SqNode Objects

Introduction

A **SqNode** object manages a single SynqNet network node connected to a SynqNet network. It represents the physical network node. It contains information about the node, as well as its status and configuration. It provides read/write access to the node via network cyclic data and service commands. It also provides an interface to any drives connected to the node.

During network initialization, the SynqNet nodes are discovered and mapped to the SynqNet object. The number of motors per SqNode is determined and mapped to the controller's motor objects. Each node connected to a controller is assigned a number (0, 1, 2, etc) in the order it is discovered. The node number is used to index the SqNode objects.

| Error Messages |

Methods

Create, Delete, Validate Methods

meiSqNode**Create**

meiSqNode**Delete**

meiSqNode**Validate**

Configuration and Information Methods

rmbAnalogInRangeGet

rmbAnalogInRangeSet

meiSqNode**Command**

meiSqNode**ConfigGet**

meiSqNode**ConfigSet**

meiSqNode**FlashConfigGet**

meiSqNode**FlashConfigSet**

meiSqNode**FpgaDefaultFileName**

meiSqNode**FpgaFilenameVerify**

meiSqNode**Info**

meiSqNode**Status**

meiSqNode**UserDataGet**

meiSqNode**UserDataSet**

Drive Interface Methods

meiSqNode**DriveConfigGet**

meiSqNode**DriveConfigSet**

meiSqNode**DriveInfo**
meiSqNode**DriveMapParamCount**
meiSqNode**DriveMapParamList**
meiSqNode**DriveMapConfigCount**
meiSqNode**DriveMapConfigList**
meiSqNode**DriveMapParamFileGet**
meiSqNode**DriveMapParamFileSet**
meiSqNode**DriveMonitor**
meiSqNode**DriveMonitorInfo**
meiSqNode**DriveMonitorConfigGet**
meiSqNode**DriveMonitorConfigSet**
meiSqNode**DriveParamCalculate**
meiSqNode**DriveParamClear**
meiSqNode**DriveParamGet**
meiSqNode**DriveParamListGet**
meiSqNode**DriveParamListSet**
meiSqNode**DriveParamReload**
meiSqNode**DriveParamRestore**
meiSqNode**DriveParamSet**
meiSqNode**DriveParamStore**

I/O Methods

meiSqNode**AnalogIn**
meiSqNode**AnalogInPtr**
meiSqNode**AnalogOutPtr**
meiSqNode**AnalogOutGet**
meiSqNode**AnalogOutSet**
meiSqNode**DigitalIn**
meiSqNode**DigitalInPtr**
meiSqNode**DigitalOutPtr**
meiSqNode**DigitalOutGet**
meiSqNode**DigitalOutSet**
meiSqNode**SegmentAnalogIn**
meiSqNode**SegmentAnalogOutGet**
meiSqNode**SegmentAnalogOutSet**
meiSqNode**SegmentInfo**
meiSqNode**SegmentDigitalIn**
meiSqNode**SegmentDigitalOutGet**
meiSqNode**SegmentDigitalOutSet**

meiSqNode**SegmentMemoryGet**
meiSqNode**SegmentMemorySet**
meiSqNode**SegmentParamDefault**
meiSqNode**SegmentParamStore**
meiSqNode**SegmentParamClear**
meiSqNode**SegmentParamGet**
meiSqNode**SegmentParamSet**
meiSqNode**SegmentUserDataGet**
meiSqNode**SegmentUserDataSet**

Action Methods

meiSqNode**Download**
meiSqNode**FlashErase**
meiSqNode**FpgaFileNameVerify**
meiSqNode**NetworkObjectNext**
meiSqNode**StatusClear**
meiSqNode**Verify**

