGateGet](#) | [meiControlGateSet](#)

# MEIMotionAttrHoldType

## Definition

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

## 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                 | This type configures a motor's input bit(s) for the hold trigger condition.                                            |
| 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](#)

# 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

|                                    |                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionAttrMaskAPPEND</b>     | 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(.). |
| <b>MPIMotionAttrMaskAUTO_START</b> | 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.                                                                   |
| <b>MPIMotionAttrMaskDELAY</b>      | 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 <a href="#">MPIMotionAttributes</a> for more information.                                                                       |



|                                      |                                                                                                                                                                                                                                                                  |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionAttrMaskID</b>           | This mask enables an identification tag to be stored in the motion profile. Please see <a href="#">MPIMotionAttributes</a> for more information. This mask can be used with mpiMotionStart(.) and mpiMotionModify(.).                                            |
| <b>MPIMotionAttrMaskELEMENT_ID</b>   | This mask enables an identification tag to be stored in the path motion profiles. Please see <a href="#">MPIMotionAttributes</a> for more information.                                                                                                           |
| <b>MPIMotionAttrMaskRELATIVE</b>     | 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.                                 |
| <b>MPIMotionAttrMaskSYNC_END</b>     | 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.                                              |
| <b>MPIMotionAttrMaskSYNC_START</b>   | 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.                                             |
| <b>MPIMotionAttrMaskREPEAT</b>       | This attribute generates a repeating cam motion. If you use this attribute you need to fill in the repeatFrom field in the <a href="#">MPIMotionAttributes</a> structure. See <a href="#">Repeating Cams</a> .                                                   |
| <b>MPIMotionAttrMaskMASTER_START</b> | This attribute specifies 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 <a href="#">MPIMotionAttributes</a> structure. See <a href="#">Starting at Specific Master Positions</a> . |

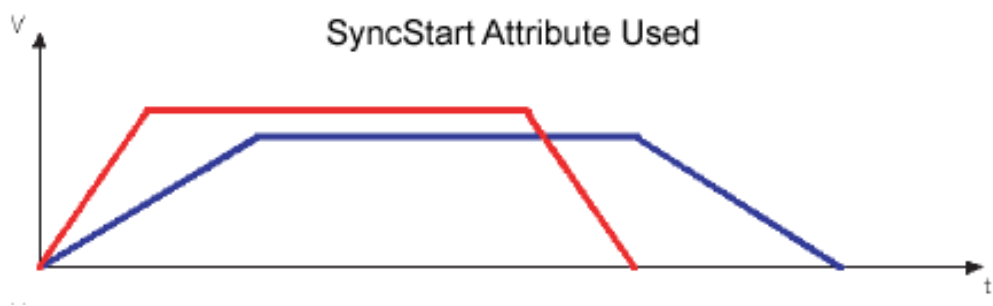
|                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionAttrMaskNO_BACKTRACK</b>      | 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 <a href="#">Reversal of the Master - Backtracking</a> . |
| <b>MPIMotionAttrMaskNO_BACKTRACK_HOLD</b> | 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 <a href="#">Reversal of the Master with NO_BACKTRACK_HOLD</a> .                                                                                             |

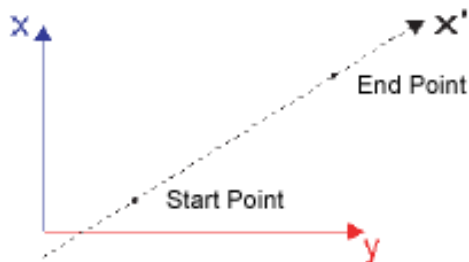
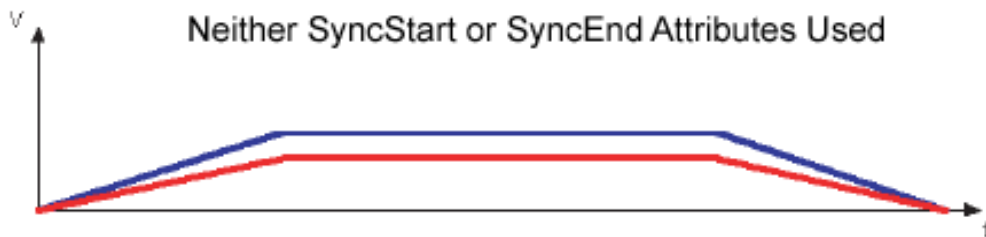
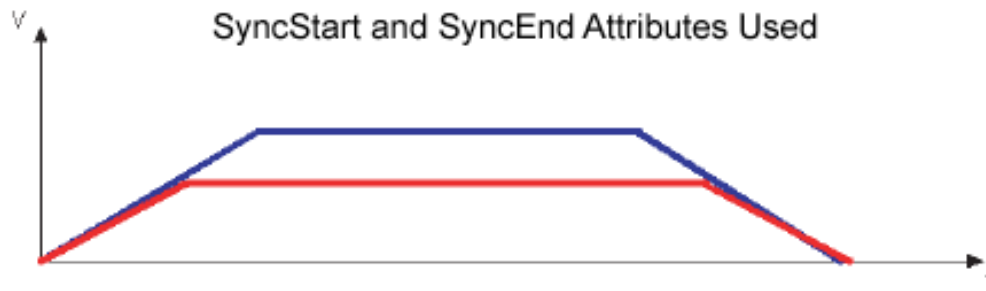
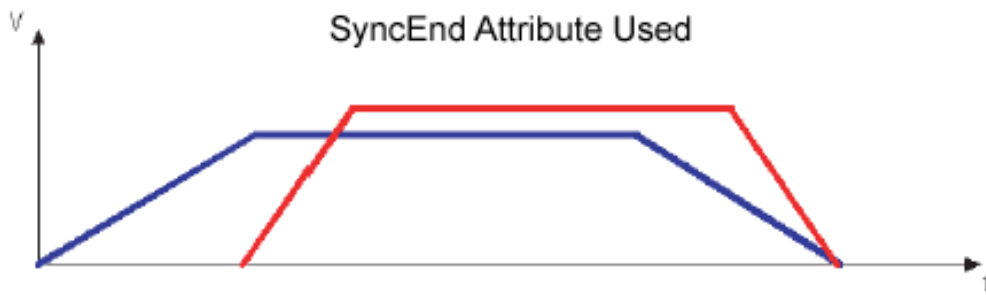
Remarks

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

|                                      |                                                                                                    |
|--------------------------------------|----------------------------------------------------------------------------------------------------|
| <b>MEIMotionAttrMaskEVENT</b>        | This mask allows the user to specify an MPIEventMask during a motion.                              |
| <b>MEIMotionAttrMaskFINAL_VEL</b>    | This mask allows the user to specify a non-zero target velocity for point to point motion types.   |
| <b>MEIMotionAttrMaskNO_REVERSAL</b>  | This mask prevents a motion profile from changing direction.                                       |
| <b>MEIMotionAttrMaskHOLD</b>         | This mask prevents a motion profile from executing until the specified trigger conditions are met. |
| <b>MEIMotionAttrMaskHOLD_LESS</b>    | Motion attribute mask for less than or equal hold logic.                                           |
| <b>MEIMotionAttrMaskHOLD_GREATER</b> | Motion attribute mask for greater than or equal hold logic.                                        |
| <b>MEIMotionAttrMaskOUTPUT</b>       | This mask allows the user to set or clear bits during a motion.                                    |
| <b>MEIMotionAttrOUTPUT</b>           | This bit allows the user to set or clear bits during a motion.                                     |

## See Also

[mpiMotionStart](#) | [mpiMotionModify](#) | [MPIMotionType](#) | [MPITrajectory](#) | [MPIEventMask](#)

## Description

|                                      |                                                                                                    |
|--------------------------------------|----------------------------------------------------------------------------------------------------|
| <b>MEIMotionAttrMaskEVENT</b>        | This mask allows the user to specify an MPIEventMask during a motion.                              |
| <b>MEIMotionAttrMaskFINAL_VEL</b>    | This mask allows the user to specify a non-zero target velocity for point to point motion types.   |
| <b>MEIMotionAttrMaskNO_REVERSAL</b>  | This mask prevents a motion profile from changing direction.                                       |
| <b>MEIMotionAttrMaskHOLD</b>         | This mask prevents a motion profile from executing until the specified trigger conditions are met. |
| <b>MEIMotionAttrMaskHOLD_LESS</b>    | Motion attribute mask for less than or equal hold logic.                                           |
| <b>MEIMotionAttrMaskHOLD_GREATER</b> | Motion attribute mask for greater than or equal hold logic.                                        |
| <b>MEIMotionAttrMaskOUTPUT</b>       | This mask allows the user to set or clear bits during a motion.                                    |
| <b>MEIMotionAttrOUTPUT</b>           | This bit allows the user to set or clear bits during a motion.                                     |

## See Also

[mpiMotionStart](#) | [mpiMotionModify](#) | [MPIMotionType](#) | [MPITrajectory](#) | [MPIEventMask](#)

# 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

|                   |                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------|
| <b>type</b>       | This value specifies the output type to determine the output bits to be set or cleared.     |
| <b>*output</b>    | This value specifies the memory address when MEIMotionAttrOutputTypeOUTPUT is used.         |
| <b>motor</b>      | This value specifies the motor number when MEIMotionAttrOutputTypeMOTOR is used.            |
| <b>offMask</b>    | This value specifies the bits to be turned OFF when MEIMotionAttrOutputTypeOFFMASK is used. |
| <b>onMask</b>     | This value specifies the bits to be turned ON when MEIMotionAttrOutputTypeONMASK is used.   |
| <b>pointIndex</b> | 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

|                                      |                                                                                     |
|--------------------------------------|-------------------------------------------------------------------------------------|
| <b>MEIMotionAttrOutputTypeNONE</b>   | This type disables the setting/clearing of output bit(s) during motion.             |
| <b>MEIMotionAttrOutputTypeMOTOR</b>  | This type configures a motor's output bit(s) to be set or cleared during motion.    |
| <b>MEIMotionAttrOutputTypeOUTPUT</b> | 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 */
    long      masterStart;     /* MPIMotionAttrMaskMASTER_START */
    long      repeatFrom;      /* MPIMotionAttrMaskREPEAT */
} MPIMotionAttributes;
```

## Description

|             |                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| delay       | This array defines the delay time (seconds) before a motion profile begins execution.                                                                                                                                                            |
| id          | This value defines the identity for a point to point motion. 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 <a href="#">Starting at Specific Master Positions</a> .       |
| 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 <a href="#">Repeating Cams</a> . |

## 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



|                       |                                                                                                                      |
|-----------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>eventMask</b>      | This structure specifies the mask to enable event generation. See <a href="#">MPIEventMask</a> for more information. |
| <b>*finalVelocity</b> | This array specifies the target velocity for each axis when MEIMotionAttrMaskFINAL_VEL is used.                      |
| <b>*hold</b>          | This array specifies the hold configurations for each axis when MEIMotionAttrMaskHOLD is used.                       |
| <b>*outputCount</b>   | This array specifies the number of points per axis, to set/clear an output when MEIMotionAttrMaskOUTPUT is used.     |
| <b>*output</b>        | 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

|                      |                                                                                                                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>positionCount</b> | This value specifies the number of points.                                                                                                                                                                                                                   |
| <b>*position</b>     | 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. |
| <b>*time</b>         | 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. |
| <b>point</b>         | This structure contains the point configuration. Please see <a href="#">MPIMotionPoint</a> data type for more information.                                                                                                                                   |

## See Also

[MPIMotionPoint](#)

# MPIMotionBSPLINE

## Definition

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

## Description

|                   |                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b> | This value specifies the number of points.                                                                                                                                                                                                                   |
| <b>*position</b>  | 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. |
| <b>*time</b>      | 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. |
| <b>point</b>      | This structure contains the point configuration. Please see MPIMotionPoint data type for more information.                                                                                                                                                   |

## See Also

[MPIMotionPoint](#)

# MPIMotionCam

## Definition

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

## Description

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

|                        |                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b>      | This field defines the type of master position source is being used.                                                                                                                                                                                                                                                                                                      |
| <b>**slavePosition</b> | This field defines the distance that each slave axis will move during each segment of the cam.                                                                                                                                                                                                                                                                            |
| <b>*masterDistance</b> | An array containing pointCount number of elements. Each element is the distance the master axis will move during each segment of the cam.                                                                                                                                                                                                                                 |
| <b>**gearRatio</b>     | 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

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

## Definition: MEIMotionConfig

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

## Description

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

## See Also

[MPIMotionDecelTime](#) | [mpiMotionModify](#)

# MPIMotionDecelTime

## Definition

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

## Description

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

## See Also

# MEIMotionFrame

## Definition

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

## Description

|                   |                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b> | The value specifies the number of frames.                                                                   |
| <b>*frame</b>     | This structure contains the frame data for each frame. See MEIXmpFrame for more information.                |
| <b>point</b>      | This structure specifies the points configuration. See <a href="#">MPIMotionPoint</a> 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,
    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 03.02.00

## 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.

### **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 [Caming](#)) 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,
} MEIMotionMessage;
```

**Required Header:** `stdmei.h`

**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.

## 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      besse;
    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.

|             |                                                                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pt</b>   | This structure contains the parameters for a PT motion type. Please see <a href="#">MPIMotionPT</a> data type for more information.                                                |
| <b>ptf</b>  | This structure contains the parameters for a PTF (position, time, and feedforward) motion type. Please see <a href="#">MPIMotionPTF</a> data type for more information.            |
| <b>pvt</b>  | This structure contains the parameters for a PVT motion type. Please see <a href="#">MPIMotionPVT</a> data type for more information.                                              |
| <b>pvtf</b> | This structure contains the parameters for a PTF (position, velocity, time, and feedforward) motion type. Please see <a href="#">MPIMotionPVTF</a> data type for more information. |

|                     |                                                                                                                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spline</b>       | This structure contains the parameters for a SPLINE motion type. Please see <a href="#">MPIMotionSPLINE</a> data type for more information.                                                                        |
| <b>bessel</b>       | This structure contains the parameters for a BESSEL motion type. Please see <a href="#">MPIMotionBESSEL</a> data type for more information.                                                                        |
| <b>bspline</b>      | This structure contains the parameters for a BSPLINE motion type. Please see <a href="#">MPIMotionBSPLINE</a> data type for more information.                                                                      |
| <b>sCurve</b>       | This structure contains the parameters for a S_CURVE motion type. Please see <a href="#">MPIMotionSCurve</a> data type for more information.                                                                       |
| <b>sCurveJerk</b>   | This structure contains the parameters for an S-Curve Jerk point to point motion type. Please see <a href="#">MPIMotionSCurveJerk</a> data type for more information.                                              |
| <b>trapezoidal</b>  | This structure contains the parameters for a Trapezoidal point to point motion type. Please see <a href="#">MPIMotionTrapezoidal</a> data type for more information.                                               |
| <b>velocity</b>     | This structure contains the parameters for a VELOCITY motion type. Please see <a href="#">MPIMotionVelocity</a> data type for more information.                                                                    |
| <b>velocityJerk</b> | This structure contains the parameters for a VELOCITY_JERK motion type. Please see <a href="#">MPIMotionVelocity</a> data type for more information.                                                               |
| <b>cam</b>          | This structure contains the parameters for a cam motion type. Please see <a href="#">MPIMotionCAM</a> data type for more information.                                                                              |
| <b>attributes</b>   | This structure contains the parameters for motion attributes. Please see <a href="#">MPIMotionAttributes</a> data type for more information.                                                                       |
| <b>*external</b>    | This points to an external structure, containing controller specific parameters. Presently, this only supports MEIMotionAttributes. Please see <a href="#">MEIMotionAttributes</a> data type for more information. |

## Definition: MEIMotionParams

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

## Description



|                      |                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>frame</b>         | This structure contains the frame data and points configuration. See <a href="#">MEIMotionFrame</a> for more information. |
| <b>attributes</b>    | This structure contains the motion attributes data. See <a href="#">MPIMotionAttributes</a> for more information.         |
| <b>attributesMEI</b> | This structure contains the motion attributes data. See <a href="#">MEIMotionAttributes</a> for more information.         |

## See Also

[MEIMotionFrame](#) | [MPIMotionAttributes](#) | [MEIMotionAttributes](#)

For more information, please see the [PT and PVT Path Motion for the XMP](#) application note.

# MPIMotionPoint

## Definition

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

## Description

|                   |                                                                                                                                                                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>retain</b>     | 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. |
| <b>final</b>      | 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.                                              |
| <b>emptyCount</b> | This value specifies the minimum number of points in the buffer. If the number of points in the buffer is below emptyCount, an E_STOP action will occur. When emptyCount is (-1), the buffer low trigger is disabled.                                       |

## See Also

[mpiMotionModify](#) | [MPIMotionAttrMaskAPPEND](#) | [MPIMotionPT](#)

# MPIMotionPT

## Definition

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

## Description

|                   |                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b> | This value specifies the number of points.                                                                                                                                                                                                                   |
| <b>position</b>   | 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. |
| <b>time</b>       | 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. |
| <b>point</b>      | This structure contains the point configuration. Please see <a href="#">MPIMotionPoint</a> data type for more information.                                                                                                                                   |

## See Also

[MPIMotionPoint](#) | [Streaming Point Motion Type Calculations](#)

For more information, please see the [PT and PVT Path Motion for the XMP](#) application note.

# 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 feedforward 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.).

|                     |                                                                                                                                                                                                                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b>   | This value specifies the number of points.                                                                                                                                                                                                                                                                                               |
| <b>*position</b>    | 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.                                                                             |
| <b>*feedforward</b> | 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). |
| <b>*time</b>        | 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

|                   |                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b> | This value specifies the number of points.                                                                                                                                                                                                                   |
| <b>position</b>   | 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. |
| <b>velocity</b>   | This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.     |
| <b>time</b>       | 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. |
| <b>point</b>      | This structure contains the point configuration. Please see <a href="#">MPIMotionPoint</a> data type for more information.                                                                                                                                   |

## See Also

[MPIMotionPoint](#) | [Streaming Point Motion Type Calculations](#)

For more information, please see the [PT and PVT Path Motion for the XMP](#) application note.

# 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 feedforward 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.).

|                     |                                                                                                                                                                                                                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b>   | This value specifies the number of points.                                                                                                                                                                                                                                                                                               |
| <b>*position</b>    | 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.                                                                             |
| <b>*velocity</b>    | This array stores the times for the motion profile. There is one time value per point. The time specifies the number of seconds between the specified position, and the next position (point). The length of the time array must be equal to pointCount.                                                                                 |
| <b>*feedforward</b> | 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). |

|              |                                                                                                                                                                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>*time</b> | 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. |
| <b>point</b> | This structure contains the point configuration. Please see <a href="#">MPIMotionPoint</a> 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.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>*trajectory</b> | 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. |
| <b>*position</b>   | 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

|                   |                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pointCount</b> | This value specifies the number of points.                                                                                                                                                                                                                   |
| <b>*position</b>  | 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. |
| <b>*time</b>      | 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. |
| <b>point</b>      | This structure contains the point configuration. Please see <a href="#">MPIMotionPoint</a> data type for more information.                                                                                                                                   |

## See Also

[MPIMotionPoint](#)

# MEIMotionTrace

## Definition

```
typedef enum {  
    MEIMotionTraceSTATUS,  
    MEIMotionTracePARAMS,  
} MEIMotionTrace;
```

## Description

|                             |                                                                       |
|-----------------------------|-----------------------------------------------------------------------|
| <b>MEIMotionTraceSTATUS</b> | This trace bit enables motion status tracing for mpiMotion calls.     |
| <b>MEIMotionTracePARAMS</b> | 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.

For 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.

|                         |                                                                                                                                                                                               |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionTypePT</b>  | This type fits constant velocity profile segments through a list of position and time points. Not supported in motion sequences. Please see <a href="#">MPIMotionPT</a> for more information. |
| <b>MPIMotionTypePTF</b> | PTF motion is identical to PT except host-calculated feedforward values can be specified for each PTF motion segment.                                                                         |
| <b>MPIMotionTypePVT</b> | This type fits jerk profile segments through a list of position, velocity and time points. Not supported in motion sequences. Please see <a href="#">MPIMotionPVT</a> for more information.   |

|                                   |                                                                                                                                                                                                                                                                                  |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotionTypePVTF</b>          | PVTF motion is identical to PVT except host-calculated feedforward values can be specified for each PVTF motion segment.                                                                                                                                                         |
| <b>MPIMotionTypeSPLINE</b>        | This type fits a Cubic spline through a specified list of position and time points. Please see <a href="#">MPIMotionSPLINE</a> data type for more information.                                                                                                                   |
| <b>MPIMotionTypeBESSEL</b>        | This type fits a Bessel spline through a specified list of position and time points. Please see <a href="#">MPIMotionBESSEL</a> data type for more information.                                                                                                                  |
| <b>MPIMotionTypeBSPLINE</b>       | This type fits a 3rd order B spline through a list of position and time points. Please see <a href="#">MPIMotionBSPLINE</a> data type for more information.                                                                                                                      |
| <b>MPIMotionTypeBSPLINE2</b>      | This type fits a 2nd order B spline through a list of position and time points. Please see <a href="#">MPIMotionBSPLINE</a> data type for more information.                                                                                                                      |
| <b>MPIMotionTypeS_CURVE</b>       | 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 <a href="#">MPIMotionSCurve</a> data type for more information.                        |
| <b>MPIMotionTypeS_CURVE_JERK</b>  | 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 <a href="#">MPIMotionSCurve</a> data type for more information. |
| <b>MPIMotionTypeTRAPEZOIDAL</b>   | 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 <a href="#">MPIMotionTrapezoidal</a> data type for more information.           |
| <b>MPIMotionTypeVELOCITY</b>      | This type specifies S-Curve acceleration to a constant velocity. The profile trajectory is specified by acceleration, velocity, and jerkPercent. Please see <a href="#">MPIMotionVelocity</a> data type for more information.                                                    |
| <b>MPIMotionTypeVELOCITY_JERK</b> | This type specifies S-Curve acceleration to a constant velocity. The profile trajectory is specified by acceleration, velocity, accelerationJerk and decelerationJerk. Please see <a href="#">MPIMotionVelocity</a> data type for more information.                              |
| <b>MPIMotionTypeCAM_LINEAR</b>    | This type generates linear interpolated cam motion. See <a href="#">Camming</a> .                                                                                                                                                                                                |



**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 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.

# 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                                | 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        |
| Adc                                   | MPIAdcMessageFIRST             |
| Adc: ADC invalid                      | MPIAdcMessageADC_INVALID       |
| 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                      | MPICanMessageCAN_INVALID       |
| Can: firmware invalid                 | MPICanMessageFIRMWARE_INVALID  |
| Can: firmware version mismatch        | MPICanMessageFIRMWARE_VERSION  |

|                                                                                    |                                          |
|------------------------------------------------------------------------------------|------------------------------------------|
| Can: firmware not initialized                                                      | MPICanMessageNOT_INITIALIZED             |
| Can: IO not supported by node                                                      | MPICanMessageIO_NOT_SUPPORTED            |
| Can: file format incorrect                                                         | MPICanMessageFILE_FORMAT_ERROR           |
| Can: user aborted operation                                                        | MPICanMessageUSER_ABORT                  |
| Can: DPR Command protocol                                                          | MPICanMessageCOMMAND_PROTOCOL            |
| Can: interface not found                                                           | MPICanMessageINTERFACE_NOT_FOUND         |
| Can: node dead                                                                     | MPICanMessageNODE_DEAD                   |
| Can: SDO Timeout                                                                   | MPICanMessageSDO_TIMEOUT                 |
| Can: SDO abort                                                                     | MPICanMessageSDO_ABORT                   |
| Can: SDO protocol                                                                  | MPICanMessageSDO_PROTOCOL                |
| Can: transmit buffer overflow                                                      | MPICanMessageTX_OVERFLOW                 |
| Can: RTR transmit buffer overflow                                                  | MPICanMessageRTR_TX_OVERFLOW             |
| Can: Receive buffer empty                                                          | MPICanMessageRX_BUFFER_EMPTY             |
| Can: Bus off                                                                       | MPICanMessageBUS_OFF                     |
| Can: Signature invalid                                                             | MPICanMessageSIGNATURE_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 |
| 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.<br>adjust<br>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: 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: 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 file                 | MEIDriveMapMessageNODE_NOT_FOUND_IN_MAP    |
| driveMap: Drive firmware version not found in drives map file    | 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_BLENDED         |
| 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   |
| 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                                                                                   | MEIFlashMessageELEMENT_NOT_FOUND      |
| List: element invalid                                                                                     | MEIFlashMessageELEMENT_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: 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: 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              |
| Motor                                                                               | MPIMotorMessageFIRST                        |
| Motor: motor invalid                                                                | MPIMotorMessageMOTOR_INVALID                |
| Motor: motor type invalid                                                           | MPIMotorMessageMOTOR_TYPE_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 ( $10^{-7} < \text{pulseWidth} < 10^{-3}$ ) | 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      |
| 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: Config file and network are different                   | MEISqNodeMessageCONFIG_NETWORK_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: 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 error                                        | MEISqNodeMessageIO_SLICE_INITIALIZATION_ERROR       |
| SqNode: I/O Slice initalization timeout                                      | MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT     |
| SqNode: I/O Slice topology mismatch                                          | MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH          |
| 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 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 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          |
| WinNT                                                                                            | MEIPlatformMessageFIRST                       |
| WinNT: platform invalid                                                                          | MEIPlatformMessagePLATFORM_INVALID            |
| WinNT: device invalid                                                                            | MEIPlatformMessageDEVICE_INVALID              |
| WinNT: device error                                                                              | MEIPlatformMessageDEVICE_ERROR                |
| WinNT: device map error                                                                          | MEIPlatformMessageDEVICE_MAP_ERROR            |

# 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.

For information about using absolute encoders with the MPI [click here](#).

## Methods

### Create, Delete, Validate Methods

|                                         |                       |
|-----------------------------------------|-----------------------|
| <a href="#"><u>mpiMotorCreate</u></a>   | Create Motor object   |
| <a href="#"><u>mpiMotorDelete</u></a>   | Delete Motor object   |
| <a href="#"><u>mpiMotorValidate</u></a> | Validate Motor object |

### Configuration and Information Methods

|                                                   |                                                |
|---------------------------------------------------|------------------------------------------------|
| <a href="#"><u>mpiMotorAmpEnableGet</u></a>       | Get state of amp enable output                 |
| <a href="#"><u>mpiMotorAmpEnableSet</u></a>       | Set state of amp enable output                 |
| <a href="#"><u>meiMotorAmpFault</u></a>           | Writes the amp fault buffer data of a motor.   |
| <a href="#"><u>meiMotorAmpFaultClear</u></a>      | Clears the amp fault buffer data of a motor.   |
| <a href="#"><u>meiMotorAmpWarning</u></a>         | Writes the amp warning buffer of the motor.    |
| <a href="#"><u>meiMotorAmpWarningClear</u></a>    | Clears the motor's amp warning message buffer. |
| <a href="#"><u>mpiMotorAxisMapGet</u></a>         | Get object map of axes                         |
| <a href="#"><u>meiMotorCommutationModeGet</u></a> | Gets the commutation mode of a motor.          |
| <a href="#"><u>meiMotorCommutationModeSet</u></a> | Sets the commutation mode of a motor.          |
| <a href="#"><u>mpiMotorConfigGet</u></a>          | Get motor configuration                        |
| <a href="#"><u>mpiMotorConfigSet</u></a>          | Set motor configuration                        |
| <a href="#"><u>meiMotorConfigStepper</u></a>      | Configures a motor for stepper mode.           |
| <a href="#"><u>meiMotorDacConfigGet</u></a>       | Get a Motor's (motor) Dac configuration        |
| <a href="#"><u>meiMotorDacConfigSet</u></a>       | Set a Motor's (motor) Dac configuration        |
| <a href="#"><u>meiMotorDacFlashConfigGet</u></a>  |                                                |
| <a href="#"><u>meiMotorDacFlashConfigSet</u></a>  |                                                |
| <a href="#"><u>mpiMotorDedicatedIn</u></a>        |                                                |

[mpiMotorDedicatedOutGet](#)

[mpiMotorFeedback](#)

Get feedback position

[mpiMotorFlashConfigGet](#)

Get flash config of motor

[mpiMotorFlashConfigSet](#)

Set flash config of motor

[mpiMotorGeneralIn](#)

[mpiMotorGeneralOutGet](#)

[mpiMotorGeneralOutSet](#)

[meiMotorInfo](#)

Get information about the network, node, and drive interface

[mpiMotorStatus](#) / [meiMotorStatus](#)

Get motor status

## Event Methods

[mpiMotorEventConfigGet](#)

Get motor's event configuration

[mpiMotorEventConfigSet](#)

Set motor's event configuration

[mpiMotorEventNotifyGet](#)

Get motor's event mask for host notification.

[mpiMotorEventNotifySet](#)

Set motor's event mask for host notification.

[mpiMotorEventReset](#)

Reset events specified in event mask

## Memory Methods

[mpiMotorMemory](#)

Get address of motor memory

[mpiMotorMemoryGet](#)

Copy motor memory to application memory

[mpiMotorMemorySet](#)

Copy application memory to motor memory

## Action Methods

[meiMotorEncoderReset](#)

Clears encoder faults.

[meiMotorMultiTurnReset](#)

Clears the SynqNet drive multi-turn data for absolute type encoders.

## Relational Methods

[mpiMotorControl](#)

Get handle to associated Control object

[mpiMotorFilterMapGet](#)

Get object map of associated Filters

[mpiMotorFilterMapSet](#)

Set the Filters using object map

[mpiMotorNumber](#)

Get index number of motor (in Control list)

## Other Methods

[meiMotorCompareListGet](#)

[meiMotorEncoderRatio](#)

Get encoder ratio from the XMP.