Event Methods

meiSqNode**EventNotifyGet**
meiSqNode**EventNotifySet**
meiSqNode**EventReset**

Memory Methods

meiSqNode**Memory**
meiSqNode**MemoryGet**
meiSqNode**MemorySet**

Relational Methods

meiSqNode**Control**
meiSqNode**Number**

Data Types

RMBAnalogInRange

MEISqNode**Channel**
MEISqNode**CmdHeader**

MEISqNode**StatusIoFaults**

MEISqNode**StatusPacketError**

MEISqNode**UserData**

Constants

MEISqNode**ConfigControlLatencyMIN_LIMIT**

MEISqNode**ConfigControlLatencyMAX_LIMIT**

MEISqNode**Drive_Param_MAX_STRING_LENGTH**

MEISqNode**ID_CHAR_MAX**

MEISqNode**FILENAME_MAX**

MEISqNode**ManufacturerDATA_CHAR_MAX**

MEISqNode**MaxFEEDBACK_SECONDARY**

MEISqNode**MaxMOTORS**

MEISqNode**NOT_AVAILABLE**

MEISqNode**SEGMENT_MAX**

MEISqNode**SEGMENT_PARAMS_MAX**

MEISqNode**SEGMENT_MEMORY_MAX**

MEISqNode**STATUS_NOT_AVAILABLE**

MEISqNode**UserDATA_CHAR_MAX**

MEISqNode**SegmentInfoMANUFACTURER_LENGTH**

MEISqNode**SegmentInfoMODEL_NAME_LENGTH**

MEISqNode**SegmentInfoSERIAL_NUMBER_LENGTH**

MEISqNode**SegmentUserDATA_CHAR_MAX**

MEI**DriveMapParamMAX_STRING_LENGTH**

MEI**FPGARINCONREV**

MEI**FpgaSqMACVersionDEFAULT**

MEI**FpgaSqMACVersionMIN**

MEI**FpgaSqMACVersionMAX**

MEI**FpgaSqNodeVersionDEFAULT**

MEI**FpgaSqNodeVersionMIN**

MEI**FpgaSqNodeVersionMAX**

meiSqNodeCreate

Declaration

```
MEISqNode meiSqNodeCreate(MPIControl  control,
                          long        number)
```

Required Header: stdmei.h

Description

meiSqNodeCreate creates a SqNode object identified by ***number***, which is associated with a control object.

SqNodeCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	an index to the SqNode. The first node number is 0, the second is 1, etc.

Return Values	
handle	to a SqNode object. After creating a SqNode object it must be validated using meiSqNodeValidate(...).
MPIHandleVOID	if the object could not be created

See Also

meiSqNodeDelete | meiSqNodeValidate

meiSqNodeDelete

Declaration

```
long meiSqNodeDelete(MEISqNode  node);
```

Required Header: stdmei.h

Description

meiSqNodeDelete deletes a SqNode object and invalidates its handle.

SqNodeDelete is the equivalent of a C++ destructor.

node	a handle of the SqNode object to delete in the reverse order to avoid memory leaks.

Return Values	
MPIMessageOK	

See Also

meiSqNodeCreate | meiSqNodeValidate

meiSqNodeValidate

Declaration

```
long meiSqNodeValidate(MEISqNode  node);
```

Required Header: stdmei.h

Description

meiSqNodeValidate validates a SqNode object and its handle.

SqNodeValidate is the equivalent of a C++ constructor.

node	a handle to a SynqNet node object.

Return Values	
MPIMessageOK	

See Also

meiSqNodeCreate | meiSqNodeDelete

rmbAnalogInRangeGet

Declaration

```
long rmbAnalogInRangeGet(MEISqNode          sqNode,
                         long               channel,
                         RMBAnalogInRange   *analogInRange);
```

Required Header: stdmei.h

Description

The analog inputs on RMB-10V2 nodes support software configurable input ranges. The **rmbAnalogInRangeGet** function allows the current input range for each channel to be read.

sqNode	a handle to a SynqNet node object.
channel	the index of the analogue input channel.
***analogInRange**	a pointer to where the current input slice will be written by this function.

Return Values	
MPIMessageOK	

See Also

rmbAnalogInRangeSet | Analog Inputs on RMB Nodes

rmbAnalogInRangeSet

Declaration

```
long rmbAnalogInRangeSet(MEISqNode          sqNode,
                         long               channel,
                         RMBAnalogInRange   analogInRange);
```

Required Header: stdmei.h

Description

The analog inputs on RMB-10V2 nodes support software configurable input ranges. The **rmbAnalogInRangeSet** function allows you to change the input range for each channel.

sqNode	a handle to a SynqNet node object.
channel	the index of the analog input channel.
analogInRange	the new analogue input range.

Return Values	
MPIMessageOK	

See Also

rmbAnalogInRangeGet | Analog Inputs on RMB Nodes

meiSqNodeCommand

Declaration

```
long meiSqNodeCommand(MEISqNode         node,
                      MEISqNodeCommand  *command,
                      MEISqNodeResponse *response);
```

Required Header: stdmei.h

Description

meiSqNodeCommand sends a service command to a SynqNet node using the data from the structure pointed to by command and writes the response into the structure pointed to by response. Service commands occur across the SynqNet network through a service channel. In SYNQ mode there is one service channel for each node. In ASYNQ mode there is one service channel for all nodes. The controller sends the command and waits for a response using a 4 state handshake. In SYNQ mode the service command data is sent through the cyclic packets, but due to the handshaking, it is not considered a cyclic operation.

For SynqNet nodes that have drive memory interfaces, service commands can be sent to drives. Also, the service commands supports access to drive data memory, program memory, I/O memory, and direct commands. Please see the MEISqNodeCommand{.} and MEISqNodeResponse{.} structures for more information. And be sure to consult the drive's header file in the (C:\MEI)\XMP\sqNodeLib \include directory, as well as, the drive manufacturer's manual for valid drive addresses.

node	a handle to a SynqNet node object
***command**	a pointer to a SynqNet node command structure
***response**	a pointer to a SynqNet node response structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEISqNodeCommand | MEISqNodeResponse | MEISqNodeCmdHeader | MEISqNodeCmdType | MEISqNodeDataSize | MEISqNodeMemory | MEISqNodeChannel

meiSqNodeConfigGet

Declaration

```
long meiSqNodeConfigGet(MEISqNode        node,
                        MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeConfigGet reads a SynqNet node's (***node***) configuration and writes it into the structure pointed to by ***config***.

node	a handle to a SynqNet node object.
***config**	a pointer to a SynqNet node config structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveConfigGet

meiSqNodeConfigSet

Declaration

```
long meiSqNodeConfigSet(MEISqNode        node,
                        MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeConfigSet writes a SynqNet node's (***node***) configuration using data from the structure pointed to by ***config***.

node	a handle to a SynqNet node object
***config**	a pointer to a SynqNet node config structure

Return Values	
MPIMessageOK	
MEIMessageARG_INVALID	
MEISqNodeMessageFEEDBACK_MAP_INVALID	

See Also

meiSqNodeInfo | meiSqNodeDriveConfigSet | meiSynqNetFlashTopologySave

meiSqNodeFlashConfigGet

Declaration

```
long meiSqNodeFlashConfigGet(MEISqNode        node,
                             void             *flash,
                             MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeFlashConfigGet reads a SynqNet node's flash configuration and writes it into the structure pointed to by ***config***.

node	a handle to a SynqNet node object.
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***config**	a pointer to a SynqNet node config structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeFlashConfigSet

meiSqNodeFlashConfigSet

Declaration

```
long meiSqNodeFlashConfigSet(MEISqNode        node,
                             void             *flash,
                             MEISqNodeConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeFlashConfigSet sets a SynqNet Node (***node***) flash configuration using data from the structure pointed to by ***config***.

NOTE: The network topology must first be saved before changing node config values in Flash memory. These values will also be cleared when network topology is cleared using meiSynqNetFlashTopologyClear(...).

node	a handle to a SynqNet node object.
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***config**	a pointer to a SynqNet node config structure.

Return Values	
MPIMessageOK	
MEISqNodeMessageFEEDBACK_MAP_INVAILD	
MPIMessageARG_INVALID	
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	

See Also

meiSqNodeFlashConfigGet | Flash Objects | meiSynqNetFlashTopologySave

meiSqNodeFpgaDefaultFileName

Declaration

```
long meiSqNodeFpgaDefaultFileName(MEISqNode           sqNode,
                                  MEISqNodeFileName   *fileName);
```

Required Header: stdmei.h

Description

meiSqNodeFpgaDefaultFileName provides the default image filename for an sqNode.

sqNode	handle to a SqNode object.
***fileName**	a pointer to a structure that has space allocated to hold an FPGA filename.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSqNodeFpgaFileNameVerify

Declaration

```
long meiSqNodeFpgaFileNameVerify(MEISqNode      sqNode,
                                 const  char*   fileName);
```

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

Description

meiSqNodeFpgaFileNameVerify verifies that the filename provided is compatible with a given sqNode.

sqNode	a handle to an SqNode object.
fileName	a pointer to a string containing the name of an SqNode image file.