## Data Types

[MEIMotorAmpFaults](#)

[MEIMotorAmpFaultMsg](#)  
[MEIMotorAmpWarnings](#)  
[MEIMotorAmpWarningMsg](#)  
[MPIMotorBrake](#)  
[MPIMotorBrakeMode](#)  
[MPIMotorConfig](#) / [MEIMotorConfig](#)  
[MEIMotorDacConfig](#)  
[MEIMotorDacChannelConfig](#)  
[MEIMotorDacChannelStatus](#)  
[MEIMotorDacStatus](#)  
[MPIMotorDedicatedIn](#)  
[MPIMotorDedicatedOut](#)  
[MEIMotorDisableAction](#)  
[MPIMotorEncoder](#) / [MEIMotorEncoder](#)  
[MPIMotorEncoderFault](#)  
[MPIMotorEncoderFaultMask](#)  
[MEIMotorEncoderRatio](#)  
[MEIMotorEncoderReverseModulo](#)  
[MEIMotorEncoderType](#)  
[MPIMotorEventConfig](#) / [MEIMotorEventConfig](#)  
[MPIMotorEventTrigger](#)  
[MEIMotorFaultBit](#)  
[MEIMotorFaultConfig](#)  
[MEIMotorFaultMask](#)  
[MPIMotorFeedback](#)  
[MPIMotorGeneralIo](#)  
[MEIMotorInfo](#)  
[MEIMotorInfoDedicatedIn](#)  
[MEIMotorInfoDedicatedOut](#)  
[MEIMotorInfoGeneralIo](#)  
[MEIMotorInfoNodeType](#)  
[MEIMotorIoConfig](#)  
[MEIMotorIoConfigIndex](#)  
[MEIMotorIoType](#)  
[MEIMotorIoTypeMask](#)  
[MPIMotorMessage](#) / [MEIMotorMessage](#)  
[MEIMotorStatus](#)  
[MEIMotorStatusOutput](#)  
[MEIMotorStepper](#)  
[MEIMotorStepperPulse](#)  
[MEIMotorStepperPulseType](#)



[MEIMotorStepperStatus](#)

[MPIMotorType](#)

## Macros

[mpiMotorEncoderFaultMaskBIT](#)

## Constants

[MEIMotorAmpFaultsMAX](#)

[MEIMotorAmpMsgMAX](#)

[MEIMotorAmpWarningsMAX](#)

# 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

|                      |                                          |
|----------------------|------------------------------------------|
| <b>handle</b>        | to a Motor object                        |
| <b>MPIHandleVOID</b> | 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**

if *MotorDelete* successfully deletes a Motor object and invalidates its handle

## See Also

[mpiMotorCreate](#) | [mpiMotorValidate](#)

# mpiMotorValidate

## Declaration

```
long mpiMotorValidate(MPIMotor motor)
```

**Required Header:** stdmpi.h

## Description

**mpiMotorValidate** validates a Motor object and its handle (*motor*).

|              |                              |
|--------------|------------------------------|
| <b>motor</b> | a handle to the Motor object |
|--------------|------------------------------|

### Return Values

|                     |                                         |
|---------------------|-----------------------------------------|
| <b>MPIMessageOK</b> | if Motor is a handle to a valid object. |
|---------------------|-----------------------------------------|

## See Also

[mpiMotorCreate](#) | [mpiMotorDelete](#)

# mpiMotorAmpEnableGet

## Declaration

```
long mpiMotorAmpEnableGet(MPIMotor motor,
                           long *ampEnable)
```

Required Header: stdmpi.h

## 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 " <i>ampEnable</i> " is | Then                |
|----------------------------|---------------------|
| FALSE (0)                  | the amp is disabled |
| TRUE (1)                   | the amp is enabled  |

| Return Values |                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>MotorAmpEnableGet</i> successfully writes the Motor's amp enable output state to the location |

## See Also

[MPIMotorConfig](#) | [mpiMotorAmpEnableSet](#)

# mpiMotorAmpEnableSet

## Declaration

```
long mpiMotorAmpEnableSet(MPIMotor motor,
                           long ampEnable)
```

Required Header: stdmpi.h

## 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.

| <i>If "ampEnable" is</i> | <i>Then</i>              |
|--------------------------|--------------------------|
| FALSE (0)                | the amp will be disabled |
| TRUE (1)                 | the amp will be enabled  |

| Return Values       |                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAmpEnableSet</i> successfully sets the Motor's amp enable output state to ampEnable |

## 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.

|               |                                                                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b>  | a handle to the Motor object                                                                                                                  |
| <b>*fault</b> | a pointer to a structure containing the number of amp faults, their coded values and message strings. See <a href="#">MEIMotorAmpFaults</a> . |

| Return Values       |                                                          |
|---------------------|----------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAmpFault</i> writes the amp fault buffer data |

## 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.

|              |                              |
|--------------|------------------------------|
| <b>motor</b> | a handle to the Motor object |
|--------------|------------------------------|

| Return Values       |                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAmpFaultClear</i> successfully flushes the Motor's amp fault message buffer. |

## 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.

|                 |                                                                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object                                                                                                                      |
| <b>*warning</b> | a pointer to a structure containing the number of amp warnings, their coded values and message strings. See <a href="#">MEIMotorAmpWarnings</a> . |

| Return Values       |                                                                           |
|---------------------|---------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAmpWarning</i> successfully writes the amp warning buffer data |

## 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.

|              |                              |
|--------------|------------------------------|
| <b>motor</b> | a handle to the Motor object |
|--------------|------------------------------|

| Return Values       |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAmpWarningClear</i> successfully flushes the motor's amp warning message buffer. |

## 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**.

|                 |                                                                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object                                                                                                                             |
| <b>*axismap</b> | 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       |                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAxisMapGet</i> successfully writes the Motor's object map of Axes to the structure |

## 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**

if *MotorCommutationModeGet* successfully gets the commutation mode of a Motor and writes it to the location

## 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**

if *MotorCommutationModeSet* successfully sets the commutation mode of a Motor to ***mode***

## 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

|                     |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorConfigGet</i> successfully writes the Motor's configuration to the structure(s) |
|---------------------|--------------------------------------------------------------------------------------------|

## 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

if *MotorConfigSet* successfully sets the Motor's configuration using data from the structure(s)

## 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.

|                      |                                                                              |
|----------------------|------------------------------------------------------------------------------|
| <b>motor</b>         | a handle to the Motion object                                                |
| <b>*config</b>       | a pointer to the motion frame buffer status structure returned by the method |
| <b>stepperNumber</b> | index to a step engine                                                       |

| Return Values       |                                                                                                                  |
|---------------------|------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorConfigStepper</i> successfully modifies the motor configuration structure pointed to by <i>config</i> |

## 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***.

|                |                                             |
|----------------|---------------------------------------------|
| <b>motor</b>   | a handle to the Motor object.               |
| <b>*config</b> | a pointer to a MEIMotorDacConfig structure. |

| Return Values       |                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorDacConfigGet</i> successfully writes the Motor's Dac configuration to the <i><b>config</b></i> structure. |

## 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*.

|                |                                             |
|----------------|---------------------------------------------|
| <b>motor</b>   | a handle to the Motor object.               |
| <b>*config</b> | a pointer to a MEIMotorDacConfig structure. |

| Return Values       |                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorDacConfigSet</i> successfully writes the Motor's Dac configuration to the controller. |

## 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*.

|                |                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b>   | a handle to the Motor object                                                                                                                             |
| <b>*flash</b>  | <i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |
| <b>*config</b> | a pointer to a MEIMotorDacConfig structure.                                                                                                              |

| Return Values       |                                                                                                                  |
|---------------------|------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorDacFlashConfigGet</i> successfully writes the DAC's configuration from flash memory to the structure. |

## 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*.

|                |                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b>   | a handle to the Motor object                                                                                                                             |
| <b>*flash</b>  | <i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |
| <b>*config</b> | a pointer to a MEIMotorDacConfig structure.                                                                                                              |

| Return Values       |                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorDacFlashConfigSet</i> successfully writes the DAC's configuration to flash memory using data from the <i>config</i> structure. |

## 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

## Description

**mpiMotorDedicatedIn** function reads the current state of one or more dedicated input bits.

|                 |                                                                        |
|-----------------|------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object                                           |
| <b>startBit</b> | the first dedicated in bit that will be returned by the function.      |
| <b>bitCount</b> | the number of dedicated in bits that will be returned by the function. |
| <b>*state</b>   | the address of the current state of the inputs that is returned.       |

| Return Values       |                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorDedicatedIn</i> successfully reads the current state of one or more dedicated input bits. |

## See Also

[Dedicated Motor I/O](#)

# 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

**mpiMotorDedicatedOut** function changes the state of one or more dedicated output bits.

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object                                            |
| <b>startBit</b> | the first dedicated out bit that will be returned by the function.      |
| <b>bitCount</b> | the number of dedicated out bits that will be returned by the function. |
| <b>*state</b>   | the address of the current state of the inputs that is returned.        |

| Return Values       |                                                                                                     |
|---------------------|-----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorDedicatedOutGet</i> successfully changes the state of one or more dedicated output bits. |

## See Also

[Dedicated Motor I/O](#)

# 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

|                     |                                                                                      |
|---------------------|--------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorFeedback</i> successfully writes the feedback position into the location. |
|---------------------|--------------------------------------------------------------------------------------|

## See Also

[mpiMotorFeedbackConfigSet](#)

# 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.

### Return Values

#### MPIMessageOK

if *MotorFlashConfigGet* successfully writes the Motor's flash configuration to the structure(s). ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

## 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.

| Return Values                                |                                                                                                                                                                                                                                                                           |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                          | if <i>MotorFlashConfigSet</i> successfully sets the Motor's flash configuration using data from the structure(s) <i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |
| <b>MEIFlashMessageNETWORK_TOPOLOGY_ERROR</b> | returned because the SynqNet topology has not yet been saved to flash using <i>meiSynqNetFlashTopologySave()</i> .                                                                                                                                                        |

## 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

## Description

**mpiMotorGeneralIn** reads the current input state of one or more general purpose bits.

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object                                              |
| <b>startBit</b> | the first general purpose bit that will be returned by the function.      |
| <b>bitCount</b> | the number of general purpose bits that will be returned by the function. |
| <b>*state</b>   | the address of the current state of the inputs that is returned.          |

| Return Values       |                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorGeneralIn</i> successfully reads the current input state of one or more general purpose bits. |

## 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

## Description

**mpiMotorGeneralOutGet** function reads the current output state of one or more general purpose bits.

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object                                                   |
| <b>startBit</b> | the first general purpose bit that will be returned by the function.           |
| <b>bitCount</b> | the number of general purpose bits that will be returned by the function.      |
| <b>*state</b>   | the address of the current state of the general purpose bits will be returned. |

| Return Values       |                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorGeneralOutGet</i> successfully reads the current output state of one or more general purpose output bits. |

## See Also

[General Purpose Motor I/O](#)

# mpiMotorGeneralOutSet

## Declaration

```
long mpiMotorGeneralOutSet(MPIMotor motor,
                           long startBit,
                           long bitCount,
                           unsigned long state,
                           long wait);
```

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

## Description

**mpiMotorGeneralOutSet** function changes the state of one or more general purpose bits.

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <b>motor</b>    | a handle to the Motor object.                                             |
| <b>startBit</b> | the first general purpose bit that will be returned by the function.      |
| <b>bitCount</b> | the number of general purpose bits that will be returned by the function. |
| <b>state</b>    | the new state of the general purpose bits that will be returned.          |
| <b>wait</b>     | See <a href="#">Motor Digital Output Waits</a> .                          |

| Return Values       |                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorGeneralOutSet</i> successfully changes the state of one or more general purpose bits. |

## 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*.

|              |                                            |
|--------------|--------------------------------------------|
| <b>motor</b> | a handle to the Motor object               |
| <b>*info</b> | a pointer to a motor information structure |

| Return Values       |                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorInfo</i> successfully writes the network and node/drive interface information into the Motor structure. |

## See Also

[meiMotorStatus](#)

# 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.

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| <b>motor</b>     | a handle to the Motor object                                   |
| <b>*status</b>   | a pointer to the motor status structure returned by the method |
| <b>*external</b> | a pointer to an implementation-specific structure              |

### Return Values

|                              |                                                                       |
|------------------------------|-----------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>MotorStatus</i> successfully gets the status of a Motor object. |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL.                                 |

## 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.

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <b>control</b>     | a handle to the Control object                                 |
| <b>motorNumber</b> | index to the motor                                             |
| <b>*status</b>     | a pointer to the motor status structure returned by the method |
| <b>*external</b>   | pointer to an implementation-specific structure                |

| Return Values                |                                                                       |
|------------------------------|-----------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>MotorStatus</i> successfully gets the status of a Motor object. |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> 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 *eventConfig,
                           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.

## 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);
}
```

#### Return Values

**MPIMessageOK**

if *MotorEventConfigGet* successfully writes the Motor's configuration for the event to the structure(s)

## 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  | if <i>MotorEventConfigGet</i> successfully writes the Motor's configuration for the event to the structure(s) |

## 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       |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorEventNotifyGet</i> successfully writes the event mask to the location(s) |

## 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 <i>eventmask</i> to MPIEventMaskALL  |
| disable host notification of all events | set <i>eventmask</i> to MPIEventTypeNONE |

## 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);
}

```

### Return Values

#### MPIMessageOK

if *MotorEventNotifySet* successfully requests host notification of the event(s) that are specified by ***eventMask*** and generated by ***motor***

## 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**

if *MotorEventReset* successfully resets the event(s) that are specified in ***eventMask*** and generated by ***motor***

## See Also

# 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**

if *MotorMemory* successfully writes the Motor's address to the contents of *memory*.

## See Also

[mpiMotorMemoryGet](#) | [mpiMotorMemorySet](#)



# mpiMotorMemoryGet

## Declaration

```
long mpiMotorMemoryGet(MPIMotor motor,  
                        void      *dst,  
                        void      *src,  
                        long      count)
```

**Required Header:** stdmpi.h

## 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**

if *MotorMemoryGet* successfully copies data from Motor memory to application memory

## See Also

[mpiMotorMemorySet](#) | [mpiMotorMemory](#)

# mpiMotorMemorySet

## Declaration

```
long mpiMotorMemorySet(MPIMotor motor,  
                        void      *dst,  
                        void      *src,  
                        long      count)
```

**Required Header:** stdmpi.h

## 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**

if *MotorMemorySet* successfully copies data from application memory to Motor memory

## 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

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorEncoderReset</i> successfully resets the motor's encoder value. |
|---------------------|----------------------------------------------------------------------------|

## 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.

|              |                              |
|--------------|------------------------------|
| <b>motor</b> | a handle to the Motor object |
|--------------|------------------------------|

| Return Values       |                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorMultiTurnReset</i> successfully clears the SynqNet drive multi-turn data for absolute type encoders. |

## See Also

# 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.

|              |                              |
|--------------|------------------------------|
| <b>motor</b> | a handle to the Motor object |
|--------------|------------------------------|

| Return Values        |                            |
|----------------------|----------------------------|
| <b>MPIControl</b>    | handle to a Control object |
| <b>MPIHandleVOID</b> | 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

|                     |                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorFilterMapGet</i> successfully writes the Filter's object map (associated with a Motor) to the structure |
|---------------------|--------------------------------------------------------------------------------------------------------------------|

## 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

|                     |                                                                                          |
|---------------------|------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorFilterMapSet</i> successfully sets the Filters using data from the object map |
|---------------------|------------------------------------------------------------------------------------------|

## 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**

if *MotorNumber* successfully writes the Motor's index to the contents of ***number***

## See Also



# meiMotorCompareListGet

## Declaration

```
long meiMotorCompareListGet(MPIMotor motor,  
                           long      *compareCount,  
                           long      *compareList);
```

**Required Header:** stdmei.h

## Description

**meiMotorCompareListGet** sets ***compareCount*** to the number of compares that ***motor*** has. It sets ***compareList*** to the number of the compare object for each compare.

### Return Values

**MPIMessageOK**

if *motor* is a valid MPIMotor object.

## 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                |                                                                               |
|------------------------------|-------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>meiMotorEncoderRatio</i> successfully gets encoder ration from the XMP. |
| <b>MPIMessageARG_INVALID</b> | if arguments are not valid                                                    |

## 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

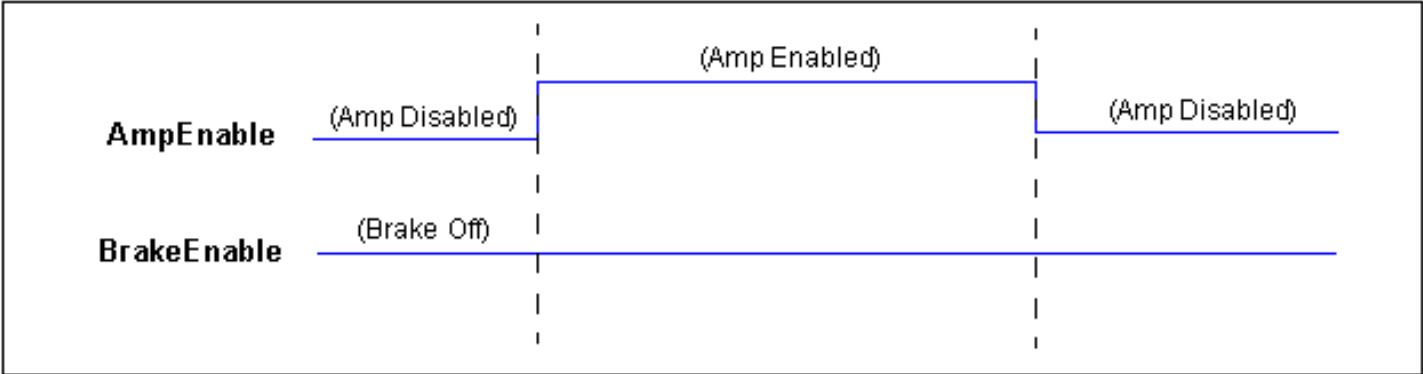
```
typedef struct MPIMotorBrake {
    MPIMotorBrakeMode    mode;
    float                applyDelay;
    float                releaseDelay;
} MPIMotorBrake;
```

## Description

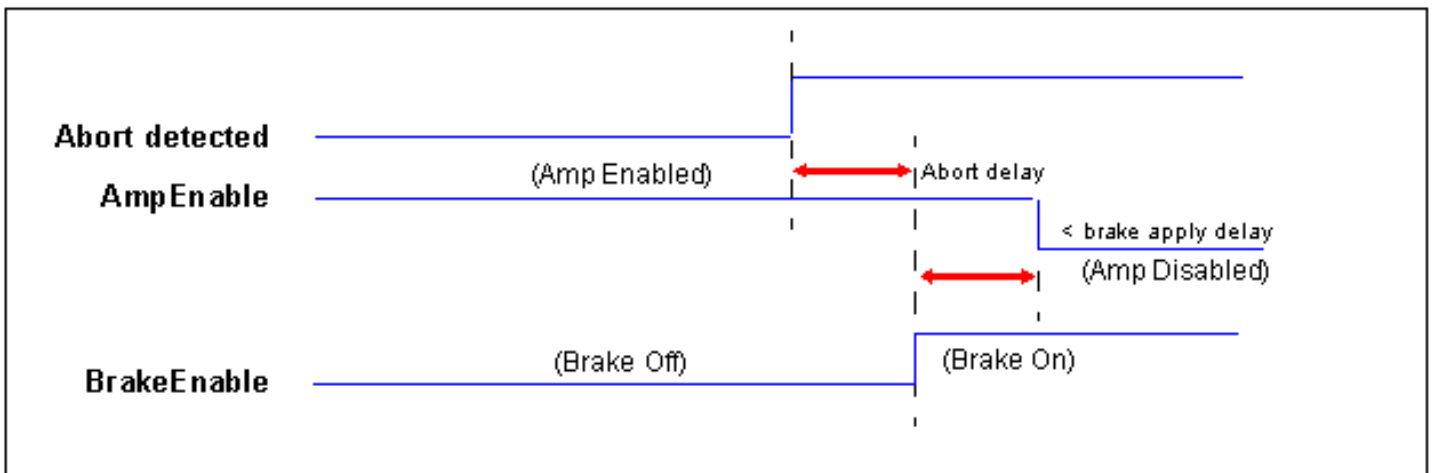
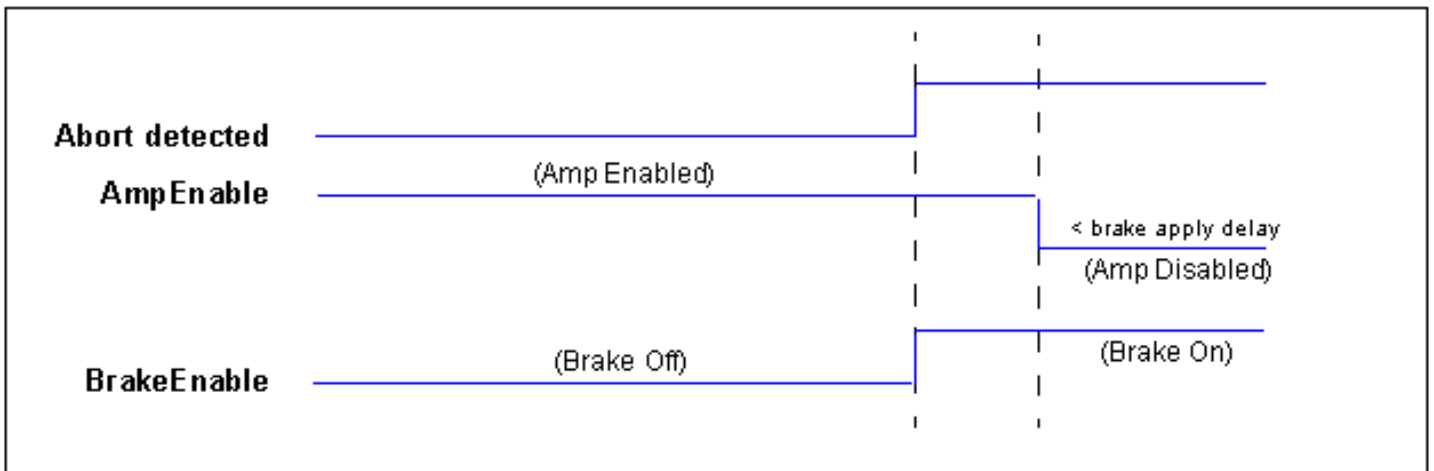
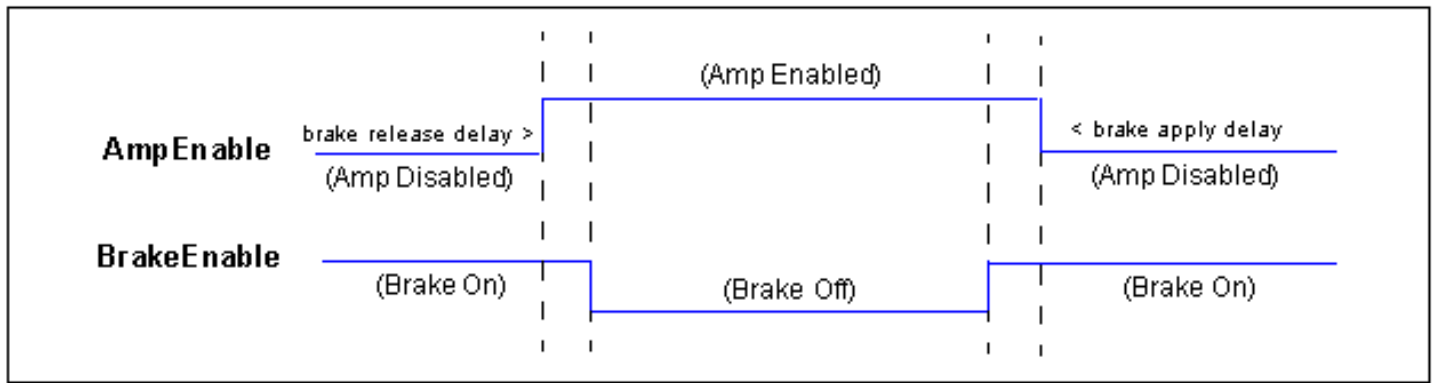
**MPIMotorBrake** specifies the configuration for a motor's dedicated brake logic. Each motor object has a dedicated brake output. The controller enables/disables the brake depending on the amp enable state and the brake configuration. When the amp enable is disabled, the brake is set to an active state. When the amp enable is enabled, the brake is set to an inactive state.

|              |                                                                                                      |
|--------------|------------------------------------------------------------------------------------------------------|
| mode         | An enumerated brake mode. See <a href="#">MPIMotorBrakeMode</a> .                                    |
| 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



## 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.

|                               |                                                  |
|-------------------------------|--------------------------------------------------|
| <b>MPIMotorBrakeModeNONE</b>  | Brake feature is disabled.                       |
| <b>MPIMotorBrakeModeDELAY</b> | 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 <a href="#">MPIMotorEventConfig</a> description.                                                                                                    |
| abortDelay  | Sets time value, in seconds, to delay Abort action after Event has occurred.                                                                                                                         |
| enableDelay | Sets time value, in seconds, to delay Enabling of the amplifier after commanded.                                                                                                                     |
| brake       | Configures the dedicated brake logic. See <a href="#">MPIMotorBrake</a> .                                                                                                                            |
| 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 <a href="#">MPIObjectMap</a> description in Object section. |

## Definition: MEIMotorConfig

```

typedef struct MEIMotorConfig {
    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;

```

## Description

|                          |                                                                                                                                                                                             |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Encoder</b>           | Structure to configure Motor Encoder type and parameters                                                                                                                                    |
| <b>StatusOutput</b>      | A structure to configure a motor's digital outputs to monitor axis status bits. Requires custom firmware.                                                                                   |
| <b>Io</b>                | An array of motor I/O configuration structures.                                                                                                                                             |
| <b>faultConfig</b>       | A structure to configure a motor's fault bits. Support for motor fault bits is node/drive specific.                                                                                         |
| <b>Stepper</b>           | Structure to configure Motor Stepper parameters. See <a href="#">MEIMotorStepper</a> description.                                                                                           |
| <b>Dac</b>               | Structure that includes Command and Auxiliary DAC configuration for each motor. See <a href="#">MEIMotorDacConfig</a> description.                                                          |
| <b>Commutation</b>       | A structure to configure controller sinusoidal commutation. This structure is controller specific. Please see the <a href="#">Sinusoidal Commutation Application Note</a> for more details. |
| <b>limit</b>             | Structure used to configure custom motor limits and events.                                                                                                                                 |
| <b>nodeFailureAction</b> | Action applied to the motor when a Node failure occurs.                                                                                                                                     |
| <b>userFaultAction</b>   | 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.

|              |                                                                                             |
|--------------|---------------------------------------------------------------------------------------------|
| <b>level</b> | <i>level</i> 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          = MEIXmpMotorDedicatedFlagsMaskAMP_FAULT, /*
bit 0 */
    MPIMotorDedicatedInBRAKE_APPLIED      = MEIXmpMotorDedicatedFlagsMaskBRAKE_ON, /*
bit 1 */
    MPIMotorDedicatedInHOME                = MEIXmpMotorDedicatedFlagsMaskHOME, /*
bit 2 */
    MPIMotorDedicatedInLIMIT_HW_POS        = MEIXmpMotorDedicatedFlagsMaskPOS_LIMIT, /*
bit 3 */
    MPIMotorDedicatedInLIMIT_HW_NEG        = MEIXmpMotorDedicatedFlagsMaskNEG_LIMIT, /*
bit 4 */
    MPIMotorDedicatedInINDEX_PRIMARY       = MEIXmpMotorDedicatedFlagsMaskENC_INDEX0, /*
bit 5 */
    MPIMotorDedicatedInFEEDBACK_FAULT      = MEIXmpMotorDedicatedFlagsMaskENC_FAULT, /*
bit 6 */
    MPIMotorDedicatedInCAPTURED            = MEIXmpMotorDedicatedFlagsMaskCAPTURED, /*
bit 7 */
    MPIMotorDedicatedInHALL_A              = MEIXmpMotorDedicatedFlagsMaskHALL_A, /*
bit 8 */
    MPIMotorDedicatedInHALL_B              = MEIXmpMotorDedicatedFlagsMaskHALL_B, /*
bit 9 */
    MPIMotorDedicatedInHALL_C              = MEIXmpMotorDedicatedFlagsMaskHALL_C, /*
bit 10 */
    MPIMotorDedicatedInAMP_ACTIVE          = MEIXmpMotorDedicatedFlagsMaskAMP_ACTIVE, /*
bit 11 */
    MPIMotorDedicatedInINDEX_SECONDARY     = MEIXmpMotorDedicatedFlagsMaskENC_INDEX1, /*
bit 12 */
    MPIMotorDedicatedInWARNING             = MEIXmpMotorDedicatedFlagsMaskWARNING, /*
bit 13 */
    MPIMotorDedicatedInDRIVE_STATUS_9      = MEIXmpMotorDedicatedFlagsMaskDRIVE_STATUS9, /*
bit 14 */
    MPIMotorDedicatedInDRIVE_STATUS_10     = MEIXmpMotorDedicatedFlagsMaskDRIVE_STATUS10, /*
bit 15 */
} MPIMotorDedicatedIn;
```

**Change History:** Added in the 03.02.00

## 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.

|                                           |                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMotorDedicatedInAMP_FAULT</b>       | Generated by the masked motor fault bits. Active when one or more masked motor faults bits are active. See <a href="#">MEIMotorFaultConfig</a> .                                                                                                                                                                  |
| <b>MPIMotorDedicatedInBRAKE_APPLIED</b>   | Mechanical brake state.                                                                                                                                                                                                                                                                                           |
| <b>MPIMotorDedicatedInHOME</b>            | Position calibration sensor.                                                                                                                                                                                                                                                                                      |
| <b>MPIMotorDedicatedInLIMIT_HW_POS</b>    | Hardware limit for the positive direction.                                                                                                                                                                                                                                                                        |
| <b>MPIMotorDedicatedInLIMIT_HW_NEG</b>    | Hardware limit for the negative direction.                                                                                                                                                                                                                                                                        |
| <b>MPIMotorDedicatedInINDEX_PRIMARY</b>   | Primary encoder index input signal.                                                                                                                                                                                                                                                                               |
| <b>MPIMotorDedicatedInFEEDBACK_FAULT</b>  | Position feedback status. TRUE when position feedback fails, FALSE when operating properly.                                                                                                                                                                                                                       |
| <b>MPIMotorDedicatedInCAPTURED</b>        | Currently not supported.                                                                                                                                                                                                                                                                                          |
| <b>MPIMotorDedicatedInHALL_A</b>          | Reflects the state of Hall Sensor A                                                                                                                                                                                                                                                                               |
| <b>MPIMotorDedicatedInHALL_B</b>          | Reflects the state of Hall Sensor B                                                                                                                                                                                                                                                                               |
| <b>MPIMotorDedicatedInHALL_C</b>          | Reflects the state of Hall Sensor C                                                                                                                                                                                                                                                                               |
| <b>MPIMotorDedicatedInAMP_ACTIVE</b>      | <p>A bit set by the drive that indicates the amplifier's state.</p> <p>1 = Amplifier is closing the current loop and the motor winding are energized.</p> <p>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.</p> |
| <b>MPIMotorDedicatedInINDEX_SECONDARY</b> | Secondary encoder index input signal.                                                                                                                                                                                                                                                                             |
| <b>MPIMotorDedicatedInWARNING</b>         | <p>Drive warning state.</p> <p>1 = drive warning status bit is active and warning message is available</p> <p>0 = drive warning status bit is not active. Support for this bit varies depending on the drive type.</p>                                                                                            |
| <b>MPIMotorDedicatedInDRIVE_STATUS_9</b>  | State of bit 9 in the SynqNet drive specific status register.                                                                                                                                                                                                                                                     |
| <b>MPIMotorDedicatedInDRIVE_STATUS_10</b> | State of bit 10 in the SynqNet drive specific status register.                                                                                                                                                                                                                                                    |

## See Also

[MPIMotorDedicatedOut](#)

# MPIMotorDedicatedOut

## Definition

```
typedef enum {
    MPIMotorDedicatedOutAMP_ENABLE      = MEIXmpMotorDedicatedFlagsMaskAMP_ENABLE,
    /* bit 0 */
    MPIMotorDedicatedOutBRAKE_RELEASE  = MEIXmpMotorDedicatedFlagsMaskBRAKE_RELEASE,
    /* bit 1 */
} MEIMotorDedicatedOut;
```

**Change History:** Added in the 03.02.00

## 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.

|                                          |                                                                                                         |
|------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>MPIMotorDedicatedOutAMP_ENABLE</b>    | Enable/disable drive or amplifier. Drive is enabled when TRUE, disabled when FALSE.                     |
| <b>MPIMotorDedicatedOutBRAKE_RELEASE</b> | Enable/disable mechanical brake. Brake is released (motor shaft is free) when TRUE, engaged when FALSE. |

## See Also

[MPIMotorDedicatedIn](#)

# 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.

|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIMotorDisableActionNONE</b>       | No action. When the Amp Enable is disabled (or enabled), the controller continues to calculate and apply the torque demand output value.                                                                                                                                                                                                                                                                                             |
| <b>MEIMotorDisableActionCMD_EQ_ACT</b> | <div>Command position equals actual position action (default). When the Amp Enable is disabled, the controller will:</div> <div><div>1) Disable the servo loop output (except for the offset).</div><div>2) Set the command position equal to the actual position every sample.</div><div>3) Clear the integrator error.</div></div> <div>When the Amp Enable is enabled, the controller will operate the servo loop normally.</div> |

## 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.

|                                 |                                          |
|---------------------------------|------------------------------------------|
| <b>MPIMotorEncoderPRIMARY</b>   | The first encoder feedback for a motor.  |
| <b>MPIMotorEncoderSECONDARY</b> | 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 */
    long                   countsPerRev;
    MEIMotorEncoderRatio    ratio;
    MEIMotorEncoderReverseModulo reverseModulo;
} MEIMotorEncoder;
```

## Description

**MEIMotorEncoder** is an enumeration of encoder feedback inputs for a motor.

|                                   |                                                                                                  |
|-----------------------------------|--------------------------------------------------------------------------------------------------|
| <b>type</b> <sup>†</sup>          | The type of feedback being used by the motor.                                                    |
| <b>encoderPhase</b> <sup>†</sup>  | Controls the direction of the encoder counts (only applicable with quadrature encoder feedback). |
| <b>filterDisable</b> <sup>†</sup> | Enables/Disables the use of filters on the encoder signal.                                       |
| <b>countsPerRev</b>               | The number of encoder counts per one revolution of the motor's shaft.                            |
| <b>ratio</b>                      | <b>Custom firmware required.</b> Encoder feedback ratio for scaling.                             |
| <b>reverseModulo</b>              | <b>Custom firmware required.</b> Reverse modulo value for the encoder to handle rollover.        |

<sup>†</sup> - 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](#)

# 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.

|                                                 |                                                                                               |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <b>MPIMotorEncoderFaultPRIMARY</b>              | Sets the Motor Event (Encoder Fault) to trigger from the primary encoder.                     |
| <b>MPIMotorEncoderFaultSECONDARY</b>            | Sets the Motor Event (Encoder Fault) to trigger from the secondary encoder.                   |
| <b>MPIMotorEncoderFaultPRIMARY_OR_SECONDARY</b> | 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.

|                                        |                                  |
|----------------------------------------|----------------------------------|
| <b>MPIMotorEncoderFaultMaskBW_DET</b>  | Mask for Broken Wire detection   |
| <b>MPIMotorEncoderFaultMaskILL_DET</b> | Mask for Illegal State detection |
| <b>MPIMotorEncoderFaultMaskABS_ERR</b> | Mask for Absolute Encoder Error  |

## 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).  $B/A = \text{ratio}$ .

**NOTE:** Custom Firmware is required to support this data type.

|          |                  |
|----------|------------------|
| <b>A</b> | the denominator. |
| <b>B</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.

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <b>*Ptr</b>     | a pointer to the XMP address.                                        |
| <b>Rollover</b> | the value is the resolution of the incoming data pointed to by *Ptr. |

## See Also

# MEIMotorEncoderType

## Definition

```
typedef enum MEIMotorEncoderType{
    MEIMotorEncoderTypeQUAD_AB,
    MEIMotorEncoderTypeDRIVE,
    MEIMotorEncoderTypeSSI,
} MEIMotorEncoderType;
```

## 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.

|                                   |                                                                                           |
|-----------------------------------|-------------------------------------------------------------------------------------------|
| <b>MEIMotorEncoderTypeQUAD_AB</b> | Quadrature encoder feedback.                                                              |
| <b>MEIMotorEncoderTypeDRIVE</b>   | Drive memory interface feedback. This includes all feedback types supported by the drive. |
| <b>MEIMotorEncoderTypeSSI</b>     | Synchronous Serial Interface (SSI) absolute encoder feedback.                             |

## See Also

# MPIMotorEventConfig / MEIMotorEventConfig

## Definition: MPIMotorEventConfig

```
typedef struct MPIMotorEventConfig {
    MPIAction          action;
    MPIMotorEventTrigger trigger;
    long              direction; /* boolean */
    float             duration;  /* seconds */
} MPIMotorEventConfig;
```

## Description

**MPIMotorEventConfig** is a structure used to configure Motor Events.

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>direction</b> | Only used for Hardware and Software limits. Setting <b><i>direction</i></b> to TRUE requires the direction of motion to be in direction of the Limit to trigger the event. If <b><i>direction</i></b> is set to FALSE, a limit event can be generated regardless of the direction of motion.       |
| <b>duration</b>  | time that Limit (e.g. Home, Pos. and Neg. Limits, User Limit) must be asserted before Event is generated. Value in seconds.<br><br><b>NOTE:</b> 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.

|                  |                                                                                                                                                                                                                                                                                                                                                       |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>condition</b> | 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 in <a href="#">App Note 215</a> .                                                                                                  |
| <b>status</b>    | 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 <b>Status</b> values are listed in the <a href="#">Values of Status</a> table below.                                                                |
| <b>logic</b>     | an enum that sets the logic applied between the two condition block outputs, Condition[0] and Condition[1]. Valid <b>Logic</b> values are listed in the <a href="#">Values of Logic</a> table below.                                                                                                                                                  |
| <b>output</b>    | 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 in <a href="#">App Note 215</a> . |
| <b>count</b>     | <b>For internal use only.</b> The MPI method, mpiMotorEventConfigSet(...) will not write these values.                                                                                                                                                                                                                                                |
| <b>state</b>     | <b>For internal use only.</b> 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... |
|--------------------------------|-------------------------------|-------------------------------------------------------|
| <b>MEIXmpLogicNEVER</b>        | Nothing                       | No event is generated                                 |
| <b>MEIXmpLogicSINGLE</b>       | Condition[0]                  | Condition[0] == TRUE                                  |
| <b>MEIXmpLogicOR</b>           | Condition[0],<br>Condition[1] | (Condition[0]    Condition[1]) == TRUE                |
| <b>MEIXmpLogicAND</b>          | Condition[0],<br>Condition[1] | (Condition[0] && Condition[1]) == TRUE                |
| other <b>MEIXmpLogic</b> enums | For internal use only.        |                                                       |

## See Also

[MPIMotorEventConfig and Motor Limit Configuration](#) |  
 See [App Note 215](#) for a more in-depth breakdown of this structure. |  
[mpiMotorEventConfigGet](#) | [mpiMotorEventConfigSet](#) | [MPIEncoderFault](#) |  
[Error Limit and Limit Switch Errors](#)

# MPIMotorEventTrigger

## Definition

```
typedef union {
    long          polarity; /* 0 => active low, else active high */
    long          position; /* MPIEventTypeLIMIT_SW_[POS|NEG] */
    float         error;    /* MPIEventTypeLIMIT_ERROR */
    MPIMotorEncoderFault encoder; /* MPIEventTypeENCODER_FAULT */
} MPIMotorEventTrigger;
```

## Description

|          |                                                                                                                                                                                                                                                                                                                                                                                           |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| polarity | Configures the polarity for the motor event trigger.<br>If polarity = 0 (FALSE), the event will trigger on an active low signal.<br>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.<br>If the positive limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_POS event.<br>If the negative limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_NEG event. |
| 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 auxiliary 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.

| <b>faultMask†</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.<br><br>Available on remote motion blocks.                                                                                                                                                                                                                    |
| MEIMotorFaultMaskDRIVE    | Bit in the drive interface status register. Indicates a fault condition in the drive.<br><br>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.<br><br>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.<br><br>NOT available on remote motion blocks.                                                                                                    |

|                                                  |                                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIMotorFaultMaskFEEDBACK</b>                 | <p>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.</p> <p>Available on remote motion blocks.</p>                                                                         |
| <b>MEIMotorFaultMaskAMP_NOT_POWERED</b>          | <p>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.</p> <p>NOT available on remote motion blocks.</p>                                                        |
| <b>MEIMotorFaultMaskDRIVE_NOT_READY</b>          | <p>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.</p> <p>NOT available on remote motion blocks.</p> |
| <b>MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY</b> | <p>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 auxiliary position feedback system has failed.</p> <p>Available on remote motion blocks.</p>                                                     |

## 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,  
} MPIMotorGeneralIo;
```

**Change History:** Added in the 03.02.00

## Description

**MPIMotorGeneralIo** enumeration gives labels for each of the general purpose outputs that a motor can support.

|                           |                    |
|---------------------------|--------------------|
| <b>MPIMotorGeneralIo0</b> | General I/O bit 0. |
| <b>MPIMotorGeneralIo1</b> | General I/O bit 1. |
| <b>MPIMotorGeneralIo2</b> | General I/O bit 2. |
| <b>MPIMotorGeneralIo3</b> | General I/O bit 3. |
| <b>MPIMotorGeneralIo4</b> | General I/O bit 4. |
| <b>MPIMotorGeneralIo5</b> | General I/O bit 5. |
| <b>MPIMotorGeneralIo6</b> | General I/O bit 6. |
| <b>MPIMotorGeneralIo7</b> | General I/O bit 7. |
| <b>MPIMotorGeneralIo8</b> | General I/O bit 8. |

|                            |                     |
|----------------------------|---------------------|
| <b>MPIMotorGeneralIo9</b>  | General I/O bit 9.  |
| <b>MPIMotorGeneralIo10</b> | General I/O bit 10. |
| <b>MPIMotorGeneralIo11</b> | General I/O bit 11. |
| <b>MPIMotorGeneralIo12</b> | General I/O bit 12. |
| <b>MPIMotorGeneralIo13</b> | General I/O bit 13. |
| <b>MPIMotorGeneralIo14</b> | General I/O bit 14. |
| <b>MPIMotorGeneralIo15</b> | General I/O bit 15. |

## 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.

|                      |                                                                         |
|----------------------|-------------------------------------------------------------------------|
| <b>nodeType</b>      | Identifies the type of node. See <a href="#">MEIMotorInfoNodeType</a> . |
| <b>networkNumber</b> | An index to the SynqNet network (0, 1, 2 etc.).                         |
| <b>nodeNumber</b>    | An index to the node (0, 1, 2, etc.).                                   |
| <b>driveIndex</b>    | An index to the drive interface (0, 1, 2, etc.).                        |
| <b>captureCount</b>  | Counts the number of captures for the motor.                            |
| <b>encoderCount</b>  | Counts the number of encoders for the motor.                            |
| <b>probeCount</b>    | Counts the number of hardware Probe engines for the motor.              |
| <b>dedicatedIn</b>   | Details about the dedicated inputs.                                     |
| <b>dedicatedOut</b>  | Details about the dedicated outputs.                                    |
| <b>generalIo</b>     | Details about the general purpose I/O.                                  |

## See Also

[meiMotorInfo](#)

# MEIMotorInfoDedicatedIn

## Definition

```
typedef struct MEIMotorInfoDedicatedIn {  
    long supported;  
} MEIMotorInfoDedicatedIn;
```

**Change History:** Added in the 03.02.00

## Description

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <b>supported</b> | 0 = This dedicated input is NOT supported by the hardware. |
|                  | 1 = This dedicated input is supported by the hardware.     |

## See Also

# MEIMotorInfoDedicatedOut

## Definition

```
typedef struct MEIMotorInfoDedicatedOut {
    long    supported;
} MEIMotorInfoDedicatedOut;
```

**Change History:** Added in the 03.02.00

## Description

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| <b>supported</b> | 0 = This dedicated output is NOT supported by the hardware. |
|                  | 1 = This dedicated output is supported by the hardware.     |

## 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.

|             |                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Type</b> | The I/O bit function. See <a href="#">MEIMotorIoType</a> . 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,  
} MEIMotorIoConfigIndex;
```

## Description

**MEIMotorIoConfigIndex** is an enumeration of configurable motor I/O, indexed by a number.

|                               |                    |
|-------------------------------|--------------------|
| <b>MEIMotorIoConfigIndex0</b> | motor I/O number 0 |
| <b>MEIMotorIoConfigIndex1</b> | motor I/O number 1 |
| <b>MEIMotorIoConfigIndex2</b> | motor I/O number 2 |
| <b>MEIMotorIoConfigIndex3</b> | motor I/O number 3 |
| <b>MEIMotorIoConfigIndex4</b> | motor I/O number 4 |
| <b>MEIMotorIoConfigIndex5</b> | motor I/O number 5 |
| <b>MEIMotorIoConfigIndex6</b> | motor I/O number 6 |
| <b>MEIMotorIoConfigIndex7</b> | motor I/O number 7 |
| <b>MEIMotorIoConfigIndex8</b> | motor I/O number 8 |
| <b>MEIMotorIoConfigIndex9</b> | motor I/O number 9 |

|                                |                     |
|--------------------------------|---------------------|
| <b>MEIMotorIoConfigIndex10</b> | motor I/O number 10 |
| <b>MEIMotorIoConfigIndex11</b> | motor I/O number 11 |
| <b>MEIMotorIoConfigIndex12</b> | motor I/O number 12 |
| <b>MEIMotorIoConfigIndex13</b> | motor I/O number 13 |
| <b>MEIMotorIoConfigIndex14</b> | motor I/O number 14 |
| <b>MEIMotorIoConfigIndex15</b> | motor I/O number 15 |

## See Also

[MEIMotorIoConfig](#)



# MEIMotorIoType

## Definition

```
typedef enum MEIMotorIoType {
    MEIMotorIoTypeOUTPUT,

    MEIMotorIoTypePULSE_A,
    MEIMotorIoTypePULSE_B,
    MEIMotorIoTypeCOMPARE_0,
    MEIMotorIoTypeCOMPARE_1,
    MEIMotorIoTypeBRAKE,
    MEIMotorIoTypeSOURCE6,
    MEIMotorIoTypeSSI_CLOCK0,
    MEIMotorIoTypeSSI_CLOCK1,

    MEIMotorIoTypeSOURCE9,
    MEIMotorIoTypeSOURCE10,
    MEIMotorIoTypeSOURCE11,
    MEIMotorIoTypeSOURCE12,
    MEIMotorIoTypeSOURCE13,
    MEIMotorIoTypeSOURCE14,
    MEIMotorIoTypeSOURCE15,

    MEIMotorIoTypeINPUT,

    MEIMotorIoTypeLAST,
    MEIMotorIoTypeFIRST = MEIMotorIoTypeINVALID+1,

    MEIMotorIoTypeSOURCE1 = MEIMotorIoTypePULSE_A,
    MEIMotorIoTypeSOURCE2 = MEIMotorIoTypePULSE_B,
    MEIMotorIoTypeSOURCE3 = MEIMotorIoTypeCOMPARE_0,
    MEIMotorIoTypeSOURCE4 = MEIMotorIoTypeCOMPARE_1,
    MEIMotorIoTypeSOURCE5 = MEIMotorIoTypeBRAKE,
    MEIMotorIoTypeSOURCE7 = MEIMotorIoTypeSSI_CLOCK0,
    MEIMotorIoTypeSOURCE8 = MEIMotorIoTypeSSI_CLOCK1,
} MEIMotorIoType;
```

**Change History:** 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.

|                               |                                                       |
|-------------------------------|-------------------------------------------------------|
| <b>MEIMotorIoTypeOUTPUT</b>   | discrete digital output                               |
| <b>MEIMotorIoTypeSOURCE1</b>  | motor I/O source number 1 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE2</b>  | motor I/O source number 2 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE3</b>  | motor I/O source number 3 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE4</b>  | motor I/O source number 4 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE5</b>  | motor I/O source number 5 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE6</b>  | motor I/O source number 6 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE7</b>  | motor I/O source number 7 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE8</b>  | motor I/O source number 8 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE9</b>  | motor I/O source number 9 for node-specific feature.  |
| <b>MEIMotorIoTypeSOURCE10</b> | motor I/O source number 10 for node-specific feature. |
| <b>MEIMotorIoTypeSOURCE11</b> | motor I/O source number 11 for node-specific feature. |
| <b>MEIMotorIoTypeSOURCE12</b> | motor I/O source number 12 for node-specific feature. |
| <b>MEIMotorIoTypeSOURCE13</b> | motor I/O source number 13 for node-specific feature. |
| <b>MEIMotorIoTypeSOURCE14</b> | motor I/O source number 14 for node-specific feature. |
| <b>MEIMotorIoTypeSOURCE15</b> | motor I/O source number 15 for node-specific feature. |
| <b>MEIMotorIoTypeINPUT</b>    | 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.

|                                     |                                        |
|-------------------------------------|----------------------------------------|
| <b>MEIMotorIoTypeMaskOUTPUT</b>     | Supports the output configuration.     |
| <b>MEIMotorIoTypeMaskPULSE_A</b>    | Supports the Pulse A configuration.    |
| <b>MEIMotorIoTypeMaskPULSE_B</b>    | Supports the Pulse B configuration.    |
| <b>MEIMotorIoTypeMaskCOMPARE_0</b>  | Currently not supported.               |
| <b>MEIMotorIoTypeMaskCOMPARE_1</b>  | Currently not supported.               |
| <b>MEIMotorIoTypeMaskBRAKE</b>      | See <a href="#">MPIMotorBrake</a>      |
| <b>MEIMotorIoTypeMaskSSI_CLOCK0</b> | Supports the SSI_CLOCK0 configuration. |
| <b>MEIMotorIoTypeMaskSSI_CLOCK1</b> | Supports the SSI_CLOCK1 configuration. |
| <b>MEIMotorIoTypeMaskINPUT</b>      | Supports the input configuration.      |

## See Also

[MPIMotorBrake](#) | [MEIMotorInfo](#) | [mpiMotorConfigGet](#) | [mpiMotorConfigSet](#)

# MPIMotorMessage / MEIMotorMessage

## Definition: MPIMotorMessage

```
typedef enum {
    MPIMotorMessageMOTOR_INVALID,
    MPIMotorMessageTYPE_INVALID,
} MPIMotorMessage;
```

**Required Header:** stdmpi.h

**Change History:** 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.

## 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**

[mpiMotorCreate](#)

# MEIMotorStatus

## Definition

```
typedef struct MEIMotorStatus {  
    MEIMotorDacStatus      dac;  
    MEIMotorFaultMask     faultMask;  
    MEIMotorStepperStatus stepper;  
} MEIMotorStatus;
```

## Description

**MEIMotorStatus** returns the specific Motor Status registers of the controller.

|                  |                                                                                                                                                                                     |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>dac</b>       | The status for both Command and Auxiliary DACs.                                                                                                                                     |
| <b>faultMask</b> | The motor fault bit masks for the SynqNet node drive interface.                                                                                                                     |
| <b>stepper</b>   | 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 {
    long                loopback;    /* TRUE = FPGA pulse feedback,
                                         FALSE = encoder feedback */

    MEIMotorStepperPulse pulseA;
    MEIMotorStepperPulse pulseB;
    double              pulseWidth; /* output pulse width (sec),
                                         10-7 < pulseWidth < 10-3 */
} MEIMotorStepper;
```

## 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 requirements 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.

|                                |                                                                                                                                                                                       |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>loopback</b> <sup>†</sup>   | 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. |
| <b>pulseA</b>                  | Structure used to configure Step Pulse A.                                                                                                                                             |
| <b>pulseB</b>                  | Structure used to configure Step Pulse B.                                                                                                                                             |
| <b>pulseWidth</b> <sup>†</sup> | sets width of Step pulse. Valid values range from a minimum width of 0.1 usec to maximum width of 25.5 usec.                                                                          |

<sup>†</sup> - 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;  
    long                        invert;  
} MEIMotorStepperPulse;
```

## 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 requirements 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.

|               |                                                                                                                                                                            |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>   | 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.                           |
| <b>invert</b> | 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.

|                                      |                                                                                                                                                                                                                                               |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIMotorStepperPulseTypeSTEP</b>  | 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.                                                                           |
| <b>MEIMotorStepperPulseTypeDIR</b>   | 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.                                                                |
| <b>MEIMotorStepperPulseTypeCW</b>    | 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.                                                                     |
| <b>MEIMotorStepperPulseTypeCCW</b>   | 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.                                                            |
| <b>MEIMotorStepperPulseTypeQUADA</b> | 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. |

|                                      |                                                                                                                                                                                                                                               |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIMotorStepperPulseTypeQUADB</b> | 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 {  
    long    pulseLockLost; /* TRUE if Pulse jitter logic has lost lock,  
                             otherwise FALSE */  
    long    pulseStatus;    /* TRUE if a Pulse was generated incorrectly,  
                             otherwise FALSE */  
} MEIMotorStepperStatus;
```

## Description

**MEIMotorStepperStatus** is a structure used to check for error conditions relating to the pulse module.

|                      |                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pulseLockLost</b> | Indicates that the pulse module lost timing lock with the SynqNet network.                                                                                                       |
| <b>pulseStatus</b>   | 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.

|                            |                             |
|----------------------------|-----------------------------|
| <b>MPIMotorTypeSERVO</b>   | Motor configured as Servo   |
| <b>MPIMotorTypeSTEPPER</b> | 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



## Working with Absolute Encoders

Currently, only absolute motor encoders of Yaskawa SGDM Sigma Series II Servopacks are supported by the MPI. Custom firmware is needed in order to use these absolute encoders. Contact MEI for more information.

The encoders are automatically interrogated at power-up or after a controller reset. Axis Origin and Command Positions are set to correctly reflect the absolute position of the motor with no position error initially. The absolute position is the position within a single revolution.

The encoder interrogation is controlled by a SEN signal (to the drive) which must be connected to a configured XMP Transceiver or User Out signal. There are no restrictions as to which XMP signal is used except that current drive limitations may limit the number of drives connected to the same XMP signal.

The MEIMotorEncoder{} structure has been added to the MEIMotorConfig{} object:

```
typedef struct MEIMotorEncoder {
    MEIXmpEncoderType    type;
    long                 countsPerRev;
} MEIMotorEncoder;

typedef struct MEIMotorConfig {
    MEIMotorEncoder      Encoder[MEIXmpMotorEncoders];
    MEIXmpIO             StatusOutput[MEIXmpMotorStatusOutputs];

    MEIMotorTransceiver  Transceiver[MEIXmpMotorTransceivers];
    long                 UserOutInvert;    /* Opto Polarity */
    MEIMotorStepper      Stepper;
    long                 EncoderTermination;

    /* Commutation is read-only from field Theta to end*/
    MEIXmpCommutationBlock Commutation;

    MEIXmpLimitData      Limit[MEIXmpLimitLAST];

    MEIMotorFilterInput   FilterInput[MEIXmpMotorFilterInputs];
} MEIMotorConfig;
```

# 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.

## 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

|                                          |                          |
|------------------------------------------|--------------------------|
| <a href="#"><u>mpiNotifyCreate</u></a>   | Create a Notify object   |
| <a href="#"><u>mpiNotifyDelete</u></a>   | Delete a Notify object   |
| <a href="#"><u>mpiNotifyValidate</u></a> | Validate a Notify object |

### Event Methods

|                                              |                                                                 |
|----------------------------------------------|-----------------------------------------------------------------|
| <a href="#"><u>mpiNotifyEvent</u></a>        | Check to see if events have occurred                            |
| <a href="#"><u>mpiNotifyEventFlush</u></a>   | Flush pending events from event queue                           |
| <a href="#"><u>mpiNotifyEventMaskGet</u></a> | Get event mask                                                  |
| <a href="#"><u>mpiNotifyEventMaskSet</u></a> | Set event mask                                                  |
| <a href="#"><u>mpiNotifyEventWait</u></a>    | Get next event from queue or wait timeout msec for it to arrive |
| <a href="#"><u>mpiNotifyEventWake</u></a>    | Wake up thread waiting for notify object                        |

## Relational Methods

### List Methods for Event Sources

|                                                |                                            |
|------------------------------------------------|--------------------------------------------|
| <a href="#"><u>mpiNotifySource</u></a>         | Get the indexth event source in list       |
| <a href="#"><u>mpiNotifySourceAppend</u></a>   | Append an event source to list             |
| <a href="#"><u>mpiNotifySourceCount</u></a>    | Count number of event sources in list      |
| <a href="#"><u>mpiNotifySourceFirst</u></a>    | Get first event source in list             |
| <a href="#"><u>mpiNotifySourceIndex</u></a>    | Get index value for event source in list   |
| <a href="#"><u>mpiNotifySourceInsert</u></a>   | Place event source after source in list    |
| <a href="#"><u>mpiNotifySourceLast</u></a>     | Get last event source in list              |
| <a href="#"><u>mpiNotifySourceListGet</u></a>  | Get list of event sources                  |
| <a href="#"><u>mpiNotifySourceListSet</u></a>  | Create a list of event sources             |
| <a href="#"><u>mpiNotifySourceNext</u></a>     | Get next event source after source in list |
| <a href="#"><u>mpiNotifySourcePrevious</u></a> | Get the event source before source in list |
| <a href="#"><u>mpiNotifySourceRemove</u></a>   | Remove event source from list              |

## Data Types

[MPINotifyMessage](#)  
[MEINotifyTrace](#)



# 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

|                      |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to a Notify object                 |
| <b>MPIHandleVOID</b> | if the object could not be created |

## See Also

[mpiNotifyDelete](#) | [mpiNotifyValidate](#)

# 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**

if *NotifyDelete* successfully deletes a Notify object and invalidates its handle

## 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

|                     |                                          |
|---------------------|------------------------------------------|
| <b>MPIMessageOK</b> | if Notify is a handle to a valid object. |
|---------------------|------------------------------------------|

## 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

if the Notify object (**notify**) successfully places the event in a FIFO event queue and signals that an event has been received

## 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**

if *NotifyEventFlush* successfully flushes the pending events from the internal FIFO event queue maintained by the Notify object

## 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**

if *NotifyEventMaskGet* successfully writes the event mask to the location pointed to by mask

## 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](#)

# 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       |                                                                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>NotifyEventMaskSet</i> successfully sets the event type(s) for which notification will be accepted by a Notify object |

## 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](#)



# 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)     | <i>NotifyEventWait</i> will not wait for an event to arrive     |
| MPIWaitFOREVER (-1) | <i>NotifyEventWait</i> will wait forever for an event to arrive |

| Return Values            |                                                                                                                                                 |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>      | if <i>NotifyEventWait</i> successfully sets the contents of the structure pointed to by status, and then removes the first event from the queue |
| <b>MPIMessageTIMEOUT</b> | if no event is present and the contents of status are undefined                                                                                 |

## See Also

[mpiNotifyEventWake](#)

# 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**

if *NotifyEventWake* successfully wakes a thread that is waiting for an event notification from a Notify object

## 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***.

|               |                                |
|---------------|--------------------------------|
| <b>notify</b> | a handle to the Notify object. |
| <b>index</b>  | a position in the list.        |

| Return Values                       |                                                                                                                                                                                          |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i><b>index</b></i> th event source | in the event source list maintained by a Notify object ( <i><b>notify</b></i> )                                                                                                          |
| NULL                                | if <i><b>notify</b></i> is invalid<br>if <i><b>index</b></i> is less than 0<br>if <i><b>index</b></i> is greater than or equal to <a href="#">mpiNotifySourceCount</a> ( <i>notify</i> ) |
| MPIMessageARG_INVALID               | if <i><b>index</b></i> is a negative number.                                                                                                                                             |
| MEIListMessageELEMENT_NOT_FOUND     | if <i><b>index</b></i> is greater than or equal to the number of elements in the list.                                                                                                   |
| MPIMessageHANDLE_INVALID            | if <i><b>notify</b></i> is an invalid handle.                                                                                                                                            |

## 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**).

|               |                                                   |
|---------------|---------------------------------------------------|
| <b>notify</b> | a handle to the Notify object.                    |
| <b>source</b> | a pointer with an arbitrary (but non-NULL) value. |

| Return Values                   |                                                                                                                            |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>             | if <i>NotifySourceAppend</i> successfully appends source to the list of event <b>sources</b> maintained by a Notify object |
| <b>MPIMessageHANDLE_INVALID</b> | Either <b>source</b> is NULL or <b>notify</b> is an invalid handle.                                                        |
| <b>MPIMessageNO_MEMORY</b>      | Not enough memory was available.                                                                                           |

## See Also

# mpiNotifySourceCount

## Declaration

```
long mpiNotifySourceCount(MPINotify notify)
```

**Required Header:** stdmpi.h

## Description

**mpiNotifySourceCount** returns the number of elements on the list.

|               |                                |
|---------------|--------------------------------|
| <b>notify</b> | a handle to the Notify object. |
|---------------|--------------------------------|

| Return Values                  |                                                                          |
|--------------------------------|--------------------------------------------------------------------------|
| <b>number of event sources</b> | in the event source list maintained by a Notify object ( <i>notify</i> ) |
| <b>-1</b>                      | if <i>notify</i> is invalid                                              |
| <b>0</b>                       | 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 conjunction with `mpiNotifySourceNext()` in order to iterate through the list.

|               |                                |
|---------------|--------------------------------|
| <b>notify</b> | a handle to the Notify object. |
|---------------|--------------------------------|

| Return Values                   |                                                                          |
|---------------------------------|--------------------------------------------------------------------------|
| <b>first event source</b>       | in the event source list maintained by a Notify object ( <i>notify</i> ) |
| <b>NULL</b>                     | if <i>notify</i> is invalid or if the event source list is empty         |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>notify</i> 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.

|               |                                                   |
|---------------|---------------------------------------------------|
| <b>notify</b> | a handle to the Notify object.                    |
| <b>source</b> | a pointer with an arbitrary (but non-NULL) value. |

| Return Values          |                                                                                                                           |
|------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>index of source</b> | in the event source list maintained by a Notify object ( <i><b>notify</b></i> )                                           |
| <b>-1</b>              | if <i><b>notify</b></i> is invalid<br>if the event source ( <i><b>source</b></i> ) 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*).

|               |                                |
|---------------|--------------------------------|
| <b>notify</b> | a handle to the Notify object. |
|---------------|--------------------------------|

| Return Values       |                                                                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>NotifySourceInsert</i> successfully places the event source after the specified event source, in the list of event sources that are maintained by a Notify object |

## 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 conjunction with `mpiNotifySourcePrevious()` in order to iterate through the list backwards.

|               |                                |
|---------------|--------------------------------|
| <b>notify</b> | a handle to the Notify object. |
|---------------|--------------------------------|

| Return Values            |                                                                         |
|--------------------------|-------------------------------------------------------------------------|
| <b>last event source</b> | in the list maintained by a Notify object ( <i><b>notify</b></i> )      |
| <b>NULL</b>              | if <i><b>notify</b></i> is invalid<br>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**

if *NotifySourceListGet* successfully returns the event source list for a Notify object

## 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**

if *NotifySourceListSet* successfully creates an *event source* list using the source pointers specified by **sourceList**

## 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 conjunction with mpiNotifySourceFirst() in order to iterate through the list.

|               |                                                   |
|---------------|---------------------------------------------------|
| <b>notify</b> | a handle to the Notify object.                    |
| <b>source</b> | a pointer with an arbitrary (but non-NULL) value. |

| Return Values                   |                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>event source</b>             | before the event source ( <i>source</i> ) in the event source list maintained by a Notify object ( <i>notify</i> )      |
| <b>NULL</b>                     | if <i>notify</i> is invalid<br>if the event source ( <i>source</i> ) is the first event source in the event source list |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i>source</i> is NULL or <i>notify</i> is an invalid handle.                                                     |

## 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 conjunction with mpiNotifySourceLast() in order to iterate through the list backwards.

|               |                                                   |
|---------------|---------------------------------------------------|
| <b>notify</b> | a handle to the Notify object.                    |
| <b>source</b> | a pointer with an arbitrary (but non-NULL) value. |

| Return Values                   |                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>event source</b>             | before the event source ( <i>source</i> ) in the event source list maintained by a Notify object ( <i>notify</i> )      |
| <b>NULL</b>                     | if <i>notify</i> is invalid<br>if the event source ( <i>source</i> ) is the first event source in the event source list |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i>source</i> is NULL or <i>notify</i> is an invalid handle.                                                     |

## 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

if *NotifySourceRemove* successfully removes the event source (***source***) from the list of event sources maintained by a Notify object

## 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.

|                             |                                                                                                                               |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>MEINotifyTraceTHREAD</b> | 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

[mpiObjectValidate](#)

### Configuration and Information Methods

[mpiObjectModuleId](#)

[mpiObjectTimeoutGet](#)

[mpiObjectTimeoutSet](#)

[meiObjectTraceGet](#)

[meiObjectTraceSet](#)

## Data Types

[MPIOBJECTMAP](#)

## Macros

[mpiObjectMapAND\\_ASSIGN](#)

[mpiObjectMapASSIGN](#)

[mpiObjectMapBitCountMAX](#)

[mpiObjectMapBitGET](#)

[mpiObjectMapBitSET](#)

[mpiObjectMapCLEAR](#)

[mpiObjectMapCOMPLEMENT](#)

[mpiObjectMapIS\\_CLEAR](#)

[mpiObjectMapIS\\_EQUAL](#)

[mpiObjectMapIS\\_VALID](#)

[mpiObjectMapMAX](#)

[mpiObjectMapOR\\_ASSIGN](#)

[meiObjectTraceGET](#)

[meiObjectTraceSET](#)

# 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

|                     |                                                                                        |
|---------------------|----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i><b>handle</b></i> is a handle to a valid object of type <i><b>moduleId</b></i> . |
|---------------------|----------------------------------------------------------------------------------------|

## 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

|                     |                                                                              |
|---------------------|------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ObjectModuleId</i> successfully writes the MPIModuleId to the location |
|---------------------|------------------------------------------------------------------------------|

## 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

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ObjectTimeoutGet</i> successfully writes the timeout value to the location |
|---------------------|----------------------------------------------------------------------------------|

## 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 " <i>timeout</i> " is | Then                                                                                                                                                                |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIWaitFOREVER</b>    | <i>ObjectTimeoutSet</i> will wait until the resource becomes available                                                                                              |
| <b>MPIWaitPOLL</b>       | <i>ObjectTimeoutSet</i> will not wait for the resource to become available. If the shared resource is not available, a timeout will be considered to have occurred. |
| <b>a value</b>           | <i>ObjectTimeoutSet</i> will wait timeout milliseconds for the resource to become available                                                                         |

### Return Values

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <b>MPIMessageOK</b>      | if ObjectTimeoutSet successfully sets the timeout value of the object |
| <b>MPIMessageTIMEOUT</b> | if a timeout occurs before the shared resource becomes available      |

## 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.

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <b>handle</b>     | a handle to an object                            |
| <b>*traceMask</b> | a pointer to the value of an object's trace mask |

### Return Values

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ObjectTraceGet</i> 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 <i>ObjectTraceSet</i> 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.

| <b>MPIObjectMap</b> |                                                                                                                                                                                                                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                     | 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

# 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) = ~(objectMap))
```

**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.

## Methods

### Create, Delete, Validate Methods

[mpiPathCreate](#)

Create a Path object

[mpiPathDelete](#)

Delete a Path object

[mpiPathValidate](#)

Validate a Path object

### Configuration and Information Methods

[mpiPathParamsGet](#)[mpiPathParamsSet](#)

### Relational Methods

[mpiPathAppend](#)

### Action Methods

[mpiPathMotionParamsGenerate](#)

## 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\*\*](#)

# mpiPathCreate

## Declaration

```
MPIPath mpiPathCreate();
```

**Required Header:** stdmpi.h

## Description

**mpiPathCreate** creates a Path object.

### Return Values

|                      |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to a Path object                   |
| <b>MPIHandleVOID</b> | 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

|                     |                                                                                      |
|---------------------|--------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>PathDelete</i> successfully deletes the Path object and invalidates its handle |
|---------------------|--------------------------------------------------------------------------------------|

## 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

|                     |                                              |
|---------------------|----------------------------------------------|
| <b>MPIMessageOK</b> | if the Path object and its handle are valid. |
|---------------------|----------------------------------------------|

## 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).

|                  |                                        |
|------------------|----------------------------------------|
| <b>path</b>      | a handle to a Path object              |
| <b>*params</b>   | a pointer to a MPIPathParams structure |
| <b>*external</b> | a pointer to a void or NULL            |

| Return Values       |                                                                 |
|---------------------|-----------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>PathParamsGet</i> successfully reads the path parameters. |

## 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.

|                  |                                        |
|------------------|----------------------------------------|
| <b>path</b>      | a handle to a Path object              |
| <b>*params</b>   | a pointer to a MPIPathParams structure |
| <b>*external</b> | a pointer to a void or NULL            |

| Return Values       |                                                                  |
|---------------------|------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>PathParamsSet</i> successfully writes the path parameters. |

## 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

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>PathAppend</i> successfully adds the elements to the path. |
|---------------------|------------------------------------------------------------------|

## 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 <a href="#">MPIMotionParams</a> structure. |

| Return Values       |                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>MotorAxisMapGet</i> successfully calculates a list of points from the path object and writes them into the motion params structure pointed to by params. |

## 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.

|                 |                                                                                                                                                                    |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>start</b>    | This value defines the arc's starting angle. Units are in degrees.                                                                                                 |
| <b>included</b> | This value defines the relative travel angle. Units are in degrees. Positive values specify counter-clockwise motion and negative values specify clockwise motion. |
| <b>radius</b>   | 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.