Return Values	
MPIMessageOK	
MEISqNodeMessageFILE_NODE_MISMATCH	

See Also

meiSqNodeInfo

Declaration

```
long meiSqNodeInfo(MEISqNode       node,
                   MEISqNodeInfo   *info);
```

Required Header: stdmei.h

Description

meiSqNodeInfo reads a SynqNet node's information and writes it into a structure pointed to by ***info***. The info structure contains read only data about the node.

The RMB-10V, RMB-10V2 and some Trust nodes support analog inputs. MPI support has been added to support the reading of node-based analog inputs. The number of analog inputs a node supports can be determined with meiSqNodeInfo(...). An analog input value can be read with meiSqNodeAnalogIn(...). The analog to digital converted value is scaled from -1.0 to +1.0, where +1.0 is a full-scale positive voltage. The input range of the ADC is hardware-specific.

node	a handle to a SynqNet node object.
***info**	a pointer to a drive specific information structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

Sample Code

```
MEISqNodeInfo sqNodeInfo;
meiSqNodeInfo( sqNode0, &sqNodeInfo );

long x = sqNodeInfo.io.digitalInputCount;
```

See Also

meiSqNodeDriveInfo | meiSqNodeConfigGet | meiSqNodeDriveConfigGet

meiSqNodeStatus

Declaration

```
long meiSqNodeStatus(MEISqNode          node,
                     MEISqNodeStatus    *status);
```

Required Header: stdmei.h

Description

meiSqNodeStatus reads status from the ***node*** associated with the SynqNet object and writes it into the structure pointed to by ***status***. The SynqNet node status structure contains error counters and event mask data.

NOTE: This data requires service commands to acces the data on the node. As a result, it may take up to 9 servo cycles to read the data. At the default sample rate of 2kHz, this would translate to 4.5ms.

node	a handle to a SynqNet node object.
***status**	pointer to a SynqNet status structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSynqNetStatus | meiSqNodeInfo

meiSqNodeUserDataGet

Declaration

```
long meiSqNodeUserDataGet(MEISqNode          node,
                          MEISqNodeUserData  *data);
```

Required Header: stdmei.h

Description

meiSqNodeUserDataGet reads the user data from the node.

node	a handle to a SynqNet node object.
***data**	a pointer to a MEISqNodeUserData structure, allocated on the host.

Return Values	
MPIMessageOK	

See Also

meiSqNodeUserDataSet | Accessing Digital Data

meiSqNodeUserDataSet

Declaration

```
long meiSqNodeUserDataSet(MEISqNode          node,
                          MEISqNodeUserData  *data);
```

Required Header: stdmei.h

Description

meiSqNodeUserDataSet writes the user data to the node.

node	a handle to a SynqNet node object.
***data**	a pointer to a MEISqNodeUserData structure, allocated on the host.

Return Values	
MPIMessageOK	

See Also

meiSqNodeUserDataGet | MEISqNodeUserData | Accessing Digital Data

meiSqNodeDriveConfigGet

Declaration

```
long meiSqNodeDriveConfigGet(MEISqNode  node,
                             long       driveIndex, /* relative to the node */
                             void       *config);   /* node specific        */
```

Required Header: stdmei.h

Description

meiSqNodeDriveConfigGet reads a SynqNet node's drive configuration and writes it into a drive specific structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive configuration can be read if the drive interface hardware supports a communication channel to the drive processor. The drive interface (s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The drive configuration structure is drive specific. The SqNodeLib includes the drive specific structures and methods. Please see the drive's header file in the (C:\MEI)\XMP\sqNodeLib\include directory, as well as, the drive manufacturer's documentation for details. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and its type.

node	a handle to a SynqNet node object
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive specific configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo | meiSqNodeConfigGet

meiSqNodeDriveConfigSet

Declaration

```
long meiSqNodeDriveConfigSet(MEISqNode  node,
                             long       driveIndex, /* relative to the node */
                             void       *config);   /* node specific       */
```

Required Header: stdmei.h

Description

meiSqNodeDriveConfigSet writes a SynqNet node's drive configuration from a drive specific structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive configuration can be written if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The drive configuration structure is drive specific. The SqNodeLib includes the drive specific structures and methods. Please see the drive manufacturer's documentation for details. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and its type.