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>center</b> | This structure defines the coordinates for the center point of the arc. Please see <a href="#">MPIPathPoint</a> data type documentation for more information.      |
| <b>angle</b>  | 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.

|                  |                                                                                                                                                                    |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>center</b>    | This structure defines the coordinates for the center point of the arc. Please see <a href="#">MPIPathPoint</a> data type documentation for more information.      |
| <b>endPoint</b>  | This structure defines the coordinates for the final point of the arc. Please see <a href="#">MPIPathPoint</a> data type documentation for more information.       |
| <b>direction</b> | This value defines the travel direction, counter-clockwise or clockwise. Please see <a href="#">MPIPathDirection</a> 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

|                            |                                                     |
|----------------------------|-----------------------------------------------------|
| <b>MPIPathDirectionCW</b>  | This value defines the clockwise direction.         |
| <b>MPIPathDirectionCCW</b> | 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

|                    |                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>        | This value defines the type of path element. Please see <a href="#">MPIPathElementType{.}</a> data type documentation for more information.                                                                                             |
| <b>blending</b>    | 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.                             |
| <b>arc</b>         | This structure defines the arc's start angle, included angle, and radius. This structure is used when the type is MPIPathElementTypeARC. Please see <a href="#">MPIPathArc</a> data type documentation for more information.            |
| <b>arcCenter</b>   | This structure defines the arc's center and angle. This structure is used when the type is MPIPathElementTypeARC_CENTER. Please see <a href="#">MPIPathArcCenter</a> data type documentation for more information.                      |
| <b>arcEndPoint</b> | This structure defines the arc's center, end point, and direction. This structure is used when the type is MPIPathElementTypeARC_END_POINT. Please see <a href="#">MPIPathArcEndPoint</a> data type documentation for more information. |
| <b>line</b>        | This structure defines the coordinates for a linear element. This structure is used when the type is MPIPathElementTypeLINE. Please see <a href="#">MPIPathLine</a> data type documentation for more information.                       |
| <b>attributes</b>  | This structure defines the attributes for a path element. Please see <a href="#">MPIPathElementAttributes</a> 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.

|                                        |                                                                                                                                                                                                                       |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIPathElementAttrMaskRELATIVE</b>  | This mask enables relative coordinates for path motion. <b>This feature is not supported and is reserved for future use.</b>                                                                                          |
| <b>MPIPathElementAttrMaskID</b>        | This mask enables an identification tag to be stored in the path. Each element can have a unique identification. Please see <a href="#">MPIPathElementAttributes{.}</a> data type documentation for more information. |
| <b>MPIPathElementAttrMaskVELOCITY</b>  | This mask enables a path velocity to be specified for each element. Please see <a href="#">MPIPathElementAttributes{.}</a> data type documentation for more information.                                              |
| <b>MPIPathElementAttrMaskACCEL</b>     | This mask enables a path acceleration to be specified for each element. Please see <a href="#">MPIPathElementAttributes{.}</a> data type documentation for more information.                                          |
| <b>MPIPathElementAttrMaskTIMESLICE</b> |                                                                                                                                                                                                                       |



## 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

|                                        |                                                                                            |
|----------------------------------------|--------------------------------------------------------------------------------------------|
| <b>MPIPathElementTypeARC</b>           | This type generates an arc specified by the arc's start angle, included angle, and radius. |
| <b>MPIPathElementTypeARC_CENTER</b>    | This type generates an arc specified by the arc's center and angle.                        |
| <b>MPIPathElementTypeARC_END_POINT</b> | This type generates an arc specified by the arc's center, end point, and direction.        |
| <b>MPIPathElementTypeLINE</b>          | 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.

|              |                                                                          |
|--------------|--------------------------------------------------------------------------|
| <b>point</b> | 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

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>dimension</b>     | This value defines the number of axes to coordinate. Please see <a href="#">MPIPathPoint</a> data type documentation for more information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>start</b>         | This structure defines the initial point for the path.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>velocity</b>      | This value defines the speed along the path. The units are in counts per second.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>acceleration</b>  | This value defines the rate of change of speed to reach the velocity along the path. The units are in counts per second * second.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>deceleration</b>  | This value defines the rate of change of speed to reach zero velocity along the path. The units are in counts per second * second.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>interpolation</b> | This value specifies the motion algorithm to generate the path. Please see <a href="#">MPIMotionType</a> data type documentation for more information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>conversion</b>    | <p>This value is an <math>N \times N</math> matrix (where <math>N</math> 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.</p> <p><b>For two axes with different resolution:</b><br/> Set conversion[0][0] to (desired x resolution / actual x resolution)<br/> Set conversion [1][1] to (desired y resolution / actual y resolution)<br/> Set conversion[0][1] and conversion[1][0] = 0</p> <p><b>For a coordinate rotation, where alpha is the rotation of the coordinate system:</b><br/> Set conversion[0][0] and conversion [1][1] = cos(alpha)<br/> Set conversion[0][1] = sin(alpha)<br/> Set conversion[1][0] = -sin(alpha)</p> |

## See Also

[mpiPathParamsGet](#) | [mpiPathParmsSet](#) | [mpiPathMotionParamsGenerate](#) | [MPIPathPointDIMENSION\\_MAX](#)

# MPIPathPoint

## Definition

```
typedef struct MPIPathPoint {  
    double    position[MPIPathPointDIMENSION_MAX];  
} MPIPathPoint;
```

## Description

| <b>position</b>                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------|
| 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](#)

# Platform Objects

## Introduction

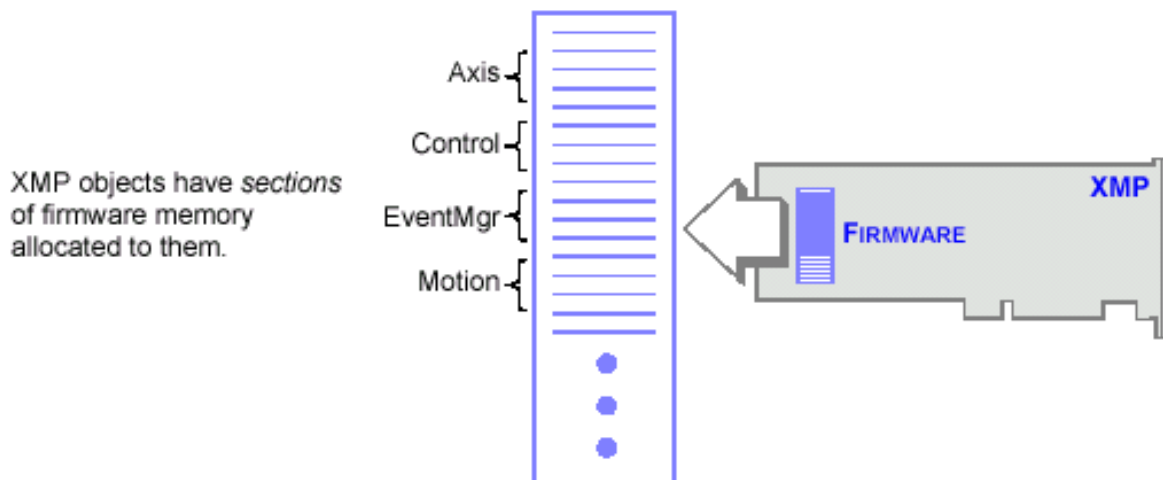
The **Platform** module provides a common interface to platform-specific functionality, such as memory allocation, resource locking, interrupts, signalling, and others.

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 **meiObjectGive/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

[meiPlatformAlloc](#)

Allocate system memory.

[meiPlatformAssertSet](#)

Set an assertion handling function to be used by the MPI library.

[meiPlatformAtof](#)

Convert a numeric string to a double.

[meiPlatformAtol](#)

Convert a numeric string to a long.

[meiPlatformFileClose](#)

Close a file handle.

[meiPlatformFileOpen](#)

Open a file handle.

[meiPlatformFileRead](#)

Read data from a file handle created by meiPlatformFileOpen.

[meiPlatformFileWrite](#)

Writes data to a file whose handle was created by meiPlatformFileOpen.

[meiPlatformFree](#)

Free system memory.

[meiPlatformInfo](#)[meiPlatformKey](#)

Return an input character if an input character is available.

[meiPlatformMemoryToFirmware](#)

Convert a host memory address to a controller memory address.

[meiPlatformMemoryToHost](#)

Covert a controller memory address to a host memory address.

[meiPlatformSleep](#)

Put the current thread to sleep for the number of milliseconds specified.

[meiPlatformProcessId](#)

Return the process identification number of the current process.

[meiPlatformTimerCount](#)

Write to ticks the current timer count.

[meiPlatformTimerFrequency](#)

Write to frequency the timer frequency of the current platform.

[meiPlatformTrace](#)

Display printf(...) style trace information

[meiPlatformTraceEol](#)

Set the end-of-line (eol) to be used by meiPlatformTrace(...).

[meiPlatformTraceFile](#)

Redirect trace output.

[meiPlatformTraceFunction](#)

Display the trace output.

## Data Types

[MEIPlatformBoardType](#)[MEIPlatformFileMode](#)[MEIPlatformInfo](#)[MEIPlatformMessage](#)

## Constants

[MEIPlatformInfoCHAR\\_MAX](#)

# 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.

|                |                                |
|----------------|--------------------------------|
| <b>control</b> | a handle to the Control object |
|----------------|--------------------------------|

| Return Values        |                              |
|----------------------|------------------------------|
| <b>MPIPlatform</b>   | handle to a Platform object  |
| <b>MPIHandleVOID</b> | if <i>control</i> is invalid |

## See Also

[mpiControlCreate](#)



# meiPlatformAlloc

## Declaration

```
void meiPlatformAlloc(long    size)
```

**Required Header:** stdmei.h

## Description

**meiPlatformAlloc** allocates system memory. meiPlatformAlloc will return NULL upon failure to allocate memory.

|             |                                  |
|-------------|----------------------------------|
| <b>size</b> | the number of bytes to allocate. |
|-------------|----------------------------------|

## See Also

[meiPlatformFree](#)

# meiPlatformAssertSet

## Declaration

```
void meiPlatformAssertSet(void (MPI_DECL2 *func)
                          (const char *file, long line));
```

**Required Header:** stdmei.h

## 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 at [Tech Email](#).

**NOTE:** meiPlatformAssertSet is only fully implemented for Windows operating systems.

|                 |                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------|
| <b>platform</b> | the handle to the controller's platform object. This should be obtained from meiControlPlatform(). |
| <b>firmware</b> | the controller memory address to be converted.                                                     |
| <b>\host</b>    | the location where the host memory address will be written.                                        |

## See Also

# 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.

|               |                        |
|---------------|------------------------|
| <b>*ascii</b> | 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.

|               |                        |
|---------------|------------------------|
| <b>*ascii</b> | string to be converted |
|---------------|------------------------|

### Returns

Converted the numeric text string ascii to a ***long*** and returned it

## See Also

[meiPlatformAtof](#)

# 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().

|             |                               |
|-------------|-------------------------------|
| <b>file</b> | 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().

|                 |                                                                                                                                                                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>fileName</b> | the name of the file to open.                                                                                                                                                                                                                      |
| <b>mode</b>     | <p>the access mode used to open the file. Different MEIPlatformFileMode values may be or'ed together to produce the desired mode.</p> <p>For example:<br/><a href="#">MEIPlatformFileModeREAD</a>   <a href="#">MEIPlatformFileModeBINARY</a>)</p> |

### 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().

|                  |                                                                   |
|------------------|-------------------------------------------------------------------|
| <b>file</b>      | the handle of the file from which data will be read.              |
| <b>buffer</b>    | the storage location where data from the file will be written to. |
| <b>byteCount</b> | 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().

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <b>file</b>      | the handle of the file to which data will be written. |
| <b>buffer</b>    | the location of the data to be written to the file.   |
| <b>byteCount</b> | the number of bytes to be written to the file.        |

## See Also

[meiPlatformFileOpen](#) | [meiPlatformFileRead](#)



# meiPlatformFree

## Declaration

```
long meiPlatformFree(void *alloc,  
                      long size)
```

**Required Header:** stdmei.h

## Description

**meiPlatformFree** frees system memory.

|              |                                                                                                                |
|--------------|----------------------------------------------------------------------------------------------------------------|
| <b>alloc</b> | the address of the memory to free.                                                                             |
| <b>size</b>  | the number of bytes to free. This must match the size specified for meiPlatformAlloc when alloc was allocated. |

## See Also

[meiPlatformAlloc](#)

# meiPlatformInfo

## Declaration

```
long meiPlatformInfo(MEIPlatform platform,  
                    MEIPlatformInfo *platformInfo);
```

**Required Header:** stdmei.h

**Change History:** Added in the 03.02.00

## Description

**meiPlatformInfo** reads static information about the host system associated with the Platform object and writes it into the structure pointed to by info. The Platform info structure contains read only data.

|                      |                                          |
|----------------------|------------------------------------------|
| <b>platform</b>      | a handle to the platform object.         |
| <b>*platformInfo</b> | a pointer to the PlatformInfo structure. |

### Returns

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

[MPIMessageHANDLE\\_INVALID](#)

[MPIMessageUNSUPPORTED](#)

## Sample Code

```
MEIPlatformInfo info;  
  
returnValue =  
    meiPlatformInfo(meiControlPlatform(control),  
                    &info);  
meiPlatformConsole("Operating system: %s\n", info.OSInfo);  
meiPlatformConsole("CPU: %s\n", info.CPUInfo);  
meiPlatformConsole("Clock = %d MHz\n", info.CPUMHz);
```

## See Also

[meiControlPlatform](#)

# 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 character is not available, *PlatformKey* waits **wait** milliseconds for an input character to become available.

**NOTE:**  
meiPlatformKey is not fully implemented for all operating systems. For example, in VentureCom's RTX Windows Extensions, a keystroke will be simulated after 10 seconds.

| <i>If "wait" is</i>   | <i>Then</i>                                                                      |
|-----------------------|----------------------------------------------------------------------------------|
| MPIWaitFOREVER (-1)   | <i>PlatformKey</i> will wait for an input character forever                      |
| MPIWaitPOLL (0)       | <i>PlatformKey</i> will return immediately                                       |
| a value (not -1 or 0) | <i>PlatformKey</i> will wait for an input character for <b>wait</b> milliseconds |

| Return Values                                 |                                                                                                                                                                                                            |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -1                                            | if no input character was available                                                                                                                                                                        |
| 0                                             | <i>PlatformKey</i> has read a non-zero character (typically a function key or other non-ASCII value), and <b>meiPlatformKey(...)</b> should be called AGAIN immediately to receive that non-zero character |
| a value (not -1 or 0)<br>(an ASCII character) | (typically a keystroke) if an input character is available                                                                                                                                                 |

## See Also

[meiSqNodeCreate](#) | [meiSqNodeValidate](#)

# meiPlatformMemoryToFirmware

## Declaration

```
long meiPlatformMemoryToFirmware(MEIPlatform platform,
                                void *host,
                                void **firmware)
```

Required Header: stdmei.h

## Description

**meiPlatformMemoryToFirmware** converts a host memory address to a controller memory address.

|                 |                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------|
| <b>platform</b> | the handle to the controller's platform object. This should be obtained from meiControlPlatform(). |
| <b>host</b>     | the host memory address to be converted.                                                           |
| <b>firmware</b> | the location where the controller's memory address will be written.                                |

## See Also

[meiPlatformMemoryToHost](#) | [meiControlPlatform](#)

# meiPlatformMemoryToHost

## Declaration

```
long meiPlatformMemoryToHost(MEIPlatform platform,
                             void *firmware,
                             void **host)
```

Required Header: stdmei.h

## Description

**meiPlatformMemoryToHost** covers a controller memory address to a host memory address.

|                 |                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------|
| <b>platform</b> | the handle to the controller's platform object. This should be obtained from meiControlPlatform(). |
| <b>firmware</b> | the controller memory address to be converted.                                                     |
| <b>host</b>     | 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).

|                     |                                                                         |
|---------------------|-------------------------------------------------------------------------|
| <b>milliseconds</b> | 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(...).

|                   |                                                    |
|-------------------|----------------------------------------------------|
| <b>*frequency</b> | 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.

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <b>node</b> | 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(...).

|                     |                                                                         |
|---------------------|-------------------------------------------------------------------------|
| <b>milliseconds</b> | 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**

if *PlatformTraceFile* successfully closes any *previously opened* trace file and redirects trace output to ***fileName***

## 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 <i>traceFunction</i> | if there is a previous function                                                    |
| NULL                              | if no <i>traceFunction</i> has been specified (the default trace function is used) |

## See Also

[meiPlatformTrace](#)

# MEIPlatformBoardType

## Definition

```
typedef enum {
    MEIPlatformBoardTypeUNKNOWN,
    MEIPlatformBoardTypeXMP,
    MEIPlatformBoardTypeZMP,
} MEIPlatformBoardType;
```

## Description

**MEIPlatformBoardType** is the type of motion controller card that is being used.

|                                    |                                                     |
|------------------------------------|-----------------------------------------------------|
| <b>MEIPlatformBoardTypeUNKNOWN</b> | Board is not being recognized.                      |
| <b>MEIPlatformBoardTypeXMP</b>     | An XMP Motion Controller board has been recognized. |
| <b>MEIPlatformBoardTypeZMP</b>     | 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.

|                                  |                                                          |
|----------------------------------|----------------------------------------------------------|
| <b>MEIPlatformFileModeREAD</b>   | Open a file for read access                              |
| <b>MEIPlatformFileModeWRITE</b>  | Open a file for write access                             |
| <b>MEIPlatformFileModeTEXT</b>   | Open a file as text format                               |
| <b>MEIPlatformFileModeBINARY</b> | Open a file as binary format                             |
| <b>MEIPlatformFileModeTRUNC</b>  | Truncate existing file or create for reading and writing |
| <b>MEIPlatformFileModeAPPEND</b> | 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.

|                |                                                       |
|----------------|-------------------------------------------------------|
| <b>OSInfo</b>  | a string containing the Operating System information. |
| <b>CPUInfo</b> | a string containing the CPU information.              |
| <b>CPUMHz</b>  | the CPU clock speed in MHz                            |

## See Also

[MEIPlatformInfoCHAR\\_MAX](#)



# MEIPlatformMessage

## Definition

```
typedef enum {
    MEIPlatformMessagePLATFORM_INVALID,
    MEIPlatformMessageDEVICE_INVALID,
    MEIPlatformMessageDEVICE_ERROR,
    MEIPlatformMessageDEVICE_MAP_ERROR,
} MEIPlatformMessage;
```

**Required Header:** stdmei.h

## 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).

## 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.

## Methods

[mpiProbeConfigGet](#)

[mpiProbeConfigSet](#)

[mpiProbeControl](#)

[mpiProbeCreate](#)

[mpiProbeDelete](#)

[mpiProbeInfo](#)

[mpiProbeParams](#)

[mpiProbeStatus](#)

[mpiProbeValidate](#)

## Data Types

[MEIProbeAddress](#)

[MPIProbeConfig](#)

[MPIProbeData](#)

[MPIProbeInfo](#)

[MPIProbeMessage](#) / [MEIProbeMessage](#)

[MPIProbeNumber](#)

[MPIProbeParams](#)

[MPIProbeSource](#)

[MPIProbeStatus](#)

[MEIProbeTrace](#)

[MPIProbeType](#)

## Constants

[MPIProbeMaxProbeRegisters](#)

# 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.

|                  |                                                    |
|------------------|----------------------------------------------------|
| <b>probe</b>     | a handle to a Probe object.                        |
| <b>*config</b>   | a pointer to a Probe configuration structure.      |
| <b>*external</b> | a pointer to an implementation specific structure. |

| Return Values                       |                                                                                          |
|-------------------------------------|------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                 | if <i>ProbeConfigGet</i> successfully reads the Probe configuration from the controller. |
| <b>MPIProbeMessagePROBE_INVALID</b> | if the specified Probe params or configurations are out of range.                        |

## 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.

|                  |                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>probe</b>     | a handle to a Probe object.                                                                                                                  |
| <b>*config</b>   | a pointer to a Probe configuration structure.                                                                                                |
| <b>*external</b> | a pointer to an implementation specific structure.<br><i><b>external</b></i> is reserved for future functionality and should be set to NULL. |

| Return Values                            |                                                                                        |
|------------------------------------------|----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                      | if <i>ProbeConfigSet</i> successfully wrote the Probe configuration to the controller. |
| <b>MPIProbeMessagePROBE_TYPE_INVALID</b> | if the specified Probe data type is not one of the enumerated values                   |
| <b>MPIProbeMessagePROBE_INVALID</b>      | if the specified Probe params or configurations are out of range.                      |

## 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.

|              |                             |
|--------------|-----------------------------|
| <b>probe</b> | a handle to a Probe object. |
|--------------|-----------------------------|

| Return Values        |                             |
|----------------------|-----------------------------|
| <b>MPIControl</b>    | handle to a Control object. |
| <b>MPIHandleVOID</b> | 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.

|                |                                             |
|----------------|---------------------------------------------|
| <b>control</b> | a handle to a Probe object.                 |
| <b>*params</b> | a pointer to a probe parameter's structure. |

| Return Values                       |                                                                                                                                                                                                                                                     |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>handle</b>                       | handle to a Probe object.                                                                                                                                                                                                                           |
| <b>MPIHandleVOID</b>                | if the object could not be created.                                                                                                                                                                                                                 |
| <b>MPIProbeMessagePROBE_INVALID</b> | 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 <a href="#">Node FPGA Images: Features Table</a> 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.

|              |                             |
|--------------|-----------------------------|
| <b>probe</b> | a handle to a Probe object. |
|--------------|-----------------------------|

### Return Values

|                     |                                                                                       |
|---------------------|---------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ProbeDelete</i> successfully deletes a Probe object and invalidates its handle. |
|---------------------|---------------------------------------------------------------------------------------|

## 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.

|              |                                                 |
|--------------|-------------------------------------------------|
| <b>probe</b> | a handle to a Probe object.                     |
| <b>*info</b> | contains read only data about the probe engine. |

| Return Values       |                                                                                                                            |
|---------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | If <i>meiProbeInfo</i> successfully reads a Probe's information and writes it into a structure pointed to by <i>info</i> . |

## 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\(.\)](#).

|                |                                             |
|----------------|---------------------------------------------|
| <b>probe</b>   | a handle to a Probe object.                 |
| <b>*params</b> | a pointer to a Probe parameter's structure. |

### Return Values

|                     |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | If <i>mpiProbeInfo</i> successfully reads the parameters of a Probe object and writes it to the structure pointed to by <i>params</i> . |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|

## 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.

|                  |                                                                                                                                    |
|------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>probe</b>     | a handle to a Probe object.                                                                                                        |
| <b>*status</b>   | a pointer to a Probe status structure.                                                                                             |
| <b>*external</b> | a pointer to an implementation specific structure. <b>external</b> is reserved for future functionality and should be set to NULL. |

| Return Values       |                                                                                                                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | If <i>mpiProbeInfo</i> successfully 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). |

## See Also

[mpiProbeInfo](#) | [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.

|              |                             |
|--------------|-----------------------------|
| <b>probe</b> | a handle to a Probe object. |
|--------------|-----------------------------|

### Return Values

|                     |                                         |
|---------------------|-----------------------------------------|
| <b>MPIMessageOK</b> | if Probe is a handle to a valid object. |
|---------------------|-----------------------------------------|

## 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.

|                           |                                                                       |
|---------------------------|-----------------------------------------------------------------------|
| <b>*primaryAddress</b>    | a pointer to the address for the Motor's primary position feedback.   |
| <b>*secondaryPosition</b> | a pointer to the address for the Motor's secondary position feedback. |
| <b>*time</b>              | a pointer to the address for the feedback time value.                 |
| <b>*status</b>            | a pointer to the address for the Probe status                         |
| <b>*data</b>              | a pointer to the address for the Probe data.                          |

## See Also

[MEIProbeInfo](#)

# MPIProbeConfig

## Definition

```
typedef struct MPIProbeConfig {  
    long          enable;          /* TRUE/FALSE */  
    MPIProbeSource source;  
    MPIProbeData  data;  
    long          inputFilter;    /* TRUE/FALSE */  
} MPIProbeConfig;
```

## Description

**MPIProbeConfig** specifies the Probe's configuration.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>enable</b>      | Enables or disables the Probe triggering. A value of TRUE enables the Probe triggering, FALSE disables Probe triggering.                                                                                                                                                                                                                                                                                                                                     |
| <b>source</b>      | An enumerated Probe trigger source input. A Probe can be configured to trigger from one input source.                                                                                                                                                                                                                                                                                                                                                        |
| <b>data</b>        | An enumerated Probe data type. A probe can be configured to collect position or time data from a motor's feedback.                                                                                                                                                                                                                                                                                                                                           |
| <b>inputFilter</b> | <p>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.</p> <p>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.</p> |

## 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.

|                                       |                                                                                                                                                                                                                                                                       |
|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIProbeDataPOSITION_PRIMARY</b>   | Position from the motor's primary feedback                                                                                                                                                                                                                            |
| <b>MPIProbeDataPOSITION_SECONDARY</b> | Position from the motor's secondary feedback                                                                                                                                                                                                                          |
| <b>MPIProbeDataTIME</b>               | 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.

|                |                                                                             |
|----------------|-----------------------------------------------------------------------------|
| <b>address</b> | 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,
    MPIProbeMessageNOT_INITIALIZED,
} MPIProbeMessage;
```

## Description

**MPIProbeMessage** is an enumeration of SynqNet node error messages that can be returned by the MPI library.

## Description

### 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 {  
    MEIProbeMessageFIRST = MPIProbeMessageLAST,  
    MEIProbeMessageLAST  
} MEIProbeMessage;
```

## 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.

|              |                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b> | An index to identify the motor that is associated with the Probe. Must be specified for MPIProbeTypeMOTOR Probe types. |
| <b>node</b>  | 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.

|                   |                                                                                                                                                                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>       | An enumerated Probe type. Determines the Probe number definition.                                                                                                                                                                                         |
| <b>number</b>     | An union to identify the object number that is associated with the Probe. The Probe type defines the number.                                                                                                                                              |
| <b>probeIndex</b> | 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 <a href="#">meiMotorInfo(.)</a> and <a href="#">meiSqNodeInfo(.)</a> . |

## 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.

|                                      |                                                             |
|--------------------------------------|-------------------------------------------------------------|
| <b>MPIProbeSourceHOME</b>            | Home input in the dedicated I/O                             |
| <b>MPIProbeSourceINDEX</b>           | Index input from the primary encoder in the dedicated I/O   |
| <b>MPIProbeSourceLIMIT_HW_NEG</b>    | Hardware Negative Limit input in the dedicated I/O          |
| <b>MPIProbeSourceLIMIT_HW_POS</b>    | Hardware Positive Limit input in the dedicated I/O          |
| <b>MPIProbeSourceINDEX_SECONDARY</b> | Index input from the secondary encoder in the dedicated I/O |

|                                  |                                              |
|----------------------------------|----------------------------------------------|
| <b>MPIProbeSourceMOTOR_IO_0</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_1</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_2</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_3</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_4</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_5</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_6</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_7</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_8</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_9</b>  | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_10</b> | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_11</b> | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_12</b> | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_13</b> | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_14</b> | Bit number 0 in the Motor's configurable I/O |
| <b>MPIProbeSourceMOTOR_IO_15</b> | 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.

|                     |                                                                                                                                                                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>maxRegisters</b> | 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.                                                                                  |
| <b>index</b>        | 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.                                                                            |
| <b>io</b>           | <p>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.</p> <p>For example, if io = 0x3, then regs[0] and regs[1] have valid Probe data.</p> |
| <b>regs</b>         | 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.

|                          |                                                                                                                                |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIProbeTypeMOTOR</b> | The Probe hardware engine is a component of the Motor object. For the Motor type, the Probe is identified by the Motor number. |
| <b>MPIProbeTypeIO</b>    | 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](#).

| [Buffer Size](#) |

## Methods

### Create, Delete, Validate Methods

[mpiRecorderCreate](#)

Create Recorder object

[mpiRecorderDelete](#)

Delete Recorder object

[mpiRecorderValidate](#)

Validate Recorder object

## Configuration and Information Methods

[mpiRecorderConfigGet](#)

Get Recorder's configuration

[mpiRecorderConfigSet](#)

Set Recorder's configuration

[mpiRecorderRecordConfig](#)

Configure type of data record that Recorder will capture

[mpiRecorderStatus](#)

Get status of Recorder

## Event Methods

[mpiRecorderEventNotifyGet](#)

Get event mask of events for which host notification has been requested

[mpiRecorderEventNotifySet](#)

Set event mask of events for which host notification will be requested

[mpiRecorderEventReset](#)

Reset the events specified in event mask that are generated by Recorder

## Action Methods

[mpiRecorderRecordGet](#)

Get data records from Recorder

[mpiRecorderStart](#)

Start recording data records using Recorder

[mpiRecorderStop](#)

Stop recording data records using Recorder

## Memory Methods

[mpiRecorderMemory](#)

Get address to Recorder's memory

[mpiRecorderMemoryGet](#)

Copy data from Recorder memory to application memory

[mpiRecorderMemorySet](#)

Copy data from application memory to Recorder memory

## Relational Methods

[mpiRecorderControl](#)

Return handle of Control object associated with Recorder

[mpiRecorderNumber](#)

## Data Types

[MPIRecorderConfig](#) / [MEIRecorderConfig](#)[MPIRecorderMessage](#)[MPIRecorderRecord](#) / [MEIRecorderRecord](#)[MEIRecorderRecordAxis](#)[MEIRecorderRecordFilter](#)[MPIRecorderRecordPoint](#)[MPIRecorderRecordType](#) / [MEIRecorderRecordType](#)[MPIRecorderStatus](#)[MEIRecorderTrace](#)[MEIRecorderTrigger](#)

[MEIRecorderTriggerCondition](#)

[MEIRecorderTriggerIndex](#)

[MEIRecorderTriggerType](#)

[MEIRecorderTriggerUser](#)

## Constants

[MPIRecorderADDRESS\\_COUNT\\_MAX](#)

[MEIRecorderMAX\\_AXIS\\_RECORDS](#)

[MEIRecorderMAX\\_FILTER\\_RECORDS](#)

# 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](#).

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>control</b> | a handle to a Control object.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>number</b>  | <p>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.</p> <p>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.</p> |



| Return Values        |                                             |
|----------------------|---------------------------------------------|
| <b>handle</b>        | to a Recorder object                        |
| <b>MPIHandleVOID</b> | 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](#).

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>control</b> | a handle to a Control object.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>number</b>  | <p>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.</p> <p>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.</p> |

| Return Values       |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>RecorderDelete</i> successfully deletes a Recorder object and invalidates its handle |

## 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](#).

|                 |                               |
|-----------------|-------------------------------|
| <b>recorder</b> | a handle to a Recorder object |
|-----------------|-------------------------------|

| Return Values                               |                                                                                                       |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                         | if Recorder is a handle to a valid object.                                                            |
| <b>MPIRecorderMessageNOT_ENABLED</b>        | if the specified recorder number has not been enabled in the controller.                              |
| <b>MPIRecorderMessageNO_RECORDERS_AVAIL</b> | if the specified recorder number is (-1) and there are no more recorders available on the controller. |

## 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**

if *RecorderConfigGet* successfully writes the Recorder's configuration to the structure(s)

## See Also

[MPIRecorderConfig](#) | [mpiRecorderConfigSet](#)

# 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  | if <i>RecorderConfigSet</i> successfully sets the Recorder's configuration using data from the structure(s) |

## See Also

[MEIRecorderConfig](#) | [mpiRecorderConfigGet](#)

# 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  | <i>count</i> data points will be recorded, and <i>handle</i> points to an array of <i>count</i> controller addresses                         |
| MEIRecorderRecordTypeAXIS   | <i>count</i> records of type MPIRecorderRecordAxis{} will be recorded, and <i>handle</i> points to an array of <i>count</i> Axis handles     |
| MEIRecorderRecordTypeFILTER | <i>count</i> records of type MPIRecorderRecordFilter{} will be recorded, and <i>handle</i> points to an array of <i>count</i> Filter handles |

| Return Values |                                                                                                          |
|---------------|----------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>RecorderRecordConfig</i> successfully configures the type of record that the Recorder will capture |

## See Also

[MPIRecorderRecordAxis](#) | [MPIRecorderRecordFilter](#)

# 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.

|                  |                                                   |
|------------------|---------------------------------------------------|
| <b>recorder</b>  | a handle to a Recorder object                     |
| <b>*status</b>   | a pointer to Recorder's status structure          |
| <b>*external</b> | a pointer to an implementation-specific structure |

| Return Values                |                                                                                                      |
|------------------------------|------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>RecorderStatus</i> successfully gets the Recorder's status and writes it into the structure(s) |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL.                                                                |

## 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  | if RecorderEventNotifyGet successfully writes the event mask to the structure(s) |

## 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**

if *RecorderEventNotifySet* successfully requests host notification of the event(s) as specified by the structure(s)

**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 latching events have occurred.

| Return Values       |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>RecorderEventReset</i> successfully resets the event(s) that are specified in <b><i>eventMask</i></b> and generated by a Recorder |

## See Also

# 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

|                     |                                                                |
|---------------------|----------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>RecorderRecordGet</i> successfully gets the data records |
|---------------------|----------------------------------------------------------------|

## See Also

[mpiRecorderRecordConfig](#)

# 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(.)`.

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>recorder</b> | a handle to a Recorder object                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>count</b>    | 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                    |                                                          |
|----------------------------------|----------------------------------------------------------|
| <b>MPIMessageOK</b>              | if the data recorder successfully begins to record data. |
| <b>MPIRecorderMessageSTARTED</b> | if the data recorder is already running.                 |

## See Also

[mpiRecorderRecordConfig](#) | [mpiRecorderStop](#) | [mpiRecorderConfigGet](#) | [mpiRecorderConfigSet](#) | [mpiControlConfigGet](#) | [mpiControlConfigSet](#)

# mpiRecorderStop

## Declaration

```
long mpiRecorderStop(MPIRecorder recorder)
```

**Required Header:** stdmpi.h

## Description

**mpiRecorderStop** instructs a Recorder (***recorder***) to stop recording data records.

|                 |                               |
|-----------------|-------------------------------|
| <b>recorder</b> | a handle to a Recorder object |
|-----------------|-------------------------------|

### Return Values

|                                  |                                                                                                                                                                                                                                        |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>              | if <i>RecorderStop</i> successfully stops recording data records                                                                                                                                                                       |
| <b>MPIRecorderMessageSTOPPED</b> | This means the the recorder was already stopped when <i>mpiRecorderStop</i> was called. This is a warning, not an error. This can be ignored if the user does not have some reason for why the recorder must be running at this point. |

## 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**

if *RecorderMemory* successfully writes the Recorder's memory address to the contents of *memory*

## See Also

[mpiRecorderMemoryGet](#) | [mpiRecorderMemorySet](#)

# mpiRecorderMemoryGet

## Declaration

```
long mpiRecorderMemoryGet(MPIRecorder recorder,  
                           void          *dst,  
                           void          *src,  
                           long          count)
```

**Required Header:** stdmpi.h

## 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**

if *RecorderMemoryGet* successfully copies data from Recorder memory to application memory

## See Also

[mpiRecorderMemory](#) | [mpiRecorderMemorySet](#)



# mpiRecorderMemorySet

## Declaration

```
long mpiRecorderMemorySet(MPIRecorder recorder,  
                           void          *dst,  
                           void          *src,  
                           long          count)
```

Required Header: stdmpi.h

## 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  | if <i>RecorderMemorySet</i> successfully copies data from application memory to Recorder memory |

## 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        |                                                        |
|----------------------|--------------------------------------------------------|
| <b>handle</b>        | to a Control object that a Recorder is associated with |
| <b>MPIHandleVOID</b> | 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.).

|                 |                                              |
|-----------------|----------------------------------------------|
| <b>recorder</b> | a handle to a Recorder object.               |
| <b>*number</b>  | a pointer to the index of a Recorder object. |

| Return Values                   |                                                                                                                                                 |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>             | if <i>RecorderNumber</i> successfully gets the index of a data recorder and writes it into the contents of a long pointed to by <i>number</i> . |
| <b>MPIMessageARG_INVALID</b>    | See <a href="#">description</a> .                                                                                                               |
| <b>MPIMessageHANDLE_INVALID</b> | See <a href="#">description</a> .                                                                                                               |

## 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 */
    long   bufferWrap;  /* TRUE/FALSE */

    long   addressCount; /* number of data point addresses in address[] */
    void   *address[MPIRecorderADDRESS_COUNT_MAX];
} MPIRecorderConfig;
```

## 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.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>period</b>       | 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.                                                                                                                                                                                                                                                                                                       |
| <b>highCount</b>    | 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 <a href="#">mpiControlConfigSet(.)</a> .                                                                                                                                                                                                                                                                                                                                                                       |
| <b>bufferWrap</b>   | 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. |
| <b>addressCount</b> | The number of controller addresses in the address array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>*address</b>     | 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.

|                |                                                             |
|----------------|-------------------------------------------------------------|
| <b>trigger</b> | 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,
} MPIRecorderMessage;
```

## 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

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.

## **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 */
    long    command;     /* command position */
    long    actual;       /* actual position */
    float    dac;         /* voltage */
} MEIRecorderRecordAxis;
```

## Description

|         |                                                                      |
|---------|----------------------------------------------------------------------|
| sample  | The XMP sample number in which the following values were recorded.   |
| command | The command position of the axis.                                    |
| actual  | The actual position of the axis.                                     |
| dac     | The output of the primary DAC of the motor associated with the axis. |

## See Also

# MEIRecorderRecordFilter

## Definition

```
typedef struct MEIRecorderRecordFilter {
    long    sample;    /* sample number */
    long    command;   /* command position */
    long    actual;     /* actual position */
    float    dac;       /* voltage */
} MEIRecorderRecordFilter;
```

## Description

|         |                                                                                  |
|---------|----------------------------------------------------------------------------------|
| sample  | The XMP sample number in which the following values were recorded                |
| command | The command position the filter uses to calculate the filter output.             |
| actual  | The actual position (of an axis) the filter uses to calculate the filter output. |
| 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 {
    long    enabled;
    long    full;
    long    recordCount;
    long    recordCountMax;
} MPIRecorderStatus;
```

## Description

|                       |                                                                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>enabled</b>        | If the recorder is enabled (recording) then enabled will equal a non-zero value (-1), otherwise enabled will equal 0.                             |
| <b>full</b>           | If the recorder is full (the number of stored records >= MPIRecorderConfig.fullCount) then full will equal TRUE, otherwise full will equal FALSE. |
| <b>recordCount</b>    | The number of stored records in the recorder.                                                                                                     |
| <b>recordCountMax</b> | The maximum number of records the recorder can store.                                                                                             |

## See Also

[mpiRecorderStatus](#)

# 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.

|             |                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------|
| <b>type</b> | The data recorder trigger type. See the <a href="#">MEIRecorderTriggerType</a> enumeration.        |
| <b>user</b> | The configurations for a user specified trigger type. See <a href="#">MEIRecorderTriggerUser</a> . |

## See Also

[MEIRecorderTrigger](#) | [mpiRecorderConfigGet](#) | [mpiRecorderConfigSet](#)

# MEIRecorderTriggerCondition

## Definition

```
typedef enum MEIRecorderTriggerCondition {  
    MEIRecorderTriggerConditionMATCH,  
    MEIRecorderTriggerConditionCHANGE,  
} MEIRecorderTriggerCondition;
```

## Description

**MEIRecorderTriggerCondition** is an enumeration of a data recorder's trigger conditions. The mask and pattern fields referred to are from the [MEIRecorderTriggerUser](#) structure.

|                                        |                                                                                                                                                                                                 |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIRecorderTriggerTriggerMATCH</b>  | Triggers when the value at the specified address ANDed with the <b>mask</b> is equal to the specified <b>pattern</b> .                                                                          |
| <b>MEIRecorderTriggerTriggerCHANGE</b> | Triggers when the value at the specified address ANDed with the <b>mask</b> changes. The <b>pattern</b> field is only used to set the initial bit pattern used to determine if a change occurs. |

## 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.

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <b>MEIRecorderTriggerIndexSTART</b> | Index to a data recorder's start trigger. |
| <b>MEIRecorderTriggerIndexSTOP</b>  | 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.

|                                       |                                                                                                                          |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>MEIRecorderTriggerTypeDISABLED</b> | The data recorder trigger is not enabled.                                                                                |
| <b>MEIRecorderTriggerTypeUSER</b>     | The data recorder trigger is user configurable. See the <a href="#">MEIRecorderTriggerUser{.}</a> 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.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>condition</b> | The logic that determines how to evaluate the addr, mask, and pattern. See the <a href="#">MEIRecorderTriggerCondition</a> enumeration.                                                                                                                                                                                                                                                                                                                                                                             |
| <b>*addr</b>     | A pointer to a controller address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>mask</b>      | A bit mask ANDed with the value at the controller address.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>pattern</b>   | A bit pattern compared to the masked value at the controller address.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>count</b>     | <p>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 <a href="#">mpiControlConfigSet(.)</a>.</p> <p>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.</p> <p>When used for the stop trigger, <i>count</i> records will be collected after the trigger has triggered.</p> |

## 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 (8)
```

## 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 (8)
```

## 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 [Special Note](#) on *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.

## Methods

### Create, Delete, Validate Methods

|                                     |                          |
|-------------------------------------|--------------------------|
| <a href="#">mpiSequenceCreate</a>   | Create Sequence object   |
| <a href="#">mpiSequenceDelete</a>   | Delete Sequence object   |
| <a href="#">mpiSequenceValidate</a> | Validate Sequence object |

### Configuration and Information Methods

|                                           |                                                          |
|-------------------------------------------|----------------------------------------------------------|
| <a href="#">mpiSequenceConfigGet</a>      | Get sequence config                                      |
| <a href="#">mpiSequenceConfigSet</a>      | Set sequence config                                      |
| <a href="#">mpiSequenceFlashConfigGet</a> | Get sequence flash config                                |
| <a href="#">mpiSequenceFlashConfigSet</a> | Set sequence flash config                                |
| <a href="#">mpiSequencePageSize</a>       | Set pageSize to number of command slots used by sequence |
| <a href="#">mpiSequenceStatus</a>         | Return sequence status                                   |

### Event Methods

|                                           |                                                      |
|-------------------------------------------|------------------------------------------------------|
| <a href="#">mpiSequenceEventNotifyGet</a> | Select an event mask for host notification of events |
| <a href="#">mpiSequenceEventNotifySet</a> | Enable host notification of sequence events          |
| <a href="#">mpiSequenceEventReset</a>     | Reset sequence events                                |



## Action Methods

[mpiSequenceCompile](#)

[mpiSequenceLoad](#)

[mpiSequenceResume](#)

[mpiSequenceStart](#)

[mpiSequenceStep](#)

[mpiSequenceStop](#)

Load sequence commands into firmware

Resume execution of sequence

Start execution of sequence

Execute count steps of a stopped sequence

Stop sequence

## Memory Methods

[mpiSequenceMemory](#)

[mpiSequenceMemoryGet](#)

[mpiSequenceMemorySet](#)

Set address used to access sequence memory

Get bytes of sequence memory and put into application memory

Put (set) bytes of application memory into sequence memory

## Relational Methods

[mpiSequenceControl](#)

[mpiSequenceNumber](#)

### List Methods for Event Sources

[mpiSequenceCommand](#)

[mpiSequenceCommandAppend](#)

[mpiSequenceCommandCount](#)

[mpiSequenceCommandFirst](#)

[mpiSequenceCommandIndex](#)

[mpiSequenceCommandInsert](#)

[mpiSequenceCommandLast](#)

[mpiSequenceCommandListGet](#)

[mpiSequenceCommandListSet](#)

[mpiSequenceCommandNext](#)

[mpiSequenceCommandPrevious](#)

[mpiSequenceCommandRemove](#)

Get handle to Control

Get index number of sequence

Return handle to indexed command of sequence

Append command to sequence

Count the number of commands in sequence

Return handle to first command in sequence

Return the index of a command in sequence

Insert command into sequence

Return handle of last command in sequence

Get list of commands in sequence

Set list of commands in sequence

Get handle to next command in list

Get handle to previous command in list

Remove command from list

## Data Types

[MPISequenceConfig](#) / [MEISequenceConfig](#)

[MPISequenceMessage](#)

[MPISequenceState](#)

[MPISequenceStatus](#)

[MEISequenceTrace](#)

## See Also

[MPICommand](#)

[MPICommandType](#)

[MPICommandParams](#)

[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                                                                                                                                                                                              |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>number</b> is -1   | <i>SequenceCreate</i> 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. |
| <b>pageSize</b> is -1 | <i>SequenceCreate</i> will allocate all remaining firmware command slots, which may prevent any more Sequence objects from being created.                                                         |

| Return Values        |                                    |
|----------------------|------------------------------------|
| <b>handle</b>        | to a Sequence object               |
| <b>MPIHandleVOID</b> | 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.

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
|-----------------|----------------------------------|

### Return Values

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceDelete</i> successfully a Sequence object and invalidates its handle |
|---------------------|------------------------------------------------------------------------------------|

## 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

|                     |                                            |
|---------------------|--------------------------------------------|
| <b>MPIMessageOK</b> | if Sequence is a handle to a valid object. |
|---------------------|--------------------------------------------|

## 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**

if *SequenceConfigGet* successfully gets and writes the configuration of a Sequence object into the structure(s)

## 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**

if *SequenceConfigSet* successfully sets a Sequence's configuration using data from the structure(s).

## 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

|                     |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceFlashConfigGet</i> successfully writes the Sequence's flash configuration to the structure(s) |
|---------------------|-------------------------------------------------------------------------------------------------------------|

## 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

if *SequenceFlashConfigSet* successfully sets the Sequence's flash configuration using data from the structure(s)

## 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**

if *SequencePageSize* successfully writes the number of command slots (available to the Sequence) to the contents of ***pageSize***

## 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.

|                  |                                                   |
|------------------|---------------------------------------------------|
| <b>sequence</b>  | a handle to a Sequence object                     |
| <b>*status</b>   | a pointer to Sequence's status structure          |
| <b>*external</b> | a pointer to an implementation-specific structure |

| Return Values                |                                                                                                               |
|------------------------------|---------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SequenceStatus</i> successfully returns the Sequence's status and writes the status to the structure(s) |
| <b>MPIMessageARG_INVALID</b> | if the <i>status</i> pointer is NULL.                                                                         |

## 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       |                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceEventNotifyGet</i> successfully writes the event mask to the structure(s) |

## 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       |                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceEventNotifySet</i> successfully requests host notification of the events in the event mask(s) |

## 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**

if *SequenceEventReset* successfully resets the event(s) that are specified in ***eventMask*** and generated by a Sequence object

## See Also

# 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.

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
|-----------------|----------------------------------|

| Return Values       |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceCompile</i> successfully reads a Sequence object's list of Command objects and creates an equivalent list of XMP commands |

## See Also



# mpiSequenceLoad

## Declaration

```
long mpiSequenceLoad(MPISequence sequence,  
                    MPICommand command,  
                    long start)
```

Required Header: stdmpi.h

## 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                                                                               |
|----------------------------------------|------------------------------------------------------------------------------------|
| <b><i>command</i></b> is MPIHandleVOID | <i>SequenceLoad</i> loads Commands starting with the first Command of the Sequence |
| <b><i>start</i></b> is not FALSE       | <i>SequenceLoad</i> starts the sequence after the commands are loaded              |

| Return Values       |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceLoad</i> successfully loads the firmware command slots of a Sequence |

## 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**

if *SequenceResume* successfully resumes a Sequence from the point where the Sequence has stopped

## 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

if *SequenceStart* successfully begins the execution of a Command Sequence

## 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**

if *SequenceStep* successfully executes ***count*** steps (Commands) of a stopped Sequence

## 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

|                     |                                                                              |
|---------------------|------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceStop</i> successfully stops a Sequence (while it is executing) |
|---------------------|------------------------------------------------------------------------------|

## 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

if *SequenceMemory* successfully writes the address (used to access Sequence memory) to the contents of memory

## See Also

[mpiSequenceMemoryGet](#) | [mpiSequenceMemorySet](#)

# mpiSequenceMemoryGet

## Declaration

```
long mpiSequenceMemoryGet(MPISequence sequence,  
                           void          *dst,  
                           void          *src,  
                           long          count)
```

**Required Header:** stdmpi.h

## 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

if *SequenceMemoryGet* successfully copies count bytes of Sequence memory to application memory

## See Also

[mpiSequenceMemorySet](#) | [mpiSequenceMemory](#)

# mpiSequenceMemorySet

## Declaration

```
long mpiSequenceMemorySet(MPISequence sequence,  
                           void          *dst,  
                           void          *src,  
                           long          count)
```

**Required Header:** stdmpi.h

## 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

if *SequenceMemorySet* successfully copies ***count*** bytes of application memory to a Sequence object's memory

## 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.

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
|-----------------|----------------------------------|

| Return Values        |                                 |
|----------------------|---------------------------------|
| <b>MPIControl</b>    | a handle to the Sequence object |
| <b>MPIHandleVOID</b> | if <i>sequence</i> 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**

if *SequenceNumber* successfully writes the Sequence's index to the contents of ***number***

## 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 <i>index</i> th Command of a Sequence ( <i>sequence</i> )                                                                                 |
| MPIHandleVOID                   | if <i>sequence</i> is invalid<br>if <i>index</i> is less than 0<br>if <i>index</i> is greater than or equal to <b>mpiSequenceCount(sequence)</b> |
| MPIMessageARG_INVALID           | if <i>index</i> is a negative number.                                                                                                            |
| MEIListMessageELEMENT_NOT_FOUND | if <i>index</i> is greater than or equal to the number of elements in the list.                                                                  |
| MPIMessageHANDLE_INVALID        | if <i>sequence</i> 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***).

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
| <b>command</b>  | a handle to a Command object.    |

| Return Values                   |                                                                              |
|---------------------------------|------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>             | if <i>SequenceCommandAppend</i> successfully appends a Command to a Sequence |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i><b>sequence</b></i> or <i><b>command</b></i> is an invalid handle. |
| <b>MPIMessageNO_MEMORY</b>      | Not enough memory was available.                                             |

## 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 ( <i>sequence</i> ) |
| -1                 | if <i>sequence</i> is invalid     |
| 0                  | if <i>sequence</i> 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 conjunction with `mpiSequenceCommandNext()` in order to iterate through the list.

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
|-----------------|----------------------------------|

| Return Values                   |                                                              |
|---------------------------------|--------------------------------------------------------------|
| <b>handle</b>                   | to the first Command in a Sequence ( <i>sequence</i> )       |
| <b>MPIHandleVOID</b>            | if <i>sequence</i> is invalid<br>if <i>sequence</i> is empty |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>sequence</i> is an invalid handle.                     |

## 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 ( <i>command</i> ) in a Sequence ( <i>sequence</i> )                                                    |
| -1            | if <i>sequence</i> is invalid<br>if the Command ( <i>command</i> ) was not found in the Sequence ( <i>sequence</i> ) |

## 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**

if *SequenceCommandInsert* successfully inserts the Command (*insert*) in a Sequence following the specified Command (*command*)

## 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 conjunction with `mpiSequenceCommandPrevious()` in order to iterate through the list backwards.

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
|-----------------|----------------------------------|

| Return Values                   |                                                              |
|---------------------------------|--------------------------------------------------------------|
| <b>handle</b>                   | to the last Command in a Sequence ( <i>sequence</i> )        |
| <b>MPIHandleVOID</b>            | if <i>sequence</i> is invalid<br>if <i>sequence</i> is empty |
| <b>MPIMessageHANDLE_INVALID</b> | if <i>sequence</i> is an invalid handle.                     |

## 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

|                     |                                                                                       |
|---------------------|---------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SequenceCommandListGet</i> successfully gets the list of Commands in a Sequence |
|---------------------|---------------------------------------------------------------------------------------|

## 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**

if *SequenceCommandListGet* successfully creates a Sequence of Commands using the Command handles specified by ***commandList***

## 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 conjunction with `mpiSequenceCommandFirst()` in order to iterate through the list.

|                 |                                  |
|-----------------|----------------------------------|
| <b>sequence</b> | a handle to the Sequence object. |
| <b>command</b>  | a handle to a Command object.    |

| Return Values                   |                                                                                                          |
|---------------------------------|----------------------------------------------------------------------------------------------------------|
| <b>handle</b>                   | to the Command following the Command ( <i>command</i> ) in a Sequence ( <i>sequence</i> )                |
| <b>MPIHandleVOID</b>            | if <i>sequence</i> is invalid<br>if <i>command</i> is the last command in a Sequence ( <i>sequence</i> ) |
| <b>MPIMessageHANDLE_INVALID</b> | Either <i>sequence</i> or <i>command</i> is an invalid handle.                                           |

## 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 conjunction 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 ( <i>command</i> ) in a Sequence ( <i>sequence</i> )                 |
| MPIHandleVOID            | if <i>sequence</i> is invalid<br>if <i>command</i> is the first command in a Sequence ( <i>sequence</i> ) |
| MPIMessageHANDLE_INVALID | Either <i>sequence</i> or <i>command</i> is an invalid handle.                                            |

## 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**

if *SequenceCommandRemove* successfully removes the Command from a Sequence

## 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.

|                                |                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPISequenceStateSTOPPED</b> | 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 <code>mpiSequenceStop</code> will stop the sequencer, and the sequencer state will be <b>MPISequenceStateSTOPPED</b> . |
| <b>MPISequenceStateSTARTED</b> | 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 <code>mpiSequenceStart</code> .                                                                                                                                |

## See Also

# MPISequenceStatus

## Definition

```
typedef struct MPISequenceStatus {  
    MPICommand          command;  
    MPISequenceState    state;  
} MPISequenceStatus;
```

## Description

**MPISequenceStatus** is a status structure for MPISequence objects.

|                |                                               |
|----------------|-----------------------------------------------|
| <b>command</b> | The current command of the MPISequence object |
| <b>state</b>   | 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

[serviceCreate](#)

Create a Service for EventMgr and the threads necessary for it to run.

[serviceDelete](#)

Stop all threads belonging to the Service and deletes the Service.

### Configuration and Information Methods

[serviceEnable](#)

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.         |
| >0               | <i>ServiceCreate</i> will attempt to assign the priority to all of the service's threads. |

| If "sleep" is | Then                                                                                                                                   |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------|
| -1            | <i>ServiceCreate</i> will attempt to create interrupt driven threads.                                                                  |
| 0             | <i>ServiceCreate</i> will create threads that call <b>mpiEventMgrService(eventMgr,...)</b> as quickly as possible.                     |
| >0            | <i>ServiceCreate</i> will create threads that attempt to call <b>mpiEventMgrService(eventMgr,...)</b> every <b>sleep</b> milliseconds. |

| Return Values        |                                     |
|----------------------|-------------------------------------|
| <b>handle</b>        | to a Service object                 |
| <b>MPIHandleVOID</b> | 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**

if *serviceDelete* successfully deletes a Service object and invalidates its handle

## 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.

| <i>If "enabled" is</i> | Then                                                      |
|------------------------|-----------------------------------------------------------|
| False                  | <i>serviceEnable</i> will disable <b><i>service</i></b> . |
| True                   | <i>serviceEnable</i> will enable <b><i>service</i></b> .  |

### Return Values

|                      |                                     |
|----------------------|-------------------------------------|
| <b>handle</b>        | to a Service object                 |
| <b>MPIHandleVOID</b> | 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.

## Methods

### Create, Delete, Validate Methods

[meiSqNodeCreate](#)

[meiSqNodeDelete](#)

[meiSqNodeValidate](#)

### Configuration and Information Methods

[meiSqNodeCommand](#)

[meiSqNodeConfigGet](#)

[meiSqNodeConfigSet](#)

[meiSqNodeFlashConfigGet](#)

[meiSqNodeFlashConfigSet](#)

[meiSqNodeFpgaDefaultFileName](#)

[meiSqNodeInfo](#)

[meiSqNodeStatus](#)

[meiSqNodeUserDataGet](#)

[meiSqNodeUserDataSet](#)

### Drive Interface Methods

[meiSqNodeDriveConfigGet](#)

[meiSqNodeDriveConfigSet](#)

[meiSqNodeDriveInfo](#)

[meiSqNodeDriveMapParamCount](#)

[meiSqNodeDriveMapParamList](#)

[meiSqNodeDriveMapConfigCount](#)

[meiSqNodeDriveMapConfigList](#)

[meiSqNodeDriveMapParamFileGet](#)

[meiSqNodeDriveMapParamFileSet](#)

[meiSqNodeDriveMonitor](#)

[meiSqNodeDriveMonitorConfigGet](#)

[meiSqNodeDriveMonitorConfigSet](#)

[meiSqNodeDriveParamCalculate](#)

[meiSqNodeDriveParamClear](#)

[meiSqNodeDriveParamGet](#)

[meiSqNodeDriveParamListGet](#)

[meiSqNodeDriveParamListSet](#)

[meiSqNodeDriveParamReload](#)

[meiSqNodeDriveParamRestore](#)

[meiSqNodeDriveParamSet](#)

[meiSqNodeDriveParamStore](#)

## I/O Methods

[meiSqNodeAnalogIn](#)

[meiSqNodeAnalogOutGet](#)

[meiSqNodeAnalogOutSet](#)

[meiSqNodeDigitalIn](#)

[meiSqNodeDigitalOutGet](#)

[meiSqNodeDigitalOutSet](#)

[meiSqNodeSegmentAnalogIn](#)

[meiSqNodeSegmentAnalogOutGet](#)

[meiSqNodeSegmentAnalogOutSet](#)

[meiSqNodeSegmentInfo](#)

[meiSqNodeSegmentDigitalIn](#)

[meiSqNodeSegmentDigitalOutGet](#)

[meiSqNodeSegmentDigitalOutSet](#)

[meiSqNodeSegmentUserDataGet](#)

[meiSqNodeSegmentUserDataSet](#)

## Action Methods

[meiSqNodeDownload](#)

[meiSqNodeFlashErase](#)

[meiSqNodeFpgaFileNameVerify](#)

[meiSqNodeStatusClear](#)

[meiSqNodeVerify](#)

## Event Methods

[meiSqNodeEventNotifyGet](#)

[meiSqNodeEventNotifySet](#)

[meiSqNodeEventReset](#)

## Memory Methods

[meiSqNodeMemory](#)

[meiSqNodeMemoryGet](#)

[meiSqNodeMemorySet](#)

## Relational Methods

[meiSqNodeControl](#)

[meiSqNodeNumber](#)

## Data Types

[MEISqNodeCallback](#)

[MEISqNodeChannel](#)

[MEISqNodeCmdHeader](#)

[MEISqNodeCmdType](#)

[MEISqNodeCommand](#)

[MEISqNodeConfig](#)

[MEISqNodeConfigIoAbort](#)

[MEISqNodeConfigAlarm](#)

[MEISqNodeConfigPacketError](#)

[MEISqNodeConfigTrigger](#)

[MEISqNodeConfigUserFault](#)

[MEISqNodeDataSize](#)

[MEISqNodeDownloadParams](#)

[MEISqNodeDriveInfo](#)

[MEISqNodeDriveMonitor](#)

[MEISqNodeDriveMonitorConfig](#)

[MEISqNodeDriveMonitorData](#)

[MEISqNodeDriveMonitorDataType](#)

[MEISqNodeDriveParamCallback](#)

[MEISqNodeDriveParamCallbackType](#)

[MEISqNodeFeedbackSecondary](#)

[MEISqNodeFileName](#)

[MEISqNodeFpgaType](#)

[MEISqNodeInfo](#)[MEISqNodeInfoId](#)[MEISqNodeInfoIo](#)[MEISqNodeInfoFpga](#)[MEISqNodeInfoNetwork](#)[MEISqNodeMemory](#)[MEISqNodeMessage](#)[MEISqNodeMonitorValue](#)[MEISqNodeMonitorValueIndex](#)[MEISqNodeResponse](#)[MEISqNodeSegmentInfo](#)[MEISqNodeSegmentUserData](#)[MEISqNodeStatus](#)[MEISqNodeStatusCrcError](#)[MEISqNodeStatusPacketError](#)[MEISqNodeUserData](#)

## Constants

[MEISqNodeID\\_CHAR\\_MAX](#)[MEISqNodeFILENAME\\_MAX](#)[MEISqNodeManufacturerDATA\\_CHAR\\_MAX](#)[MEISqNodeMaxFEEDBACK\\_SECONDARY](#)[MEISqNodeMaxMOTORS](#)[MEISqNodeNOT\\_AVAILABLE](#)[MEISqNodeSTATUS\\_NOT\\_AVAILABLE](#)[MEISqNodeUserData\\_CHAR\\_MAX](#)[MEISqNodeSegmentInfoMANUFACTURER\\_LENGTH](#)[MEISqNodeSegmentInfoMODEL\\_NAME\\_LENGTH](#)[MEISqNodeSegmentInfoSERIAL\\_NUMBER\\_LENGTH](#)[MEISqNodeSegmentUserData\\_CHAR\\_MAX](#)[MEIDriveMapParamMAX\\_STRING\\_LENGTH](#)[MEIFPGARINCONREV](#)[MEIFpgaSqMACVersionDEFAULT](#)[MEIFpgaSqMACVersionMIN](#)[MEIFpgaSqMACVersionMAX](#)[MEIFpgaSqNodeVersionDEFAULT](#)

[MEIFpgaSqNodeVersionMIN](#)

[MEIFpgaSqNodeVersionMAX](#)

# meiSynqNetCreate

## 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.

|                |                                                                           |
|----------------|---------------------------------------------------------------------------|
| <b>control</b> | a handle to a Control object                                              |
| <b>number</b>  | an index to the SqNode. The first node number is 0, the second is 1, etc. |

### Return Values

|                      |                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------|
| <b>handle</b>        | to a SqNode object. After creating a SqNode object it must be validated using meiSqNodeValidate(). |
| <b>MPIHandleVOID</b> | 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.