node	a handle to a SynqNet node object
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive specific configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo | meiSqNodeConfigSet

meiSqNodeDriveInfo

Declaration

```
long meiSqNodeDriveInfo(MEISqNode           node,
                        long                driveIndex, /* relative to the node */
                        MEISqNodeDriveInfo  *info,
                        void                *external);  /* node specific */
```

Required Header: stdmei.h

Description

meiSqNodeDriveInfo reads a SynqNet node's drive information and writes it into a drive specific structure pointed to by ***info***. The drive info structure contains read only data. SynqNet nodes may support one or more drive interfaces. The drive information can be read if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The drive information structure is drive specific. The SqNodeLib includes the drive specific structures and methods. Please see the drive's header file for the drive specific information structures, as well as, the drive manufacturer's documentation for details. All supported drive header files are located in the (C:\MEI)\XMP\sqNodeLib\include directory. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and its type.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***info**	a pointer to a structure that contains general drive information.
***external**	a pointer to a drive specific information structure. See the appropriate drive vendor *.h for definition. (NOTE: it can be NULL)

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeConfigGet | meiSqNodeDriveConfigGet

meiSqNodeDriveMapParamCount

Declaration

```
long meiSqNodeDriveMapParamCount (MEISqNode     sqNode,
                                  MEIDriveMap   driveMap,
                                  long          driveIndex,
                                  long          *paramsCount);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamCount scans the drive map file for a drive entry that matches this node on the network. If an entry is found, then this function returns the number of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiSqNodeDriveMapParamList(...) function. First, this function is called in order to get the size of the drive parameter list. Then the user can use this size to allocate enough memory to hold the complete parameter list before calling meiSqNodeDriveMapParamList(...) to fill in the list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***paramsCount**	pointer to the variable that will be set by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamList

meiSqNodeDriveMapParamList

Declaration

```
long meiSqNodeDriveMapParamList (MEISqNode          sqNode,
                                 MEIDriveMap        driveMap,
                                 long               driveIndex,
                                 long               paramsCount,
                                 MEIDriveParamInfo  *driveParamInfo);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamList scans the drive map file for an entry that matches the node on the network. If a drive entry is found, this function writes the drive parameter information about each of the drive parameters to the ***driveParamInfo*** list.

This function is normally used with the meiSqNodeDriveMapParamCount(...) function. The meiSqNodeDriveMapParamCount(...) function is called first to get the size of the parameter list, the user can then use this size to allocate enough memory to hold the complete parameter list before calling this function to fill in the parameter list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
paramsCount	the number of drive parameter information records that can be written to the *driveParamInfo* list.
***driveParamInfo**	pointer to the list of drive parameter information records that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamCount

meiSqNodeDriveMapConfigCount

Declaration

```
long meiSqNodeDriveMapConfigCount (MEISqNode     sqNode,
                                   MEIDriveMap   driveMap,
                                   long          driveIndex,
                                   long          *configCount);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapConfigCount scans the drive map file for a drive entry that matches this node on the network. If an entry is found, this function returns the number of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiSqNodeDriveMapConfigList function. This function is first called to get the size of the drive configuration list. Then the user can use this size to allocate enough memory to hold the complete configuration list before calling meiSqNodeDriveMapConfigList(...) to fill in the list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***configCount**	pointer to the variable that will be set by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapConfigList

meiSqNodeDriveMapConfigList

Declaration

```
long meiSqNodeDriveMapConfigList (MEISqNode     sqNode,
                                  MEIDriveMap   driveMap,
                                  long          driveIndex,
                                  long          configCount,
                                  long          *configList);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapConfigList scans the drive map file for a drive entry that matches this node on the network. If an entry is found, this function returns the list of drive parameters that need to be preserved for the configuration of the drive.

This function is normally used with the meiSqNodeDriveMapConfigCount(...) function. The meiSqNodeDriveMapConfigCount(...) function is first called to get the size of the drive configuration list. Then the user can use this size to allocate enough memory to hold the complete configuration list before calling this function to fill in the list.