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <b>node</b> | a handle of the SqNode object to delete in the reverse order to avoid memory leaks. |
|-------------|-------------------------------------------------------------------------------------|

### Return Values

|                     |                                                         |
|---------------------|---------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDelete</i> successfully deleted the object. |
|---------------------|---------------------------------------------------------|

## 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.

|             |                                    |
|-------------|------------------------------------|
| <b>node</b> | a handle to a SynqNet node object. |
|-------------|------------------------------------|

|                      |  |
|----------------------|--|
| <b>Return Values</b> |  |
|----------------------|--|

|                     |                                                 |
|---------------------|-------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNode</i> is a handle to a valid object. |
|---------------------|-------------------------------------------------|

## See Also

[meiSqNodeCreate](#) | [meiSqNodeDelete](#)

# meiSynqNetCommand

## 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.

|                  |                                                |
|------------------|------------------------------------------------|
| <b>node</b>      | a handle to a SynqNet node object              |
| <b>*command</b>  | a pointer to a SynqNet node command structure  |
| <b>*response</b> | a pointer to a SynqNet node response structure |

| Return Values                            |                                                                                       |
|------------------------------------------|---------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                      | if <i>SqNodeCommand</i> successfully sends a service command and receives a response. |
| <b>MPIMessageARG_INVALID</b>             | if the command pointer is NULL or the response pointer is NULL                        |
| <b>MEISynqNetMessageRESPONSE_TIMEOUT</b> | if the node does not respond to the service command.                                  |
| <b>MEISynqNetMessageREADY_TIMEOUT</b>    | if the node is busy and does not acknowledge the service command.                     |

## 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***.

|                |                                               |
|----------------|-----------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.            |
| <b>*config</b> | a pointer to a SynqNet node config structure. |

### Return Values

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeConfigGet</i> successfully reads the node configuration. |
|---------------------|----------------------------------------------------------------------|

## 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***.

|                |                                              |
|----------------|----------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object            |
| <b>*config</b> | a pointer to a SynqNet node config structure |

| Return Values                               |                                                                                                                           |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                         | if <i>SqNodeConfigGet</i> successfully writes the node configuration.                                                     |
| <b>MEIMessageARG_INVALID</b>                | if the upStreamError or downStreamError fault/fail limits are less than 0 or greater than 255.                            |
| <b>MEISqNodeMessageFEEDBACK_MAP_INVALID</b> | if secondary encoder (n) is not mappable to the motor on the node specified by MEISqNodeFeedbackSecondary[n].motorIndex.. |

## 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***.

|                |                                                                                                                                                                        |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                                                                                                                     |
| <b>*flash</b>  | <i><b>flash</b></i> is either an MEIFlash handle or MPIHandleVOID. If <i><b>flash</b></i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |
| <b>*config</b> | a pointer to a SynqNet node config structure.                                                                                                                          |

| Return Values       |                                                                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeFlashConfigGet</i> successfully reads a SynqNet node's flash configuration and writes it into the structure pointed to by <i>config</i> . |

## 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\(.\)](#).

|                |                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>*flash</b>  | <p><i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.</p> <p>If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use <a href="#">meiFlashMemoryFromFileType(...)</a> to prompt the actual write to <i>flash</i>.</p> |
| <b>*config</b> | a pointer to a SynqNet node config structure.                                                                                                                                                                                                                                                                                                                                                                               |

| Return Values                               |                                                                                                                                                                 |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                         | if <i>SqNodeFlashConfigSet</i> successfully sets a SynqNet Node ( <i>node</i> ) flash configuration using data from the structure pointed to by <i>config</i> . |
| <b>MEISqNodeMessageFEEDBACK_MAP_INVAILD</b> | given secondary encoder ( <i>n</i> ) is not mappable to the motor on the node specified by MEISqNodeFeedbackSecondary[n].motorIndex.                            |
| <b>MPIMessageARG_INVALID</b>                | If the upStreamError or downStreamError fault/fail limits are less than 0 or greater than 255.                                                                  |

## **MEIFlashMessageNETWORK\_TOPOLOGY\_ERROR**

if a valid flash handle was supplied and SynqNet topology had not yet been saved to flash using meiSynqNetFlashTopologySave(.).

## **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.

|                  |                                                                             |
|------------------|-----------------------------------------------------------------------------|
| <b>sqNode</b>    | handle to a SqNode object.                                                  |
| <b>*fileName</b> | a pointer to a structure that has space allocated to hold an FPGA filename. |

| Return Values                |                                                                                          |
|------------------------------|------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SqNodeFpgaDefaultFileName</i> successfully returns the name of an FPGA image file. |
| <b>MPIMessageARG_INVALID</b> | if the pointer to filename is NULL                                                       |

## 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.

|              |                                                      |
|--------------|------------------------------------------------------|
| <b>node</b>  | a handle to a SynqNet node object.                   |
| <b>*info</b> | a pointer to a drive specific information structure. |

### Return Values

|                              |                                                               |
|------------------------------|---------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SqNodeInfo</i> successfully reads the node information. |
| <b>MPIMessageARG_INVALID</b> | if the info pointer is NULL.                                  |

## 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.

|                |                                        |
|----------------|----------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.     |
| <b>*status</b> | pointer to a SynqNet status structure. |

| Return Values                |                                                                                              |
|------------------------------|----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SynqNetStatus</i> successfully reads the node status and writes it into the structure. |
| <b>MPIMessageARG_INVALID</b> | if the status pointer is NULL.                                                               |

## 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.

|              |                                                                    |
|--------------|--------------------------------------------------------------------|
| <b>node</b>  | a handle to a SynqNet node object.                                 |
| <b>*data</b> | a pointer to a MEISqNodeUserData structure, allocated on the host. |

| Return Values       |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeUserDataGet</i> successfully gets the user data from the node. |

## See Also

[meiSqNodeUserDataSet](#)

# meiSqNodeUserDataSet

## Declaration

```
long meiSqNodeUserDataSet(MEISqNode node,  
                           MEISqNodeUserData *data);
```

Required Header: stdmei.h

## Description

**meiSqNodeUserDataSet** writes the user data to the node.

|              |                                                                    |
|--------------|--------------------------------------------------------------------|
| <b>node</b>  | a handle to a SynqNet node object.                                 |
| <b>*data</b> | a pointer to a MEISqNodeUserData structure, allocated on the host. |

| Return Values       |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeUserDataSet</i> successfully sets the user data to the node. |

## See Also

[meiSqNodeUserDataGet](#) | [MEISqNodeUserData](#)

# 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.

|                   |                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object                                                                      |
| <b>driveIndex</b> | an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc. |
| <b>*config</b>    | a pointer to a drive specific configuration structure.                                                 |

| Return Values       |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveConfigGet</i> successfully reads the drive configuration. |

## 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.

|                   |                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object                                                                      |
| <b>driveIndex</b> | an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc. |
| <b>*config</b>    | a pointer to a drive specific configuration structure.                                                 |

| Return Values       |                                                                             |
|---------------------|-----------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveConfigSet</i> successfully writes the drive configuration. |

## 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.

|                   |                                                                                                                                  |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object.                                                                                               |
| <b>driveIndex</b> | an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.                           |
| <b>*info</b>      | a pointer to a structure that contains general drive information.                                                                |
| <b>*external</b>  | a pointer to a drive specific information structure. See the appropriate drive vendor *.h for definition. (NOTE: it can be NULL) |

| Return Values       |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveInfo</i> successfully reads the drive information. |

## 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.

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| <b>sqNode</b>       | a handle to a SynqNet node object.                          |
| <b>driveMap</b>     | a handle to a DriveMap object.                              |
| <b>driveIndex</b>   | an index to the drive (0, 1, 2, etc), relative to the node. |
| <b>*paramsCount</b> | pointer to the variable that will be set by this function.  |

| Return Values       |                                                                                                                                                                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSqNodeDriveMapParamCount</i> successfully scans the drive map file for a drive entry that matches this node on the network and returns the number of drive parameters that need to be preserved for the configuration of the drive. |

## 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.

|                        |                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------|
| <b>sqNode</b>          | a handle to a SynqNet node object.                                                                       |
| <b>driveMap</b>        | a handle to a DriveMap object.                                                                           |
| <b>driveIndex</b>      | an index to the drive (0, 1, 2, etc), relative to the node.                                              |
| <b>paramsCount</b>     | the number of drive parameter information records that can be written to the <i>driveParamInfo</i> list. |
| <b>*driveParamInfo</b> | pointer to the list of drive parameter information records that will be filled in by this function.      |

| Return Values       |                                                                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSqNodeDriveMapParamList</i> successfully scans the drive map file for an entry that matches the node on the network and writes the information about each of the drive parameters to the <i>driveParamInfo</i> list. |

## 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.

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| <b>sqNode</b>       | a handle to a SynqNet node object.                          |
| <b>driveMap</b>     | a handle to a DriveMap object.                              |
| <b>driveIndex</b>   | an index to the drive (0, 1, 2, etc), relative to the node. |
| <b>*configCount</b> | pointer to the variable that will be set by this function.  |

| Return Values       |                                                                                                                                                                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSqNodeDriveMapConfigCount</i> successfully scans the drive map file for a drive entry that matches this node on the network and returns the number of drive parameters that need to be preserved for the configuration of the drive. |

## 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.

|                    |                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>sqNode</b>      | a handle to a SynqNet node object.                                                                                    |
| <b>driveMap</b>    | a handle to a DriveMap object.                                                                                        |
| <b>driveIndex</b>  | an index to the drive (0, 1, 2, etc), relative to the node.                                                           |
| <b>configCount</b> | the number of drive parameter information records that can be written to the <i>configList</i> list.                  |
| <b>*configList</b> | pointer to the list of drive parameters that make up the drive configuration that will be filled in by this function. |

| Return Values       |                                                                                                                                                                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSqNodeDriveMapConfigList</i> successfully scans the drive map file for a drive entry that matches this node on the network and returns the list of drive parameters that need to be preserved for the configuration of the drive. |

## 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***.

|                             |                                                                                                                                                                                                                                         |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sqNode</b>               | a handle to the SynqNet node object.                                                                                                                                                                                                    |
| <b>driveMap</b>             | a handle to the DriveMap object.                                                                                                                                                                                                        |
| <b>driveIndex</b>           | an index to the drive (0, 1, 2, etc), relative to the node.                                                                                                                                                                             |
| <b>*driveConfigFilename</b> | the name of the file that holds the stored drive configuration file.                                                                                                                                                                    |
| <b>append</b>               | <div>1 = The new data is appended to the existing drive configuration file if it exists.</div> <div>0 = A new drive configuration file is created to hold the drive parameters. If a file already exists, it will be overwritten.</div> |

| Return Values       |                                                                                                                                                     |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveMapParamFileGet</i> successfully saves the current set of drive parameters in the drive to the <i><b>driveConfigFilename</b></i> . |

## See Also

[meiSqNodeDriveMapParamFileSet](#)

# 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.

|                             |                                                                                                                                                                                                                                                          |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sqNode</b>               | a handle to the SynqNet node object.                                                                                                                                                                                                                     |
| <b>driveMap</b>             | a handle to the DriveMap object.                                                                                                                                                                                                                         |
| <b>driveIndex</b>           | an index to the drive (0, 1, 2, etc), relative to the node.                                                                                                                                                                                              |
| <b>*driveConfigFilename</b> | the name of the file that holds the stored drive configuration file.                                                                                                                                                                                     |
| <b>callback</b>             | 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.                 |
| <b>warning</b>              | 0 = if setting a drive parameter fails, this function will fail immediately.<br><br>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       |                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveMapParamFileSet</i> successfully loads the drive parameters stored in the file named <i>driveConfigFilename</i> into the drive. |

## See Also

[meiSqNodeDriveMapParamFileGet](#)

# 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***.

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object.                          |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node. |
| <b>*value</b>     | pointer to a structure of monitor values                    |

| Return Values       |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveMonitor</i> successfully reads the monitor information. |

## See Also

[MEISqNodeMonitorValue](#)

# 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.

|                   |                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object.                                                                     |
| <b>driveIndex</b> | an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc. |
| <b>*config</b>    | a pointer to a drive monitor configuration structure.                                                  |

| Return Values       |                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveMonitorConfigGet</i> successfully reads the drive monitor configuration. |

## 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.

|                   |                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object.                                                                     |
| <b>driveIndex</b> | an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc. |
| <b>*config</b>    | a pointer to a drive monitor configuration structure.                                                  |

| Return Values       |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveMonitorConfigSet</i> successfully writes the drive monitor configuration. |

## 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.

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <b>sqNode</b>     | a handle to a SynqNet node object                           |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node. |

| Return Values       |                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamCalculate</i> successfully has the drive to calculate its internal quantities. |

## 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.

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <b>sqNode</b>     | a handle to a SynqNet node object                           |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node. |

| Return Values       |                                                                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamClear</i> successfully clears the previously saved drive by loading the default set of drive parameters into the current and non-volatile storage on the drive. |

## 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.

|                   |                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to the SynqNet node object.                                                             |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node.                                      |
| <b>param</b>      | an index for the drive parameter that is being accessed.                                         |
| <b>paramType</b>  | the type of the data read from the drive and which field will be used in the <i>value</i> union. |
| <b>*value</b>     | a pointer to the union that will be filled in.                                                   |

| Return Values       |                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamGet</i> successfully reads a drive parameter from the drive and fills in the appropriate field of the union pointed to by <i>value</i> . |

## 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  | if <i>SqNodeDriveParamListGet</i> successfully reads a series of drive parameters from the drive and fills in the appropriate fields of the unions pointed to by <i>paramValues</i> . |

## 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.

|                     |                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>         | a handle to the SynqNet node object.                                                                                                  |
| <b>driveIndex</b>   | an index to the drive (0, 1, 2, etc), relative to the node.                                                                           |
| <b>size</b>         | the number of drive parameters to be written.                                                                                         |
| <b>*paramList</b>   | 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. |
| <b>*paramTypes</b>  | a pointer to a list of the drive parameter indexes that are being accessed.                                                           |
| <b>*paramValues</b> | a pointer to a list of unions that will be written to the drive.                                                                      |

| Return Values       |                                                                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamListSet</i> successfully writes a series of drive parameters pointed to by <b>value</b> to the drive. |

## 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.

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <b>sqNode</b>     | a handle to a SynqNet node object                           |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node. |

| Return Values       |                                                                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamReload</i> successfully overwrites the current set of drive parameters with the set from the non-volatile storage on the drive. |

## 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.

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <b>sqNode</b>     | a handle to a SynqNet node object                           |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node. |

| Return Values       |                                                                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamRestore</i> successfully loads the default set of drive parameters into current set of drive parameters on the drive. |

## 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.

|                   |                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to the SynqNet node object.                                                                |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node.                                         |
| <b>param</b>      | an index for the drive parameter that is being accessed.                                            |
| <b>paramType</b>  | the type of data being written to the drive and which field will be used in the <i>value</i> union. |
| <b>*value</b>     | pointer to the union that will be written to the drive.                                             |

| Return Values       |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamSet</i> successfully writes the drive parameter that is pointed to by <b>value</b> to the drive. |

## 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.

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <b>sqNode</b>     | a handle to a SynqNet node object                           |
| <b>driveIndex</b> | an index to the drive (0, 1, 2, etc), relative to the node. |

| Return Values       |                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDriveParamStore</i> successfully saves the drive parameters into non-volatile storage on the drive. |

## 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

## Description

**meiSqNodeAnalogIn** gets the current state of an analog input on a SynqNet node.

|                |                                                                                |
|----------------|--------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                             |
| <b>channel</b> | the index of the analog input channel (with respect to the node).              |
| <b>*state</b>  | a pointer to where the current state of the input is written by this function. |

| Return Values       |                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeAnalogIn</i> successfully gets the current state of an analog input on a SynqNet node. |

## See Also

# 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.

|                |                                                                                 |
|----------------|---------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                              |
| <b>channel</b> | the index of the analog output channel (with respect to the node).              |
| <b>*state</b>  | a pointer to where the current state of the output is written by this function. |

| Return Values       |                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeAnalogOutGet</i> successfully reads the current state of an analog output on a SynqNet node. |

## See Also

[meiSqNodeAnalogSet](#)

# meiSqNodeAnalogOutSet

## Declaration

```
long meiSqNodeAnalogOutSet(MEISqNode node,  
                           long channel,  
                           long state,  
                           long wait);
```

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

## Description

**meiSqNodeAnalogOutSet** changes the current state of an analog output on a SynqNet node.

|                |                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                                                                                        |
| <b>channel</b> | the index of the analog output channel (with respect to the node).                                                                        |
| <b>state</b>   | the desired state of the analogue output.                                                                                                 |
| <b>wait</b>    | determines what happens if two output functions are called in short succession. See <a href="#">Overview of Motor I/O: Output Waits</a> . |

| Return Values       |                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeAnalogOutSet</i> successfully changes the current state of an analog output on a SynqNet node. |

## See Also

[meiSqNodeAnalogOutGet](#)

# 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.

|                 |                                                                                        |
|-----------------|----------------------------------------------------------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object.                                                     |
| <b>bitStart</b> | the first bit.                                                                         |
| <b>bitCount</b> | the number of bits to be read.                                                         |
| <b>*state</b>   | a pointer to where the current state of the input bits is written to by this function. |

| Return Values       |                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeDigitalIn</i> successfully gets the current state of multiple digital inputs on the specified SynqNet node. |

## See Also

# 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.

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object.                                                      |
| <b>bitStart</b> | the first bit.                                                                          |
| <b>bitCount</b> | the number of bits to be read.                                                          |
| <b>*state</b>   | a pointer to where the current state of the output bits is written to by this function. |

| Return Values       |                                                                                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeAnalogOutGet</i> successfully reads the current state of the multiple digital output bits on the specified SynqNet node. |

## See Also

[meiSqNodeDigitalOutSet](#)

# meiSqNodeDigitalOutSet

## Declaration

```
long meiSqNodeDigitalOutSet(MEISqNode node,
                             long bitStart,
                             long bitCount,
                             unsigned long state,
                             long wait);
```

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

## Description

**meiSqNodeDigitalOutSet** changes the state of multiple digital outputs on the specified SynqNet node.

|                 |                                                                                                                                                                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object.                                                                                                                                                                                                                                                 |
| <b>bitStart</b> | the first bit.                                                                                                                                                                                                                                                                     |
| <b>bitCount</b> | the number of bits to be changed.                                                                                                                                                                                                                                                  |
| <b>state</b>    | a pointer to where the current state of the output bits is written to by this function.                                                                                                                                                                                            |
| <b>wait</b>     | 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       |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeAnalogOutSet</i> successfully changes the state of multiple digital outputs on the specified SynqNet node. |

## See Also

[meiSqNodeDigitalOutGet](#)



# 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.

|                |                                                                                |
|----------------|--------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                             |
| <b>segment</b> | the index of the slice/module attached to this SynqNet node.                   |
| <b>channel</b> | the index of the analog input channel (with respect to the slice).             |
| <b>*state</b>  | a pointer to where the current state of the input is written by this function. |

| Return Values       |                                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentAnalogIn</i> successfully gets the current state of an analog input on the specified slice on a SynqNet node. |

## See Also

# 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.

|                |                                                                                 |
|----------------|---------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                              |
| <b>segment</b> | the index of the slice/module attached to this SynqNet node.                    |
| <b>channel</b> | the index of the analog output channel (with respect to the slice).             |
| <b>*state</b>  | a pointer to where the current state of the output is written by this function. |

| Return Values       |                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentAnalogOutGet</i> successfully gets the current state of an analog output on the specified slice on a SynqNet node. |

## See Also

[meiSqNodeSegmentAnalogOutSet](#)

# meiSqNodeSegmentAnalogOutSet

## Declaration

```
long meiSqNodeSegmentAnalogOutSet(MEISqNode node,  
                                   long segment,  
                                   long channel,  
                                   long state,  
                                   long wait);
```

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

## Description

**meiSqNodeSegmentAnalogOutSet** changes the current state of an analog output on the specified slice on a SynqNet node.

|                |                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                                                                                                        |
| <b>segment</b> | the index of the slice/module attached to this SynqNet node.                                                                              |
| <b>channel</b> | the index of the analog input channel (with respect to the slice).                                                                        |
| <b>state</b>   | the desired state of the analog output.                                                                                                   |
| <b>wait</b>    | determines what happens if two output functions are called in short succession. See <a href="#">Overview of Motor I/O: Output Waits</a> . |

| Return Values       |                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentAnalogOutSet</i> successfully changes the current state of an analog output on the specified slice on a SynqNet node. |

## See Also

[meiSqNodeSegmentAnalogOutGet](#)

# 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.

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object                                 |
| <b>segment</b> | the index of the slice / module attached to this SynqNet node.    |
| <b>*info</b>   | a pointer to a structure that will be filled in by this function. |

### Return Values

#### MPIMessageOK

if *SqNodeSegmentInfo* successfully reads the constant data about a segment on a SynqNet node and fills in the structure pointer by the ***info*** argument.

## See Also

[MEISqNodeSegmentInfo](#)

# 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.

|                 |                                                                   |
|-----------------|-------------------------------------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object.                                |
| <b>segment</b>  | the index of the slice/module attached to this SynqNet node.      |
| <b>bitStart</b> | the first bit                                                     |
| <b>bitCount</b> | the number of bits to be read.                                    |
| <b>*state</b>   | a pointer to a long word that will be filled in by this function. |

| Return Values       |                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentDigitalIn</i> successfully gets the current state of multiple digital inputs on the specified slice on a SynqNet node. |

## See Also

# 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.

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object.                                                      |
| <b>segment</b>  | the index of the slice/module attached to this SynqNet node.                            |
| <b>bitStart</b> | the first bit                                                                           |
| <b>bitCount</b> | the number of bits to be read.                                                          |
| <b>state</b>    | a pointer to where the current state of the output bits is written to by this function. |

| Return Values       |                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentDigitalOutGet</i> successfully changes the state of multiple digital outputs on the specified slice on a SynqNet node. |

## See Also

[meiSqNodeSegmentDigitalOutSet](#)

# meiSqNodeSegmentDigitalOutSet

## Declaration

```
long meiSqNodeSegmentDigitalOutSet(MEISqNode node,  
                                   long segment,  
                                   long bitStart,  
                                   long bitCount,  
                                   unsigned long state,  
                                   long wait);
```

**Required Header:** stdmei.h  
**Change History:** 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.

|                 |                                                                                                                                                                                                                                                        |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object.                                                                                                                                                                                                                     |
| <b>segment</b>  | the index of the slice/module attached to this SynqNet node.                                                                                                                                                                                           |
| <b>bitStart</b> | the first bit                                                                                                                                                                                                                                          |
| <b>bitCount</b> | the number of bits to be read.                                                                                                                                                                                                                         |
| <b>state</b>    | a pointer to where the current state of the output bits is written to by this function.                                                                                                                                                                |
| <b>wait</b>     | <div>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.<br/><br/>You should be able to use TRUE for this argument in most applications.</div> |

| Return Values       |                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentDigitalOutSet</i> successfully gets the current state of the multiple digital output bits on the specified slice/module on a SynqNet node. |

## See Also

[meiSqNodeSegmentDigitalOutGet](#)

# 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.

|                |                                                               |
|----------------|---------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                            |
| <b>segment</b> | the index of the slice/module attached to this SynqNet node.  |
| <b>data</b>    | a pointer to where the user data is written by this function. |

| Return Values       |                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentUserDataGet</i> successfully gets the user data from a segment on a SynqNet node. |

## See Also

[meiSqNodeSegmentUserDataSet](#) | [MPI Overview I/O: User Data](#)



# 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.

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object.                           |
| <b>segment</b> | the index of the slice/module attached to this SynqNet node. |
| <b>data</b>    | a pointer to the new user data.                              |

| Return Values       |                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeSegmentUserDataSet</i> successfully changes the user data stored on a segment on a SynqNet node. |

## See Also

[meiSqNodeSegmentUserDataGet](#) | [MPI Overview I/O: User Data](#)

# 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.

|                |                                                 |
|----------------|-------------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object               |
| <b>*params</b> | a pointer to the download parameters structure. |

| Return Values       |                                                              |
|---------------------|--------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeNumber</i> successfully reads the network number |

## 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.

|             |                                    |
|-------------|------------------------------------|
| <b>node</b> | a handle to a SynqNet node object. |
|-------------|------------------------------------|

| Return Values       |                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeFlashErase</i> successfully 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. |

## See Also

# meiSqNodeFpgaFileNameVerify

## Declaration

```
long meiSqNodeFpgaFileNameVerify(MEISqNode    sqNode,  
                                char*        fileName);
```

Required Header: stdmei.h

## Description

**meiSqNodeFpgaFileNameVerify** verifies that the filename provided is compatible with a given sqNode.

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <b>sqNode</b>   | a handle to an SqNode object.                                      |
| <b>fileName</b> | a pointer to a string containing the name of an SqNode image file. |

| Return Values                             |                                                                     |
|-------------------------------------------|---------------------------------------------------------------------|
| <b>MPIMessageOK</b>                       | If the FPGA image file is compatible with the specified SqNode.     |
| <b>MEISqNodeMessageFILE_NODE_MISMATCH</b> | If the FPGA image file is NOT compatible with the specified SqNode. |

## See Also

# 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.

|             |                                   |
|-------------|-----------------------------------|
| <b>node</b> | a handle to a SynqNet node object |
|-------------|-----------------------------------|

| Return Values       |                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeStatusClear</i> successfully clears node CRC errors on all ports, clears node Packet errors, clears node ioAbort state, and resets SqNode events. |

## 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.

|                |                                                     |
|----------------|-----------------------------------------------------|
| <b>node</b>    | a handle to SqNode object.                          |
| <b>*params</b> | a pointer to parameters used in the verify routine. |

| Return Values                      |                                                                                             |
|------------------------------------|---------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                | If the <i>meisqNodeVerify</i> successfully verifies that there is a matching runtime image. |
| <b>MEISqNodeMessageVERIFY_FAIL</b> | If the file provided does not match the runtime image on the node.                          |

## 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.

|                   |                                                                                        |
|-------------------|----------------------------------------------------------------------------------------|
| <b>node</b>       | a handle to a SynqNet node object                                                      |
| <b>*eventMask</b> | pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations. |
| <b>*external</b>  | pointer to external                                                                    |

| Return Values                |                                                                  |
|------------------------------|------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SqNodeEventNotifyGet</i> successfully reads the event mask |
| <b>MPIMessageARG_INVALID</b> | if the eventMask pointer is NULL                                 |

## 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).

|                  |                                                                                        |
|------------------|----------------------------------------------------------------------------------------|
| <b>node</b>      | a handle to a SynqNet node object                                                      |
| <b>eventMask</b> | pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations. |
| <b>*external</b> | pointer to external                                                                    |

| Return Values                |                                                                    |
|------------------------------|--------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SynqNetEventNotifySet</i> successfully writes the event mask |
| <b>MPIMessageARG_INVALID</b> | if the eventMask pointer is NULL                                   |



## 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          | if <i>SynqNetEventReset</i> successfully writes the event mask |
| MPIMessageARG_INVALID | if the eventMask pointer is NULL                               |

## See Also

# 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(.).

|                 |                                        |
|-----------------|----------------------------------------|
| <b>node</b>     | a handle to a SynqNet node object      |
| <b>**memory</b> | a pointer to an SqNode memory address. |

| Return Values       |                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeMemory</i> successfully writes the SqNode memory address to the contents of memory. |

## See Also

[meiSqNodeMemoryGet](#) | [meiSqNodeMemorySet](#)

# meiSqNodeMemoryGet

## Declaration

```
long meiSqNodeMemoryGet(MEISqNode    node,
                        void          *dst,
                        void          *src,
                        long          count);
```

**Required Header:** stdmei.h

## Description

**meiSqNodeMemoryGet** reads count bytes of an SqNode's memory, starting from address **src** and writes it to application memory, starting at address **dst**.

|              |                                                          |
|--------------|----------------------------------------------------------|
| <b>node</b>  | a handle to a SynqNet node object                        |
| <b>*dst</b>  | pointer to the destination address in application memory |
| <b>*src</b>  | pointer to the source address in SqNode memory           |
| <b>count</b> | number of bytes to copy                                  |

### Return Values

#### MPIMessageOK

if *SqNodeMemoryGet* successfully copies data from SqNode memory to application memory.