sqNode	a handle to a SynqNet node object.
driveMap	a handle to a DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
configCount	the number of drive parameter information records that can be written to the *configList* list.
***configList**	pointer to the list of drive parameters that make up the drive configuration that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapConfigCount

meiSqNodeDriveMapParamFileGet

Declaration

```
long meiSqNodeDriveMapParamFileGet(MEISqNode     sqNode,
                                   MEIDriveMap   driveMap,
                                   long          driveIndex,
                                   char          *driveConfigFilename,
                                   long          append);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamFileGet saves the current set of drive parameters in the drive to the ***driveConfigFilename***.

sqNode	a handle to the SynqNet node object.
driveMap	a handle to the DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***driveConfigFilename**	the name of the file that holds the stored drive configuration file.
append	1 = The new data is appended to the existing drive configuration file if it exists. 0 = A new drive configuration file is created to hold the drive parameters. If a file already exists, it will be overwritten.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamFileSet

SynqNet Drive Parameters

meiSqNodeDriveMapParamFileSet

Declaration

```
long meiSqNodeDriveMapParamFileSet(MEISqNode                  sqNode,
                                   MEIDriveMap                driveMap,
                                   long                       driveIndex,
                                   char                       *driveConfigFilename,
                                   MEISqNodeDriveParamCallback callback,
                                   long                       warning);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMapParamFileSet loads the drive parameters stored in the file named ***driveConfigFilename*** into the drive.

sqNode	a handle to the SynqNet node object.
driveMap	a handle to the DriveMap object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***driveConfigFilename**	the name of the file that holds the stored drive configuration file.
callback	A callback function that this function calls to indicate if the function is changing the value of a drive parameter or setting a new drive parameter that has failed. Passing NULL for this parameter will disable the callback feature.
warning	0 = if setting a drive parameter fails, this function will fail immediately. 1 = if setting a drive parameter fails, then the function will continue with the remaining drive parameters and generate a warning by calling the callback function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveMapParamFileGet

SynqNet Drive Parameters

meiSqNodeDriveMonitor

Declaration

```
long meiSqNodeDriveMonitor(MEISqNode              node,
                           long                   driveIndex, /* relative
to
                                                              the node */
                           MEISqNodeMonitorValue  *value);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitor reads the monitor fields from the drive and writes them into the structure pointed to by ***value***.

node	a handle to a SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
***value**	pointer to a structure of monitor values

Return Values	
MPIMessageOK	

See Also

MEISqNodeMonitorValue

meiSqNodeDriveMonitorInfo

Declaration

```
long  meiSqNodeDriveMonitorInfo(MEISqNode                  node,
                                long                       driveIndex,
                                MEISqNodeDriveMonitorInfo  *info);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitorInfo reads a SynqNet node drive's monitor information and writes it into a structure pointed to by ***info***. The ***info*** structure contains read only data about the number of supported drive monitor fields, their locations, and how to decode them.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***info**	a pointer to a MEISqNodeDriveMonitorInfo structure. It contains information about the number of monitor fields, their locations, and how to decode them.

Return Values	
MPIMessageOK	

See Also

MEISqNodeDriveMonitorInfo

meiSqNodeDriveMonitorConfigGet

Declaration

```
long meiSqNodeDriveMonitorConfigGet(MEISqNode                node,
                                    long                     driveIndex, /* relative to
                                                                            the node */
                                    MEISqNodeDriveMonitorConfig *config);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitorConfigGet reads a SynqNet node's drive monitor configuration and writes it into a structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive monitor configuration can be read if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The SynqNet network packets have some extra fields that can be configured to read drive data every sample. Each monitor field is 32 bits. SynqNet nodes with drive interfaces that support drive monitoring can be configured to transmit the data. The drive manufacturer determines what data is available for monitoring. The monitor data can be specified by a predetermined index or memory address. Please see the drive's header file for the drive specific configuration structures, as well as, the drive manufacturer's documentation for details. All supported drive header files are located in the (C:\MEI)\XMP\sqNodeLib\include directory.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive monitor configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo

meiSqNodeDriveMonitorConfigSet

Declaration

```
long meiSqNodeDriveMonitorConfigSet(MEISqNode  node,
                                    long       driveIndex, /* relative to
                                                              the node */
                  MEISqNodeDriveMonitorConfig  *config);
```

Required Header: stdmei.h

Description

meiSqNodeDriveMonitorConfigSet writes a SynqNet node's drive monitor configuration from a structure pointed to by ***config***. SynqNet nodes may support one or more drive interfaces. The drive monitor configuration can be written if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The SynqNet network packets have some extra fields that can be configured to read drive data every sample. Each monitor field is 32 bits. SynqNet nodes with drive interfaces that support drive monitoring can be configured to transmit the data. The drive manufacturer determines what data is available for monitoring. The monitor data can be specified by a predetermined index or memory address. Please see the drive's header file for the drive specific configuration structures, as well as, the drive manufacturer's documentation for details. All supported drive header files are located in the (C:\MEI)\XMP\sqNodeLib \include directory.

node	a handle to a SynqNet node object.
driveIndex	an index to a drive interface on a SynqNet node. The first drive interface is 0, the second is 1, etc.
***config**	a pointer to a drive monitor configuration structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSqNodeDriveInfo

meiSqNodeDriveParamCalculate

Declaration