## See Also

[meiSqNodeMemory](#) | [meiSqNodeMemorySet](#)

# meiSqNodeMemorySet

## Declaration

```
long meiSqNodeMemorySet(MEISqNode node,
                        void *dst,
                        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*.

|              |                                                     |
|--------------|-----------------------------------------------------|
| <b>node</b>  | a handle to a SynqNet node object                   |
| <b>*dst</b>  | pointer to the destination address in SqNode memory |
| <b>*src</b>  | pointer to the source address in application memory |
| <b>count</b> | number of bytes to copy                             |

| Return Values       |                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeMemorySet</i> successfully copies data from application memory to SqNode memory. |

## 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.

|             |                                   |
|-------------|-----------------------------------|
| <b>node</b> | a handle to a SynqNet node object |
|-------------|-----------------------------------|

|                      |  |
|----------------------|--|
| <b>Return Values</b> |  |
|----------------------|--|

|                      |                              |
|----------------------|------------------------------|
| <b>MPIControl</b>    | a handle to a control object |
| <b>MPIHandleVOID</b> | 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.).

|                |                                           |
|----------------|-------------------------------------------|
| <b>node</b>    | a handle to a SynqNet node object         |
| <b>*number</b> | a pointer to the index of a SynqNet node. |

| Return Values       |                                                               |
|---------------------|---------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SqNodeNumber</i> successfully reads the network number. |

## See Also

[meiSynqNetInfo](#) | [meiSynqNetNumber](#)

# MEISqNodeCallback

## Definition

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

## Description

**MEISqNodeCallback** is a pointer to a function, which can be used to monitor the `meiSqNodeDownload(...)` progress.

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| <b>percentage</b> | The portion (from 0% to 100%) of memory download that has been completed. |
|-------------------|---------------------------------------------------------------------------|

## See Also

[meiSqNodeDownload](#)



# 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.).

|                               |                              |
|-------------------------------|------------------------------|
| <b>MEISqNodeChannelDRIVE0</b> | interface to drive number 0  |
| <b>MEISqNodeChannelDRIVE1</b> | interface to drive number 1  |
| <b>MEISqNodeChannelDRIVE2</b> | interface to drive number 2  |
| <b>MEISqNodeChannelDRIVE3</b> | interface to drive number 3  |
| <b>MEISqNodeChannelDRIVE4</b> | interface to drive number 4  |
| <b>MEISqNodeChannelDRIVE5</b> | interface to drive number 5  |
| <b>MEISqNodeChannelDRIVE6</b> | interface to drive number 6  |
| <b>MEISqNodeChannelDRIVE7</b> | interface to drive number 7  |
| <b>MEISqNodeChannelNODE</b>   | interface to the node device |

## See Also

[meiSqNodeCommand](#) | [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.

|                |                                                                                    |
|----------------|------------------------------------------------------------------------------------|
| <b>channel</b> | Communication interface to a device. See <a href="#">MEISqNodeChannel</a> .        |
| <b>memory</b>  | The memory region to access. See <a href="#">MEISqNodeMemory</a> .                 |
| <b>size</b>    | The length of data to send or receive. See <a href="#">MEISqNodeDataSize</a> .     |
| <b>type</b>    | The service command action (read or write). See <a href="#">MEISqNodeCmdType</a> . |

## See Also

[meiSqNodeCommand](#) | [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

[meiSqNodeCommand](#) | [MEISqNodeCmdHeader](#)

# MEISqNodeCommand

## Definition

```
typedef struct MEISqNodeCommand {
    MEISqNodeCmdHeader header;
    unsigned long address; /* command registers */
    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 <a href="#">MEISqNodeCmdHeader</a> . |
| address | A memory location to read or write the data.                                                                   |
| data    | The command data to send.                                                                                      |

## See Also

[meiSqNodeCommand](#) | [MEISqNodeResponse](#)

# MEISqNodeConfig

## Definition

```
typedef struct MEISqNodeConfig {
    MEISqNodeConfigAlarm      nodeAlarm;
    MEISqNodeConfigIoAbort    ioAbort;
    MEISqNodeConfigPacketError upStreamError;
    MEISqNodeConfigPacketError downStreamError;
    MEISqNodeConfigUserFault  userFault;
    MEISqNodeFeedbackSecondary feedbackSecondary
                                [MEISqNodeMaxFEEDBACK\_SECONDARY];
} MEISqNodeConfig;
```

## Description

**MEISqNodeConfig** specifies the SynqNet node configurations.

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>nodeAlarm</b>     | 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 <a href="#">MEISqNodeConfigIoAbort</a> and <a href="#">MEISqNodeConfigNodeAlarm</a> for the trigger configurations.                                                                                          |
| <b>ioAbort</b>       | 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). See <a href="#">MEISqNodeConfigIoAbort</a> for the trigger configurations.                                                                                                                                                                                                                                                                           |
| <b>upStreamError</b> | <p>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 <a href="#">MEISqNodeConfigPacketError</a> for appropriate ranges and resulting actions.</p> <p><b>NOTE:</b> 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.</p> |

|                          |                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>downStreamError</b>   | 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 <a href="#">MEISqNodeConfigPacketError</a> for appropriate ranges and resulting actions. |
| <b>userFault</b>         | 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 <a href="#">MEISqNodeConfigUserFault</a> for the trigger configurations.                                                                                                      |
| <b>feedbackSecondary</b> | A structure to configure the secondary encoder resources on the node. See <a href="#">MEISqNodeFeedbackSecondary</a> for more information.                                                                                                                                                                                                                 |

## See Also

[meiSqNodeConfigGet](#) | [meiSqNodeConfigSet](#) | [MEISqNodeConfigPacketError](#)

# MEISqNodeConfigIoAbort

## Definition

```
typedef struct MEISqNodeConfigIoAbort {
    MEISqNodeConfigTrigger    synqLost;      /* communication error */
    MEISqNodeConfigTrigger    nodeDisable;   /* external input */
    MEISqNodeConfigTrigger    powerFault;    /* analog power failure */
    long                      userFault;    /* TRUE = user fault causes ioabort */
} MEISqNodeConfigIoAbort;
```

## 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. 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.

|                    |                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqLost</b>    | Occurs when a SynqNet node drops out of SYNQ (cyclic) mode to SYNQ_LOST mode. See <a href="#">MEISqNodeConfigTrigger</a> .                                                                                                                                                                                                                                                                   |
| <b>nodeDisable</b> | 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 <a href="#">MEISqNodeConfigTrigger</a> .                                                                                                                                                                                                              |
| <b>powerFault</b>  | 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 <a href="#">MEISqNodeConfigTrigger</a> . |
| <b>userFault</b>   | A user configurable trigger condition. A value of TRUE enables the trigger, FALSE disables the trigger.                                                                                                                                                                                                                                                                                      |

## See Also

[meiSqNodeConfigGet](#) | [meiSqNodeConfigSet](#) | [mpiMotionAction](#)

# MEISqNodeConfigAlarm

## Definition

```
typedef struct MEISqNodeConfigAlarm {
    unsigned long    mask; /* One bit per drive/motor. Triggered by
                           the MEIMotorDedicatedInAMP_FAULT input. */
    long             notCyclicEnable; /* allow nodeAlarm to be asserted
                                     when the node is not in cyclic mode */
    long             ioAbortEnable; /* allow ioAbort to assert nodeAlarm */
} MEISqNodeConfigAlarm;
```

## 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 | <p>This Boolean variable is used to specify whether or not a node can receive an alarm when it is not in cyclic mode.</p> <p>TRUE = node alarm can be asserted in any mode.<br/>FALSE = node alarm can only be asserted in cyclic mode.</p> |
| ioAbortEnable   | <p>This Boolean variable is used to specify the effect an I/O abort will have on the node alarm output.</p> <p>TRUE = an I/O abort will trigger a node alarm.<br/>FALSE = an I/O abort will not necessarily trigger a node alarm.</p>       |

## See Also

[meiSqNodeConfigGet](#) | [meiSqNodeConfigSet](#)



# MEISqNodeConfigPacketError

## Definition

```
typedef struct MEISqNodeConfigPacketError {
    long faultLimit;    /* 0 - 255 */
    long failLimit;    /* 0 - 255 */
} MEISqNodeConfigPacketError;
```

## Description

**MEISqNodeConfigPacketError** specifies the limit conditions for SynqNet node packet rate errors.

|                   |                                                                                                                                                                                                                      |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>faultLimit</b> | 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 0 to 255. The value saturates at 255. |
| <b>failLimit</b>  | 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 0 to 255. The value saturates at 255.                |

## See Also

[meiSqNodeConfigGet](#) | [meiSqNodeConfigSet](#)

# MEISqNodeConfigTrigger

## Definition

```
typedef struct MEISqNodeConfigTrigger {  
    long    enable;  
    long    invert;  
} MEISqNodeConfigTrigger;
```

## Description

**MEISqNodeConfigTrigger** specifies trigger configurations.

|               |                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------|
| <b>enable</b> | Enables or disables the trigger. A value of TRUE enables the trigger, FALSE disables the trigger.                  |
| <b>invert</b> | 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.

|                |                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>*addr</b>   | A pointer to a controller address.                                                                                                        |
| <b>mask</b>    | A bit mask ANDed with the value at the controller address.                                                                                |
| <b>pattern</b> | 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.

|                               |                    |
|-------------------------------|--------------------|
| <b>MEISqNodeDataSize8BIT</b>  | 8 bit data length  |
| <b>MEISqNodeDataSize16BIT</b> | 16 bit data length |
| <b>MEISqNodeDataSize24BIT</b> | 24 bit data length |
| <b>MEISqNodeDataSize32BIT</b> | 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.

|                  |                                                                                                                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>*filename</b> | A pointer to a file name. The file contains a header and binary code/data. Files are node/drive specific. Please see the <a href="#">Node Binary Files: Product Table</a> or the drive manufacturer's documentation for the drive binary files. |
| <b>channel</b>   | A communication interface to a node's logic device or drive processor. See <a href="#">MEISqNodeChannel</a> .                                                                                                                                   |
| <b>callback</b>  | A pointer to a callback function, to monitor the download progress. See <a href="#">MEISqNodeCallback</a> .                                                                                                                                     |

## See Also

[meiSqNodeDownload](#)

# MEISqNodeDriveInfo

## Definition

```
typedef struct MEISqNodeDriveInfo {  
    char    firmwareVersion[MEISqNodeDriveParamMAX_STRING_LENGTH];  
} MEISqNodeDriveInfo;
```

## Description

**MEISqNodeDriveInfo** contains information about a specified drive.

| <b>firmwareVersion</b>                                                                                         |
|----------------------------------------------------------------------------------------------------------------|
| 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 <a href="#">MEISqNodeDriveMonitorDataType</a> . |
| data | The location of the drive data. See <a href="#">MEISqNodeDriveMonitorData</a> .             |

## 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.

|                 |                                   |
|-----------------|-----------------------------------|
| <b>monitorA</b> | configuration for drive monitor A |
| <b>monitorB</b> | configuration for drive monitor B |
| <b>monitorC</b> | 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.

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <b>index</b>   | A drive specific value to select a monitor data field from a table. |
| <b>address</b> | A drive specific memory address to select the monitor data.         |

## See Also

[meiSqNodeDriveMonitorConfigGet](#) | [meiSqNodeDriveMonitorConfigSet](#)

# MEISqNodeDriveMonitorDataType

## Definition

```
typedef enum MEISqNodeDriveMonitorDataType {
    MEISqNodeDriveMonitorDataTypeINDEX,
    MEISqNodeDriveMonitorDataTypeADDRESS,
} MEISqNodeDriveMonitorDataType;
```

## 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.          |

## See Also

[meiSqNodeDriveMonitorConfigGet](#) | [meiSqNodeDriveMonitorConfigSet](#)

# MEISqNodeDriveParamCallback

## Definition

```
typedef void (*MEISqNodeDriveParamCallback)
             (MEISqNodeDriveParamCallbackType type,
              char *name,
              long number,
              char *value);
```

## Description

In the **MEISqNodeDriveParamCallback** structure, the function's pointer type defines a function that can be passed to the meiSqNodeDriveParamFileSet function. The meiSqNodeDriveParamFileSet 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

[meiSqNodeDriveParamFileSet](#)

# MEISqNodeDriveParamCallbackType

## Definition

```
typedef enum {
    MEISqNodeDriveParamCallbackTypeCHANGED,
    MEISqNodeDriveParamCallbackTypeSET_FAILED,
} MEISqNodeDriveParamCallbackType;
```

## 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 | Setting this drive parameter failed.                                                           |

## 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.

| <b>motorIndex</b>                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.

|                 |                                                     |
|-----------------|-----------------------------------------------------|
| <b>fileName</b> | 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.

|                                  |                                                                                                                                                                                             |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEISqNodeFpgaTypeBOOT</b>     | The FPGA is operating with a boot image. The boot image only supports basic SynqNet communication. Use <code>meiSqNodeDownload(.)</code> to download the runtime image to the SynqNet node. |
| <b>MEISqNodeFpgaTypeRUN_TIME</b> | 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.

|                               |                                                                                                                          |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>motorCount</b>             | The number of motors that the SynqNet node supports.                                                                     |
| <b>driveCount</b>             | The number of drives interfaces that the SynqNet node supports.                                                          |
| <b>motorOffset</b>            | The starting number for the first motor on the SynqNet node.                                                             |
| <b>feedbackSecondaryCount</b> | The number of auxillary feedbacks on the node.                                                                           |
| <b>id</b>                     | A structure that contains identification data for the SynqNet node. See <a href="#">MEISqNodeInfoId</a> .                |
| <b>fpga</b>                   | A structure that contains identification data for the SynqNet node FPGA. See <a href="#">MEISqNodeInfoFpga</a> .         |
| <b>network</b>                | A structure that contains network interface information for the SynqNet node. See <a href="#">MEISqNodeInfoNetwork</a> . |
| <b>io</b>                     | 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 */

    long            exactMatch; /* TRUE/FALSE */

    char            serialNumber[MEISqNodeID\_CHAR\_MAX];
    char            modelName[MEISqNodeID\_CHAR\_MAX];
    char            manufacturerData[MEISqNodeManufacturerDATA\_CHAR\_MAX];
} MEISqNodeInfoId;
```

## 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 **modelName** 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**.

|                  |                                                                                                                                                                                                                                                                                                                                                                      |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>nodeType</b>  | 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. |
| <b>*nodeName</b> | A string that represents the SynqNet nodeType. The nodeName string matches the name of the SqNodeLib node specific header file.                                                                                                                                                                                                                                      |
| <b>option</b>    | The product option code within a product series.                                                                                                                                                                                                                                                                                                                     |
| <b>switchId</b>  | 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).                                                                                                                                                 |

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>unique</b>           | <p>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.</p> <p><b>NOTE:</b> 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.</p> |
| <b>exactMatch</b>       | 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.                                                                                                                                                                                                                             |
| <b>serialNumber</b>     | 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.                                                                                                                                                                                                                                                                                                                      |
| <b>modelNumber</b>      | A string that represents the SynqNet node model number. The SynqNet node manufacturer determines the model number.                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>manufacturerData</b> | 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.

|                        |                                                                                                                        |
|------------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>digitalInCount</b>  | The number of digital inputs on a SynqNet node.                                                                        |
| <b>digitalOutCount</b> | The number of digital outputs on a SynqNet node.                                                                       |
| <b>analogInCount</b>   | The number of analog inputs on a SynqNet node.                                                                         |
| <b>analogOutCount</b>  | The number of analog outputs on a SynqNet node.                                                                        |
| <b>segmentCount</b>    | The total number of segments on a SynqNet node.                                                                        |
| <b>maxWait</b>         | This is the maximum amount of time between when the output bit is set in software and the hardware state takes effect. |

## See Also

[meiSqNodeInfo](#) | [meiSqNodeSegmentDigitalOutGet](#) | [meiSqNodeSegmentDigitalOutSet](#) | [meiSqNodeSegmentAnalogOutGet](#) | [meiSqNodeSegmentAnalogOutSet](#)

# MEISqNodeInfoFpga

## Definition

```
typedef struct MEISqNodeInfoFpga {
    MEISqNodeFpgaType      type;
    unsigned long          vendorDevice;
    unsigned long          version;
    unsigned long          branchVersion;
    long                   defaultVersion; /* TRUE/FALSE */
} MEISqNodeInfoFpga;
```

## Description

**MEISqNodeInfoFpga** contains identification data for the SynqNet node FPGA.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>         | The FPGA type. See <a href="#">MEISqNodeFpgaType</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>vendorDevice</b> | 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. |
| <b>version</b>      | <p>A 32-bit value that represents the revision of the device.</p> <p>The upper 16 bits (SqMac Version), represent the SynqNet network interface revision.</p> <p>The lower 16 bits (Node Version), represent the device revision.</p> <p>Typically, the version value is displayed in hexadecimal format.</p> <p>Ex: 0x02400344</p> <p>SqMac Version: 0240<br/>Node Version: 0344</p>                                                                                                                                                                |

|                       |                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>branchVersion</b>  | <p>A 32-bit value that identifies the branch from an existing version (MajorMinor) or from another Branch.</p> <p>The upper 16 bits (SqMac Branch Version), represent the SynqNet network interface branch revision.</p> <p>The lower 16 bits (Node Branch Version), represent the device branch revision.</p> <p>Ex: 0x01010102</p> <p>SqMac Branch Version: 0101<br/>Node Branch Version: 0102</p> |
| <b>defaultVersion</b> | <p>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.</p>                                                                                                                                                                        |

## 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.

|                 |                                                             |
|-----------------|-------------------------------------------------------------|
| <b>number</b>   | An index to a SynqNet network associated with a controller. |
| <b>inPorts</b>  | The number of SynqNet IN port network interfaces.           |
| <b>outPorts</b> | 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.

|                               |                                   |
|-------------------------------|-----------------------------------|
| <b>MEISqNodeMemoryDATA</b>    | node/drive processor data memory  |
| <b>MEISqNodeMemoryPROGRAM</b> | drive processor program memory    |
| <b>MEISqNodeMemoryIO</b>      | drive I/O memory                  |
| <b>MEISqNodeMemoryDRIVE</b>   | direct command to drive processor |

## See Also

[MEISqNodeCmdHeader](#) | [meiSqNodeCommand](#)

# MEISqNodeMessage

## Definition

```
typedef enum {
    MEISqNodeMessageINVALID,
    MEISqNodeMessageNODE_INVALID,
    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_INVALID,
    MEISqNodeMessageNODE_FAILURE,

    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_ERROR,
    MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT,
    MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH,

    MEISqNodeMessageBOOT_FILE_NOT_FOUND,
    MEISqNodeMessagePARAM_READ_ONLY,
    MEISqNodeMessagePARAM_LOCKED,
```



```

    MEISqNodeMessageMONITOR_INDEX,
    MEISqNodeMessageMONITOR_ADDRESS,
} MEISqNodeMessage;

```

**Required Header:** stdmei.h

**Change History:** 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.

### MEISqNodeMessageCONFIG\_NETWORK\_MISMATCH

The type of map file specified in meiSqNodeDriveParamFileSet 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

Currently unused and is reserved for future use.

### MEISqNodeMessageREADY\_TIMEOUT

Currently unused and is reserved for future use.

### MEISqNodeMessageSRVC\_ERROR

Currently unused and is reserved for future use.

**MEISqNodeMessageSRVC\_UNSUPPORTED**

Currently unused and is reserved for future use.

**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 resources.

**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.defaultVersion](#).

**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 may 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.

**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**

During the intialization of the modules attached to a SQID node, the length of at least one of the inter module buses did not match the length calculated from data in the module EEPROMs.

**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\_ERROR**

An error was encountered while initializing the slice node.

**MEISqNodeMessageIO\_SLICE\_INITIALIZATION\_TIMEOUT**

The slice node did not initialize within the expected time.

**MEISqNodeMessageIO\_SLICE\_TOPOLOGY\_MISMATCH**

The slices attached to the network adapter are different than the stored/expected arrangement of slices.

**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.

## See Also

[meiSqNodeDriveMapParamFileSet](#) | [meiSqNodeConfigSet](#)

# 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 <code>MEISqNodeMonitorValueIndex</code> enumeration. |

## See Also

[meiSqNodeDriveMonitor](#) | [meiSqNodeDriveMonitorConfigGet](#) | [meiSqNodeDriveMonitorConfigSet](#)

# MEISqNodeMonitorValueIndex

## Definition

```
typedef enum MEISqNodeMonitorValueIndex {  
    MEISqNodeMonitorValueIndexA,  
    MEISqNodeMonitorValueIndexB,  
    MEISqNodeMonitorValueIndexC,  
    MEISqNodeMonitorValueIndexD,  
} MEISqNodeMonitorValueIndex;
```

## Description

**MEISqNodeMonitorValueIndex** is an enumeration of indices to node monitor values.

|                                    |                                |
|------------------------------------|--------------------------------|
| <b>MEISqNodeMonitorValueIndexA</b> | Index to node monitor value A. |
| <b>MEISqNodeMonitorValueIndexB</b> | Index to node monitor value B. |
| <b>MEISqNodeMonitorValueIndexC</b> | Index to node monitor value C. |
| <b>MEISqNodeMonitorValueIndexD</b> | Index to node monitor value D. |

## See Also

[meiSqNodeDriveMonitor](#) | [meiSqNodeDriveMonitorConfigGet](#) | [meiSqNodeDriveMonitorConfigSet](#)

# MEISqNodeResponse

## Definition

```
typedef struct MEISqNodeResponse {  
    unsigned long    data;    /* response data */  
} MEISqNodeResponse;
```

## Description

**MEISqNodeResponse** contains the service command response data.

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>data</b> | 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.

|                     |                                                                                                                                                                                                                  |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>id</b>           | This field contains a 32-bit number that uniquely identifies this kind of segment.<br>For modules attached to a SQID node, the top 16 bits are the manufacturer code and the bottom 16 bits are to product code. |
| <b>option</b>       | The option code for the segment.<br>For slices attached to a Slice network adaptor this field is always zero.                                                                                                    |
| <b>serialNumber</b> | The serial number of this segment .                                                                                                                                                                              |
| <b>modelName</b>    | A text string giving the model name of this module.<br><br>For slices attached to a Slice network adaptor, this field is always a null string.                                                                   |

|                         |                                                                                                                                                     |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>digitalInCount</b>   | The total number of digital inputs on this segment.                                                                                                 |
| <b>digitalOutCount</b>  | The total number of digital outputs on this segment.                                                                                                |
| <b>analogInCount</b>    | The total number of analog inputs on this segment.                                                                                                  |
| <b>analogOutCount</b>   | The total number of analog outputs on this segment.                                                                                                 |
| <b>version</b>          | The version of the segment.<br>For modules attached to a SQID node, this field is always zero.                                                      |
| <b>paramCount</b>       | The total number segment parameters supported by this segment.<br>For modules attached to a SQID node this field is always zero.                    |
| <b>memoryCount</b>      | The total number of memory bytes available on this segment.<br>For modules attached to a SQID node, this field is always zero.                      |
| <b>manufacturerData</b> | A series of characters programmed into the node during manufacturing.<br>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.

|             |                         |
|-------------|-------------------------|
| <b>data</b> | 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;
} MEISqNodeStatus;
```

## Description

**MEISqNodeStatus** contains error counters and the ***eventMask*** for a SynqNet node.

|                        |                                                                                                                                                                                                                           |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>upStreamError</b>   | The rate and count of bad synqNet messages received by the controller from the Node. See <a href="#">MEISqNodeStatusPacketError</a> .                                                                                     |
| <b>downStreamError</b> | The rate and count of bad synqNet messages received by the Node from the controller. See <a href="#">MEISqNodeStatusPacketError</a> .                                                                                     |
| <b>crcError</b>        | Counters for the CRC errors. See <a href="#">MEISqNodeStatusCrcError</a> .                                                                                                                                                |
| <b>eventMask</b>       | Array that defines the event mask bits. The array is defined as:<br>typedef MPIEventMaskeLEMENT_TYPE<br>MPIEEventMask[MPIEventMaskeLEMENTS]<br>The bits are defined by the <a href="#">MPI/MEIEventType</a> enumerations. |

## See Also

[meiSqNodeStatus](#) | [meiSynqNetStatus](#) | [MEISqNodeConfig](#)

# 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.

|             |                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>port</b> | 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. |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------|

## See Also

[meiSqNodeStatus](#) | [MEINetworkPort](#)

# 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. |
| count | The packet error counter. The valid range is 0 to 255. The value saturates at 255.      |

## 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.

|             |                                                                                                                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>data</b> | 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](#)

# 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. 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](#)

# 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:** stdmpi.h

## Description

**MEIDriveMapParamMAX\_STRING\_LENGTH** macro defines the maximum length of a string that can be read from or written to a drive parameter.

## See Also

# MEIFPGARINCONREV

## Definition

```
#define  MEIFPGARINCONREV  (0x0224)
```

**Change History:** 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    (0x020E)
```

## 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    (0x0207)
```

## 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 (0x02FF)
```

## 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    (0x0341)
```

**Change History:** 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    (0x0303)
```

## 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    (0x03FF)
```

## 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 network 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.

## Methods

### Create, Delete, Validate Methods

[meiSynqNetCreate](#)

[meiSynqNetDelete](#)

[meiSynqNetValidate](#)

### Configuration and Information Methods

[meiSynqNetCableNumToNodePort](#)

[meiSynqNetConfigGet](#)

[meiSynqNetConfigSet](#)

[meiSynqNetFlashConfigGet](#)

[meiSynqNetFlashConfigSet](#)

[meiSynqNetIdleCableListGet](#)

[meiSynqNetIdleCableStatus](#)

[meiSynqNetInfo](#)

[meiSynqNetPacketFlashConfigGet](#)

[meiSynqNetPacketFlashConfigSet](#)

[meiSynqNetPortToCableNum](#)

[meiSynqNetPacketConfigGet](#)

[meiSynqNetPacketConfigSet](#)

[meiSynqNetStatus](#)

[meiSynqNetTiming](#)

### Action Methods

[meiSynqNetFlashTopologyClear](#)

[meiSynqNetFlashTopologySave](#)

[meiSynqNetInit](#)

[meiSynqNetShutdown](#)

## Event Methods

[meiSynqNetEventNotifyGet](#)

[meiSynqNetEventNotifySet](#)

[meiSynqNetEventReset](#)

## Relational Methods

[meiSynqNetControl](#)

[meiSynqNetNumber](#)

## Data Types

[MEISynqNetCableList](#)

[MEISynqNetCableLength](#)

[MEISynqNetCableStatus](#)

[MEISynqNetConfig](#)

[MEISynqNetFailedNodeMask](#)

[MEISynqNetInfo](#)

[MEISynqNetMessage](#)

[MEISynqNetRecoveryMode](#)

[MEISynqNetResourceCommand](#)

[MEISynqNetResourceIoBits](#)

[MEISynqNetResourceMonitor](#)

[MEISynqNetPacketCfg](#)

[MEISynqNetPacketCfgEncoder](#)

[MEISynqNetPacketCfgIo](#)

[MEISynqNetPacketCfgMotor](#)

[MEISynqNetPacketCfgNode](#)

[MEISynqNetPacketCfgProbe](#)

[MEISynqNetResourceCfgProbeDepth](#)

[MEISynqNetState](#)

[MEISynqNetStatus](#)

[MEISynqNetStatusCrcError](#)

[MEISynqNetTiming](#)

[MEISynqNetTrace](#)

## Constants

[MEISynqNetMaxCableHOP\\_COUNT](#)

[MEISynqNetMaxMotorFEEDBACK\\_PRIMARY\\_COUNT](#)

[MEISynqNetMaxMotorFEEDBACK\\_SECONDARY\\_COUNT](#)

[MEISynqNetMaxMotorCAPTURE\\_COUNT](#)

[MEISynqNetMaxMotorCOMPARE\\_COUNT](#)

[MEISynqNetMaxMotorENCODER\\_COUNT](#)

[MEISynqNetMaxMotorPULSE\\_ENGINE\\_COUNT](#)

[MEISynqNetMaxMotors](#)

[MEISynqNetMaxNodeMOTORS](#)

[MEISynqNetMaxNODE\\_COUNT](#)

[MEISynqNetNodeMaskELEMENTS](#)

[MEISynqNetPacketCfgIo](#)

[MEISynqNetPacketCfgMotor](#)

[MEISynqNetPacketCfgNode](#)

# 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.

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| <b>control</b> | a handle to a Control object                                                                           |
| <b>number</b>  | An index to the SynqNet network.<br>First network = 0<br>Second network = 1<br>Third network = 2, etc. |

| Return Values        |                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------|
| <b>handle</b>        | to a SynqNet object. After creating a SynqNet object it must be validated using meiSynqNetValidate(). |
| <b>MPIHandleVOID</b> | 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++ constructor. All objects that are created must be deleted in the reverse order to avoid memory leaks.

|                |                               |
|----------------|-------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object. |
|----------------|-------------------------------|

### Return Values

|                     |                                                          |
|---------------------|----------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SynqNetDelete</i> successfully deleted the object. |
|---------------------|----------------------------------------------------------|

## 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.

|                |                                   |
|----------------|-----------------------------------|
| <b>synqNet</b> | handle to a valid SynqNet object. |
|----------------|-----------------------------------|

| Return Values                               |                                                         |
|---------------------------------------------|---------------------------------------------------------|
| <b>MPIMessageOK</b>                         | if SynqNet is a handle to a valid object.               |
| <b>MEISynqNetMessageINTERFACE_NOT_FOUND</b> | if the controller does not support a SynqNet interface. |

## 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  | if <i>SynqNetCableNumToNodePort</i> successfully converts a cable number on a SynqNet network into a node number and port. |

## 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***.

|                |                                          |
|----------------|------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object             |
| <b>*config</b> | a pointer to a SynqNet config structure. |

| Return Values       |                                                                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSynqNetConfigGet</i> successfully reads a SynqNet network's configuration and writes it into the structure pointed to by <i>config</i> . |

## 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*.

|                |                                          |
|----------------|------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object             |
| <b>*config</b> | a pointer to a SynqNet config structure. |

| Return Values                     |                                                                                                                                      |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>               | if <i>meiSynqNetConfigSet</i> successfully writes a SynqNet network's configuration from the structure pointed to by <i>config</i> . |
| <b>MPIMessageARG_INVALID</b>      | if config is NULL                                                                                                                    |
| <b>MPIMessagePARAM_INVALID</b>    | if an invalid recoveryMode was specified or a non-zero cable length value was specified for a non-existent cable.                    |
| <b>MPISynqNetMessageRING_ONLY</b> | only a value of MEISynqNetRecoveryModeDISABLED for recoveryMode is allowed on a string network.                                      |

## 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***.

|                |                                                                                                                                                                        |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object                                                                                                                                           |
| <b>*config</b> | pointer to a locally instantiated MEISynqNetConfig structure.                                                                                                          |
| <b>*flash</b>  | <i><b>flash</b></i> is either an MEIFlash handle or MPIHandleVOID. If <i><b>flash</b></i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |

| Return Values       |                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSynqNetIdleFlashConfigGet</i> successfully writes the Motor's flash configuration to the structure(s). |

## See Also

[meiSynqNetFlashConfigSet](#) | [MEISynqNetConfig](#) | [MEIFlash](#)

# meiSynqNetFlashConfigSet

## Declaration

```
long meiSynqNetFlashConfigSet(MEISynqNet          synqNet,  
                             MEISynqNetConfig  *flash,  
                             MEIFlash        *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(.).

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>*flash</b>  | <i><b>flash</b></i> is either an MEIFlash handle or MPIHandleVOID. If <i><b>flash</b></i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.<br><br>If <i><b>flash</b></i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated but the actual write to controller flash will not occur. Use <a href="#">meiFlashMemoryFromFileType(...)</a> to prompt the actual write to flash. |
| <b>*config</b> | pointer to a locally instantiated <a href="#">MEISynqNetConfig</a> structure that has been initialized by performing a call to <a href="#">meiSynqNetFlashConfigGet(.)</a> or <a href="#">meiSynqNetConfigGet(.)</a> .                                                                                                                                                                                                            |

### Return Values

|                                |                                                                                                                            |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>            | if <i>meiSynqNetIdleFlashConfigSet</i> successfully sets the Motor's flash configuration using data from the structure(s). |
| <b>MPIMessageARG_INVALID</b>   | if config is NULL.                                                                                                         |
| <b>MPIMessagePARAM_INVALID</b> | if an invalid recoveryMode was specified or a non-zero cable length value was specified for a non-existent cable.          |

|                                              |                                                                                                           |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetMessageRING_ONLY</b>            | only a value of<br>MEISynqNetRecoveryModeDISABLED<br>for recoveryMode is allowed on a string<br>network.  |
| <b>MEIFlashMessageNETWORK_TOPOLOGY_ERROR</b> | if topology has not yet been saved to<br>flash using<br><a href="#">meiSynqNetFlashTopologyClear(...)</a> |

## 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.

|                   |                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqNet</b>    | a handle to a SynqNet object                                                                                                            |
| <b>*idleCable</b> | a pointer to a SynqNet cable list structure. The cable list structure contains the number of idle cables and their identifying numbers. |

| Return Values       |                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSynqNetIdleCableListGet</i> successfully reads a SynqNet network's list of idle cables and writes them into the structure pointed to by <i>idleCable</i> . |

## 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. SynqNetIdleCableStatus 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.

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <b>synqNet</b>      | a handle to a SynqNet object                          |
| <b>cableNumber</b>  | the number of the cable to be tested                  |
| <b>*cableStatus</b> | a pointer to a SynqNet cable status enumerated value. |

| Return Values       |                                                                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>meiSynqNetIdleCableStatus</i> successfully reads an idle cable's status and writes it into the structure pointed to by <i>cableStatus</i> . |

## 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.

|                |                                                    |
|----------------|----------------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object                       |
| <b>*info</b>   | pointer to a SynqNet network information structure |

### Return Values

|                              |                                                                                                         |
|------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SynqNetInfo</i> successfully reads the network information and writes it into the info structure. |
| <b>MPIMessageARG_INVALID</b> | if the info pointer is NULL.                                                                            |

## 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  | <i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.<br><br>If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use <a href="#">meiFlashMemoryFromFileType(...)</a> to prompt the actual write to flash. |
| *config | pointer to configuration structure defined by <a href="#">MEISynqNetPacketCfg</a> .                                                                                                                                                                                                                                                                                                                           |

| Return Values         |                          |
|-----------------------|--------------------------|
| MPIMessageUNSUPPORTED | Currently not supported. |

## 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.

|                |                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object                                                                                                                                                                                                                                                                                                                                                                    |
| <b>*flash</b>  | <p><i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.</p> <p>If <i>flash</i> 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</p> |
| <b>*config</b> | pointer to configuration structure defined by MEISynqNetPacketCfg.                                                                                                                                                                                                                                                                                                                              |

| Return Values                |                          |
|------------------------------|--------------------------|
| <b>MPIMessageUNSUPPORTED</b> | Currently not supported. |

## 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.

|                     |                                |
|---------------------|--------------------------------|
| <b>synqNet</b>      | a handle to the SynqNet object |
| <b>nodeNumber</b>   | the number of the node         |
| <b>port</b>         | an enumerated port value       |
| <b>*cableNumber</b> | a pointer to a cable number    |

| Return Values       |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SynqNetPortToCableNum</i> successfully converts a node number and port on a SynqNet network into a cable number. |

## See Also

[MEISynqNetCableList](#)

# meiSynqNetPacketConfigGet

## Declaration

```
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 SynqNetPacketConfigSet. This method is also useful for optimizing network traffic (bandwidth).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

|                |                                                                    |
|----------------|--------------------------------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object                                       |
| <b>*config</b> | pointer to configuration structure defined by MEISynqNetPacketCfg. |

| Return Values                |                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SynqNetPacketConfigGet</i> successfully reads the network packet configuration. |
| <b>MPIMessageARG_INVALID</b> | if the config pointer is NULL.                                                        |
| <b>MPIMessageUNSUPPORTED</b> | An sqNodeLib module is missing or it contains a corrupt a resource table.             |

## See Also

[meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfgNode](#)

# meiSynqNetPacketConfigSet

## Declaration

```
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.

This method is useful for optimizing network traffic (bandwidth).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

|                |                                                                    |
|----------------|--------------------------------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object                                       |
| <b>*config</b> | pointer to configuration structure defined by MEISynqNetPacketCfg. |

| Return Values                                 |                                                                                                                                    |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                           | if <i>SynqNetPacketConfigSet</i> successfully writes the network packet configuration.                                             |
| <b>MPIMessageARG_INVALID</b>                  | if the config pointer is NULL.                                                                                                     |
| <b>MEISynqNetMessageINCOMPLETE_MOTOR</b>      | minimum packet requirements for a motor have not been met. Be sure that your motor has at least one command field and one encoder. |
| <b>MEISynqNetMessageINVALID_AUX_ENC_COUNT</b> | too many secondary encoders have been configured on a node, or there are more secondary encoders than configured/enabled motors.   |
| <b>MEISynqNetMessageINVALID_MOTOR_COUNT</b>   | more motors configured than supported by the node.                                                                                 |



|                                               |                                                                                                                                                                                   |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetMessageINVALID_COMMAND_CFG</b>   | an unsupported command value has been selected, or it is not supported by the node.                                                                                               |
| <b>MEISynqNetMessageINVALID_ENCODER_COUNT</b> | feedback count exceeds MEISynqNetMessageMaxEncoderFEEDBACK_COUNT, or exceeds the count supported by the node.                                                                     |
| <b>MEISynqNetMessageINVALID_CAPTURE_COUNT</b> | capture count value is out of range, capture count exceeds capture resources supported by the node, or capture count is non-zero when encoder is disabled (feedbackCount == 0x0). |
| <b>MEISynqNetMessageINVALID_COMPARE_COUNT</b> | compare count value out of range, compare count exceeds compare resources supported by the node, or compare count is non-zero when encoder is disabled (feedbackCount == 0x0).    |
| <b>MEISynqNetMessageINVALID_INPUT_COUNT</b>   | ioInput count is out of range or it exceeds the count supported by the node.                                                                                                      |
| <b>MEISynqNetMessageINVALID_OUTPUT_COUNT</b>  | ioOutput count is out of range or it exceeds the count supported by the node.                                                                                                     |
| <b>MEISynqNetMessageINVALID_MONITOR_CFG</b>   | monitorConfig is out of range or it exceeds the count supported by the node.                                                                                                      |
| <b>MPIMessageUNSUPPORTED</b>                  | an sqNodeLib module is missing or it contains a corrupt a resource table.                                                                                                         |

## See Also

[meiSynqNetMessageConfigGet](#) | [MEISynqNetMessageCfgNode](#) | [MEISynqNetMessageCfg](#)

# 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.

|                |                                       |
|----------------|---------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object          |
| <b>*status</b> | pointer to a SynqNet status structure |

| Return Values                |                                                                                                  |
|------------------------------|--------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>meiSynqNetStatus</i> successfully gets the network status and writes it into the structure |
| <b>MPIMessageARG_INVALID</b> | if the status pointer is NULL                                                                    |

## 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.

|                |                                            |
|----------------|--------------------------------------------|
| <b>synqNet</b> | handle to a valid SynqNet object.          |
| <b>*timing</b> | a pointer to a MEISynqNetTiming structure. |

| Return Values                |                                                                              |
|------------------------------|------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>meiSynqNetTiming</i> successfully retrieves the network timing values. |
| <b>MPIMessageARG_INVALID</b> | if <i>*timing</i> equals NULL.                                               |

## 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.

|                |                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqNet</b> | a handle to an MEISynqNet object whose network topology is to be cleared.                                                                 |
| <b>flash</b>   | is either a vaild MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |

| Return Values                          |                                                                                                                                                                                                                                                                         |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                    | if <i>meiSynqNetFlashTopologyClear</i> successfully clears the networks topology information. <b><i>flash</i></b> is either an MEIFlash handle or MPIHandleVOID.<br>If <b><i>flash</i></b> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |
| <b>MPISynqNetMessageTOPOLOGY_CLEAR</b> | if topology information has already been cleared or if it has not yet been saved to flash.                                                                                                                                                                              |

## 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. 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.

|                |                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>synqNet</b> | a handle to an MEISynqNet object whose network topology is to be saved.                                                                   |
| <b>flash</b>   | is either a valid MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. |

| Return Values                          |                                                                                                                                           |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MPIMessageOK</b>                    | if <i>meiSynqNetFlashTopologySave</i> successfully saves the current SynqNet topology to the controller's flash memory.                   |
| <b>MPISynqNetMessageTOPOLOGY_SAVED</b> | if topology information has already been saved to flash. You must clear network topology before saving another network topology to flash. |

## 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 mpiControllInit(.).

|         |                              |
|---------|------------------------------|
| synqNet | a handle to a SynqNet object |
|---------|------------------------------|

| Return Values         |                                                            |
|-----------------------|------------------------------------------------------------|
| MPIMessageOK          | if <i>SynqNetInit</i> successfully initializes the network |
| MPIMessageARG_INVALID | if the <i>eventMask</i> pointer is NULL.                   |

## See Also

[meiSynqNetInfo](#) | [meiSynqNetStatus](#)



# meiSynqNetShutdown

## Declaration

```
long  meiSynqNetPacketFlashConfigSet(MEISynqNet      synqNet,  
                                     void            *flash,  
                                     MEISynqNetPacketCfg *config);
```

**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  | if <i>meiSynqNetShutdown</i> successfully disables a SynqNet network by shutting off cyclic data transmission from the controller to the nodes. |

## 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          | if meiSynqNetEventNotifyGet successfully reads the event mask |
| MPIMessageARG_INVALID | if the eventMask pointer is NULL.                             |

## 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).

|                  |                                                                                        |
|------------------|----------------------------------------------------------------------------------------|
| <b>synqNet</b>   | a handle to a SynqNet object                                                           |
| <b>eventMask</b> | pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations. |
| <b>*external</b> | pointer to external                                                                    |

| Return Values                |                                                                    |
|------------------------------|--------------------------------------------------------------------|
| <b>MPIMessageOK</b>          | if <i>SynqNetEventNotifySet</i> successfully writes the event mask |
| <b>MPIMessageARG_INVALID</b> | if the eventMask pointer is NULL                                   |

## 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 | <div>An array that defines the event mask bits. The array is defined as:<br/><br/><pre>typedef MPIEventMaskELEMENT_TYPE<br/>        MPIEventMask[MPIEventMaskELEMENTS]</pre><br/>The bits are defined by the MPI/MEIEventType enumerations.</div> |

| Return Values |                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| MPIMessageOK  | if <i>meiSynqNetEventReset</i> successfully resets the status/event bits that are specified in the eventMask and generated by the synqNet object. |

## See Also

[meiSynqNetStatus](#) | [meiSqNodeEventReset](#) | [meiSqNodeStatus](#)

# 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        |                               |
|----------------------|-------------------------------|
| <b>MPIControl</b>    | a handle to a control object. |
| <b>MPIHandleVOID</b> | 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.).

|                |                                              |
|----------------|----------------------------------------------|
| <b>synqNet</b> | a handle to a SynqNet object.                |
| <b>*number</b> | a pointer to the index of a SynqNet network. |

| Return Values       |                                                                |
|---------------------|----------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>SynqNetNumber</i> successfully reads the network number. |

## 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.

|                    |                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>count</b>       | The number of cables and number of valid elements in the <code>cableNumber[]</code> array.. Range is 0 to <code>MEISynqNetCableHOP_COUNT</code> . |
| <b>cableNumber</b> | 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.

|                |                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------|
| <b>minimum</b> | Minimum cable length in meters.<br><br>A value less than zero or greater than the nominal value is invalid. |
| <b>nominal</b> | Nominal cable length in meters.<br><br>A value less than zero is invalid.                                   |
| <b>maximum</b> | Maximum cable length in meters.<br><br>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.

|                                            |                                                                                                              |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetCableStatusGOOD</b>           | Network communication across the cable is working properly in both directions.                               |
| <b>MEISynqNetCableStatusBAD_UPSTREAM</b>   | Network communication across the cable failed in the upstream direction (from node to controller).           |
| <b>MEISynqNetCableStatusBAD_DOWNSTREAM</b> | Network communication across the cable failed in the downstream direction (from the controller to the node). |
| <b>MEISynqNetCableStatusBAD</b>            | 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  | <p>An array of cable lengths. Range for cable lengths is 0 to 100 meters.</p> <p>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 <code>MEISynqNetMessageCABLE_LENGTH_MISMATCH</code>.</p> <p>By default, the values in this structure are zero. Most applications will not need to modify these defaults.</p> <p><b>NOTE:</b> 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 <code>meiSynqNetFlashTopologyClear(.)</code></p> |

## See Also

[meiSynqNetConfigGet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetInfo](#) | [meiSynqNetFlashConfigGet](#) | [meiSynqNetFlashConfigSet](#) | [meiSynqNetFlashTopologyClear](#)

# MEISynqNetFailedNodeMask

## Definition

```
#define MEISynqNetNodeMaskELEMENTS  (1)

typedef long MEISynqNetFailedNodeMask  [MEISynqNetNodeMaskELEMENTS];
```

## 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.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>networkType</b> | contains currently discovered network topology type.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>nodeCount</b>   | Contains the number of nodes currently discovered on the network.                                                                                                                                                                                                                                                                                                                                                               |
| <b>nodeOffset</b>  | Starting node number for nodes on this network. Nodes are currently numbered sequentially across all networks on a controller.                                                                                                                                                                                                                                                                                                  |
| <b>cableLength</b> | <div>An array of measured network cable lengths. These values are estimated values.</div> <div>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.</div> <div>These values are estimated values and are the values used in the network scheduling calculations.</div> |

## 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_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_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,
} MEISynqNetMessage;

```

**Required Header:** stdmei.h

**Change History:** 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 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\(.\)](#), [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\_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\_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 to the 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.

**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.



|                                          |                                                                                                                                                                                                                                   |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetRecoveryModeDISABLED</b>    | The network will not attempt to recover from a fault condition. This is the default mode for string topologies.                                                                                                                   |
| <b>MEISynqNetRecoveryModeSINGLE_SHOT</b> | The network will only attempt to recover from a fault condition one time. A second fault will be ignored.                                                                                                                         |
| <b>MEISynqNetRecoveryModeAUTO_ARM</b>    | 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_ABC,
} MEISynqNetResourceCommand;
```

## 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.<br>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_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.

|                                        |                                                      |
|----------------------------------------|------------------------------------------------------|
| <b>MEISynqNetResourceIoBitsINVALID</b> | Non-valid bit count was detected.                    |
| <b>MEISynqNetResourceIoBitsNONE</b>    | No I/O bits will be sent/received.                   |
| <b>MEISynqNetResourceIoBits16</b>      | One data field of 16 I/O bits will be sent/received. |
| <b>MEISynqNetResourceIoBits48</b>      | 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.

|                                         |                                                                         |
|-----------------------------------------|-------------------------------------------------------------------------|
| <b>MEISynqNetResourceMonitorINVALID</b> | Non-valid bit count was detected.                                       |
| <b>MEISynqNetResourceMonitorNONE</b>    | No monitor data fields will be sent/received for this motor.            |
| <b>MEISynqNetResourceMonitorAB</b>      | Two 16bit fields of monitor data will be sent/received for this motor.  |
| <b>MEISynqNetResourceMonitorABCD</b>    | 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.

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>feedbackCount</b> | <p>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.</p> <p><b>NOTE:</b> 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).</p>                                                                           |
| <b>captureCount</b>  | <p>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.</p> <p><b>NOTE:</b> 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.</p> |

**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.

|                        |                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------|
| <b>digitalInCount</b>  | Selects the number of 32bit words of digital input data to receive over the SynqNet network. |
| <b>digitalOutCount</b> | Selects the number of 32bit words of digital output data to send over the SynqNet network.   |
| <b>analogInCount</b>   | Selects the number of 32bit words of analog input data to receive over the SynqNet network.  |
| <b>analogOutCount</b>  | 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.

|                             |                                                                                                                                                                                                                                                                                                                 |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>command</b>              | Selects the command data type and count. Please see <a href="#">MEISynqNetResourceCommand</a> .                                                                                                                                                                                                                 |
| <b>pulseEngineCount</b>     | Configures the number of pulse engine to be enable for a motor. Valid numbers are zero thru <b>MEISynqNetMaxMotorPULSE_ENGINE_COUNT</b> , but may be further limited by the available resources on the node.<br><br>Each pulse engine requires a 64-bit upstream data field and a 32-bit downstream data field. |
| <b>feedbackPrimaryCount</b> | Configures the number of 32-bit feedback data fields to be sent for a motor. Valid numbers are zero thru <b>MEISynqNetMaxEncoderFEEDBACK_COUNT</b> , but may be further limited by the available resources on the node.                                                                                         |

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>captureCount</b> | <p>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.</p> <p><b>NOTE:</b> 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.</p> |
| <b>compareCount</b> | <p>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.</p> <p><b>NOTE:</b> 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.</p>                                                                    |
| <b>ioInput</b>      | Selects the number of input bit data received from this motor. Please see <a href="#">MEISynqNetResourceIoBits</a> .                                                                                                                                                                                                                                                                                                                                                          |
| <b>ioOutput</b>     | Selects the number of output bit data sent to this motor. Please see <a href="#">MEISynqNetResourceIoBits</a> .                                                                                                                                                                                                                                                                                                                                                               |
| <b>monitor</b>      | Selects the number of montior data fields received from this motor. Please see <a href="#">MEISynqNetResourceMonitor</a> .                                                                                                                                                                                                                                                                                                                                                    |
| <b>probe</b>        | 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.

|                               |                                                                                                                                                                                                                                                   |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>motor</b>                  | 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 <a href="#">MEISynqNetMaxNodeMOTORS</a> .                        |
| <b>io</b>                     | Configures the network data packets general purpose node I/O.                                                                                                                                                                                     |
| <b>feedbackSecondaryCount</b> | Configures the number of 32-bit secondary feedback data fields to be sent for a node. Valid numbers are zero thru <a href="#">MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT</a> , 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.

|                                            |                                                        |
|--------------------------------------------|--------------------------------------------------------|
| <b>MEISynqNetResourceProbeDepthNONE</b>    | zero Probe data registers will be transmitted upstream |
| <b>MEISynqNetResourceProbeDepthTWO</b>     | 2 Probe data registers will be transmitted upstream    |
| <b>MEISynqNetResourceProbeDepthFOUR</b>    | 4 Probe data registers will be transmitted upstream    |
| <b>MEISynqNetResourceProbeDepthEIGHT</b>   | 8 Probe data registers will be transmitted upstream    |
| <b>MEISynqNetResourceProbeDepthSIXTEEN</b> | 16 Probe data registers will be transmitted upstream   |

## See Also

# 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(.)`.

|                                 |                                                                                                                                                                                                                                                                                  |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetStateDISCOVERY</b> | 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. |
| <b>MEISynqNetStateASYNQ</b>     | This state is used for off-line operations, like node binary download. Data packets are sent/received asynchronously.                                                                                                                                                            |
| <b>MEISynqNetStateSYNQ</b>      | 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;
    long                   topologySaved; /* TRUE/FALSE */
    MEISynqNetStatusCrcError crcError;
    MPIEventMask            eventMask;
    MEISynqNetFailedNodeMask failedNodeMask;
} MEISynqNetStatus;
```

## 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.<br><br>FALSE means the topology is dynamically discovered at initialization.<br><br>TRUE means that the topology is verified against an expected topology at initialization.<br><br>See <a href="#">Saving Current Topology To Flash</a> 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:<br><br><pre>typedef MPIEventMaskELEMENT_TYPE         MPIEventMask[MPIEventMaskELEMENTS]</pre><br>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];
```

## See Also

[meiSynqNetStatus](#) | [meiSqNodeStatus](#) | [MEISqNodeStatus](#)

# 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.

|             |                                                    |
|-------------|----------------------------------------------------|
| <b>port</b> | 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 driveUpdateFreq;     /* kHz */
    double driveUpdatePeriod;  /* uS */

    long   bandwidthUsage;      /* % */

    struct {
        double calculationTime; /* uS */
        double calculationSlack; /* uS */
        double demandLatency;   /* uS */
        double feedbackLatency; /* uS */

        double total;           /* uS */
        double latencySlack;    /* uS */
    } controlLatency;

    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;
} MEISynqNetTiming;
```

## 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.

|                          |                                                                                                                                                                                                                                                                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>controllerFreq</b>    | <p>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.</p> <p>This value is configurable. See <a href="#">MPIControlConfig</a>.</p> |
| <b>controllerPeriod</b>  | <p>The sample period calculation of the SynqNet controller board, in uS. Derived from controllerFreq.</p>                                                                                                                                                                                                                   |
| <b>txTime</b>            | <p>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%.</p> <p>This value is configurable. See <a href="#">MEIControlConfig</a>.</p>                                                |
| <b>calculationLimit</b>  | <p>The maximum allowed foreground calculation time, in uS. Derived from txTime * controllerPeriod.</p>                                                                                                                                                                                                                      |
| <b>driveUpdateFreq</b>   | <p>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.</p>                                                                                       |
| <b>driveUpdatePeriod</b> | <p>The cyclic communication period of a drive processor on a SynqNet node, in uS. Derived from driveUpdateFreq.</p>                                                                                                                                                                                                         |
| <b>bandwidthUsage</b>    | <p>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.</p>                                                                |
| <b>calculationTime</b>   | <p>The foreground calculation time of the controller, in uS.</p>                                                                                                                                                                                                                                                            |
| <b>calculationSlack</b>  | <p>The available slack time between the foreground calculation time, and the scheduled txTime, in uS.</p>                                                                                                                                                                                                                   |
| <b>demandLatency</b>     | <p>The time to send SynqNet demand data downstream from controller to nodes, in uS. Does not include drive demand delays.</p>                                                                                                                                                                                               |
| <b>feedbackLatency</b>   | <p>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.</p>                                                                                                  |

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>latencySlack</b>   | The slack time from the configured feedbackLatency (which is rounded up to an integer multiple of the driveUpdatePeriod), to the minimum possible upstream delay, in uS.                                                                                                                                                                                                                                                                                                                                                                       |
| <b>controlLatency</b> | 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. |
| <b>synq</b>           | Downstream SYNQ packet payload, in uS. Includes spacing between packets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>demand</b>         | Downstream DEMAND packet payload, in uS. Includes spacing between packets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>control</b>        | Downstream CONTROL packet payload, in uS. Includes spacing between packets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>downstream</b>     | Total downstream (controller to nodes) packet payload, in uS. Includes spacing between packets. Breaks down into the following three packet types.                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>feedback</b>       | Upstream FEEDBACK packet payload, in uS. Includes spacing between packets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>status</b>         | Upstream STATUS packet payload, in uS. Includes spacing between packets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>upstream</b>       | Total upstream (nodes to controller) packet payload, in uS. Includes spacing between packets. Breaks down into the following two packet types.                                                                                                                                                                                                                                                                                                                                                                                                 |

## 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.

|                                   |                                                                                                                             |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>MEISynqNetTraceDYNA_ALLOC</b>  | Enables trace statements for controller external memory allocation during SynqNet initialization.                           |
| <b>MEISynqNetTraceDYNA_FREE</b>   | Enables trace statements for controller external memory de-allocation.                                                      |
| <b>MEISynqNetTraceINIT</b>        | Enables trace statements for SynqNet network initialization.                                                                |
| <b>MEISynqNetTraceRESET</b>       | Enables trace statements for SynqNet network reset.                                                                         |
| <b>MEISynqNetTraceTIMING</b>      | Enables trace statements for SynqNet network timing calculations.                                                           |
| <b>MEISynqNetTraceSERVICE_CMD</b> | Enables trace statements for SynqNet service command transactions between the controller and SynqNet nodes.                 |
| <b>MEISynqNetTraceFLOWCTRL</b>    | This trace bit enables trace statements for the SynqNet service command handshake between the controller and SynqNet nodes. |
| <b>MEISynqNetTraceMAP</b>         | 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 **XMPI\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\(...\)](#).

[Trace Masks](#) | [Global Trace Outputs](#) | [Per-Object Trace Outputs](#)

## Methods

### Configuration and Information Methods

|                                     |                                                         |
|-------------------------------------|---------------------------------------------------------|
| <a href="#">meiTraceEol</a>         | Set the end-of-line character to be used by Trace       |
| <a href="#">meiTraceFile</a>        | Send trace output to a file                             |
| <a href="#">meiTraceFunction</a>    | sets function used to display a trace buffer            |
| <a href="#">meiTraceGet</a>         | Get global trace mask                                   |
| <a href="#">meiTraceMaskBits</a>    | Convert the trace mask into an array of trace bits.     |
| <a href="#">meiTraceMsg</a>         | Convert the message identification value into a string. |
| <a href="#">meiTraceMsgFunction</a> | Set a module's trace message function.                  |
| <a href="#">meiTraceSet</a>         | Set global trace mask                                   |

## Data Types

[MEITrace](#)  
[MEITraceFunction](#)  
[MEITraceMask](#)  
[MEITraceMsgFunction](#)

## Constants

[MEITraceMaskGLOBAL](#)

# 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 <i>traceFunction</i> | if there is a previous function                                                    |
| NULL                              | if no <i>traceFunction</i> has been specified (the default trace function is used) |

## See Also

[meiPlatformTrace](#)

# 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.

### 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 " <i>traceMask</i> " 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](#)

# 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.

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <b>handle</b>     | a handle to an object                            |
| <b>*traceMask</b> | a pointer to the value of an object's trace mask |

### Return Values

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ObjectTraceGet</i> 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***.

|                  |                                     |
|------------------|-------------------------------------|
| <b>handle</b>    | a handle to an object               |
| <b>traceMask</b> | the value of an object's trace mask |

### Return Values

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <b>MPIMessageOK</b> | if <i>ObjectTraceSet</i> 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](#)

# 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.

|                  |                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>mask</b>      | A bit mask of enumerated trace bits.                                                                                            |
| <b>*bitCount</b> | 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.    |
| <b>*bit</b>      | 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*.

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <b>messageId</b>    | a message identification value.                                        |
| <b>*messageText</b> | 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.

|                 |                                           |
|-----------------|-------------------------------------------|
| <b>moduleId</b> | an enumerated module identification value |
| <b>function</b> | 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.

|                                |                                                                  |
|--------------------------------|------------------------------------------------------------------|
| <b>MEITraceFUNCTION_ENTRY</b>  | Trace the entry into all methods.                                |
| <b>MEITraceFUNCTION_RETURN</b> | Trace the return from all methods.                               |
| <b>MEITraceMEMORY_ALLOC</b>    | Enables trace statements for all host memory allocations.        |
| <b>MEITraceMEMORY_FREE</b>     | Enables trace statements for all host memory de-allocations.     |
| <b>MEITraceMEMORY_GET</b>      | Enables trace statements for all controller memory reads.        |
| <b>MEITraceMEMORY_SET</b>      | Enables trace statements for all controller memory writes.       |
| <b>MEITraceVALIDATE</b>        | Enables trace statements for all function parameter validations. |
| <b>MEITraceLOCK_GIVE</b>       | Enables trace statements for all IPC lock releases.              |
| <b>MEITraceLOCK_TAKE</b>       | Enables trace statements for all IPC lock takes.                 |
| <b>MEITraceEVENT</b>           | 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<br>(for MPI objects) | declared in stdmei.h.                                                                                                             |
| object-specific<br>(for MEI objects) | declared in the object header file (for MEI objects). Note that the trace mask bits for object-specific trace categories overlap. |

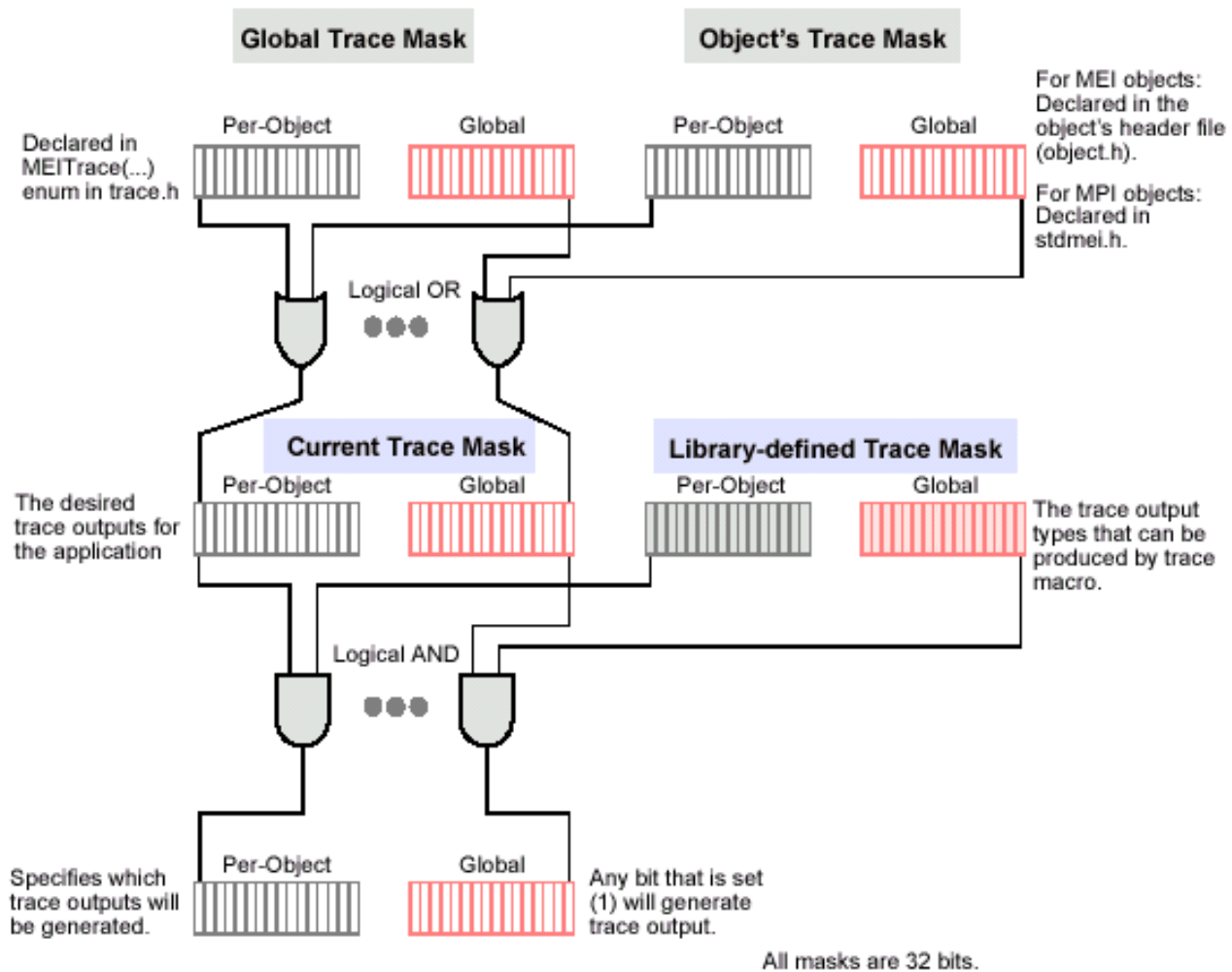
An object will produce trace output for a trace category when the logical **OR** of the **global trace mask** and the **object's trace mask** has the bit set that corresponds to the trace category.

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:

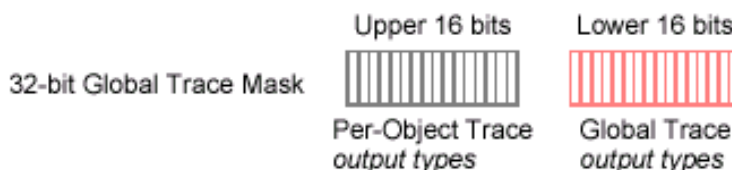
1. The **global trace mask** is logically **ORed** with the **object trace mask**. This yields the **current trace mask**, representing the desired trace output types as specified by the application.
2. The **current trace mask** (from step1) is logically **ANDed** with a **library-defined trace mask** (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

[MEIXmpSwitchType](#)

# MEIXmpSwitchType

## Declaration

```
typedef enum {
    MEIXmpSwitchTypeNONE,
    MEIXmpSwitchTypeMOTION_ONLY,
    MEIXmpSwitchTypeWINDOW,
    MEIXmpSwitchTypeUSER,
} MEIXmpSwitchType;
```

**Required Header:** stdmpi.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](#))**

|                                    |                                                                                                                                                                                                                  |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEIFilterGainIndexNO_MOTION</b> | When command velocity = 0                                                                                                                                                                                        |
| <b>MEIFilterGainIndexACCEL</b>     | When command acceleration > 0                                                                                                                                                                                    |
| <b>MEIFilterGainIndexDECEL</b>     | When command acceleration < 0                                                                                                                                                                                    |
| <b>MEIFilterGainIndexVELOCITY</b>  | 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](#)