```
long meiSqNodeDriveParamCalculate(MEISqNode    sqNode,
                                  long         driveIndex);
```

Required Header: stdmei.h

Description

Some drives need to calculate some internal quantities after a drive parameter has been changed. The **meiSqNodeDriveParamCalculate** function will instruct the drive to calculate its internal quantities. This feature is not supported or required by all drives.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamClear

Declaration

```
long meiSqNodeDriveParamClear(MEISqNode    sqNode,
                              long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamClear clears the previously saved drive by loading the default set of drive parameters into the current and non-volatile storage on the drive. These drive parameters will be used each time this drive is subsequently started (after a power-on or network reset). The default drive parameters will take effect immediately.

NOTE: This function may not be supported by all drives. The default set of drive parameters may be different between different drive types and different drive manufactures.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamReload

meiSqNodeDriveParamGet

Declaration

```
long meiSqNodeDriveParamGet(MEISqNode                node,
                            long                     driveIndex,
                            long                     param,
                            MEIDriveMapParamType     paramType,
                            MEIDriveMapParamValue    *value);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamGet reads a drive parameter from the drive and fills in the appropriate field of the union pointed to by ***value***. The ***paramType*** defines the type of data that is read from the drive and also defines which field will be used in the ***value*** union.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
param	an index for the drive parameter that is being accessed.
paramType	the type of the data read from the drive and which field will be used in the *value* union.
***value**	a pointer to the union that will be filled in.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamSet

meiSqNodeDriveParamListGet

Declaration

```
long meiSqNodeDriveParamListGet(MEISqNode                node,
                                long                     driveIndex,
                                long                     size,
                                long                     *paramList,
                                MEIDriveMapParamType     *paramTypes,
                                MEIDriveMapParamValue    *paramValues);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamListGet reads a series of drive parameters from the drive and fills in the appropriate fields of the unions pointed to by ***paramValues***. The ***paramTypes*** defines the type of data that is read from the drive and also defines which fields in the ***paramValues*** unions are going to be used.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
size	the number of drive parameters to be read.
***paramList**	a pointer to a list of the drive parameter indexes that are being accessed.
***paramTypes**	a pointer to a list of drive parameter types to be read from the drive and which field in the paramValues union is going to be used.
***paramValues**	a pointer to a list of unions that will be filled in by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamListSet

meiSqNodeDriveParamListSet

Declaration

```
long meiSqNodeDriveParamListSet(MEISqNode                node,
                                long                     driveIndex,
                                long                     size,
                                long                     *paramList,
                                MEIDriveMapParamType     *paramTypes,
                                MEIDriveMapParamValue    *paramValues);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamListSet writes a series of drive parameters pointed to by ***value*** to the drive. The ***paramTypes*** defines each type of data item that is written to the drive and also defines which field in the ***paramValues*** unions are going to be used.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
size	the number of drive parameters to be written.
***paramList**	a pointer to a list of drive parameter types to be written to the drive and which field in the paramValues union is going to be used.
***paramTypes**	a pointer to a list of the drive parameter indexes that are being accessed.
***paramValues**	a pointer to a list of unions that will be written to the drive.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamListGet

meiSqNodeDriveParamReload

Declaration

```
long meiSqNodeDriveParamReload(MEISqNode    sqNode,
                               long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamReload overwrites the current set of drive parameters with the set from the non-volatile storage on the drive. These new drive parameters will take effect immediately.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamClear

meiSqNodeDriveParamRestore

Declaration

```
long meiSqNodeDriveParamRestore(MEISqNode    sqNode,
                                long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamRestore loads the default set of drive parameters into current set of drive parameters on the drive. The default drive parameters will take effect immediately.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamStore

meiSqNodeDriveParamSet

Declaration

```
long meiSqNodeDriveParamSet(MEISqNode                node,
                            long                     driveIndex,
                            long                     param,
                            MEIDriveMapParamType     paramType,
                            MEIDriveMapParamValue    *value);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamSet writes the drive parameter that is pointed to by ***value*** to the drive. The ***paramType*** defines the type of data that is written to the drive and also defines which field will be used in the ***value*** union.

node	a handle to the SynqNet node object.
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.
param	an index for the drive parameter that is being accessed.
paramType	the type of data being written to the drive and which field will be used in the *value* union.
***value**	pointer to the union that will be written to the drive.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamGet

meiSqNodeDriveParamStore

Declaration

```
long meiSqNodeDriveParamStore(MEISqNode    sqNode,
                              long         driveIndex);
```

Required Header: stdmei.h

Description

meiSqNodeDriveParamStore saves the drive parameters into non-volatile storage on the drive. These drive parameters will be used each time the drive is subsequently started (after a power-on or network reset).

NOTE: This function may not be supported by all drives.

sqNode	a handle to a SynqNet node object
driveIndex	an index to the drive (0, 1, 2, etc), relative to the node.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDriveParamRestore

meiSqNodeAnalogIn

Declaration

```
long meiSqNodeAnalogIn(MEISqNode     node,
                       long          channel,
                       long          *state);
```

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

Description

meiSqNodeAnalogIn gets the current state of an analog input on a SynqNet node.

NOTE: meiSqNodeAnalogIn replaced meiSqNodeAnalogInput in the MPI library.

node	a handle to a SynqNet node object.
channel	the index of the analog input channel (with respect to the node).
***state**	a pointer to where the current state of the input is written by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code reads the current state of the first analog input on Node 2:

```
long x;
meiSqNodeAnalogIn( sqNode2, 0, &x );
```

See Also

Accessing Analog Data | MEISqNodeInfoIo

meiSqNodeAnalogInPtr

Declaration

```
long meiSqNodeAnalogInPtr(MEISqNode   sqNode,
                          long        channel,
                          long        *mask,
                          long        **ptr);
```

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

Description

meiSqNodeAnalogInPtr gets a controller memory pointer and mask of an analog input (**channel** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

sqNode	a handle to a SynqNet node object.
channel	an index to a node's analog input channel.
***mask**	a pointer to a bitwise mask that corresponds to the specified word.
****ptr**	a pointer to a controller address that contains the analog input word.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record analog channel 2 on node 1.

```
long mask;
long * pointer;
meiSqNodeAnalogInPtr( sqNode1, 2, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalInPtr
meiSqNodeDigitalOutPtr
meiSqNodeAnalogOutPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeAnalogOutPtr

Declaration

```
long meiSqNodeAnalogOutPtr(MEISqNode   sqNode,
                           long        channel,
                           long        *mask,
                           long        **ptr);
```

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

Description

meiSqNodeAnalogOutPtr gets a controller memory pointer and mask of an analog output (**channel** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

Analog values are held as 16 bit words either in the upper or lower two bytes of a controllers memory location hence the mask obtained from this function will either be 0xFFFF0000 or 0x0000FFFF.

sqNode	a handle to a SynqNet node object.
channel	an index to a node's analog output channel.
***mask**	a pointer to a bitwise mask that corresponds to the specified word.
****ptr**	a pointer to a controller address that contains the analog output word.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record analog output channel 2 on node 1.

```
long mask;
long * pointer;
meiSqNodeAnalogOutPtr( sqNode1, 2, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalInPtr
meiSqNodeDigitalOutPtr
meiSqNodeAnalogInPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeAnalogOutGet

Declaration

```
long meiSqNodeAnalogOutGet(MEISqNode     node,
                           long          channel,
                           long          *state);
```

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

Description

meiSqNodeAnalogOutGet reads the current state of an analog output on a SynqNet node.

node	a handle to a SynqNet node object.
channel	the index of the analog output channel (with respect to the node).
***state**	a pointer to where the current state of the output is written by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeAnalogSet | Accessing Analog Data | MEISqNodeInfoIo

meiSqNodeAnalogOutSet

Declaration

```
long meiSqNodeAnalogOutSet(MEISqNode     node,
                           long          channel,
                           long          state,
                           MPI_BOOL      wait);
```

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

Description

meiSqNodeAnalogOutSet changes the current state of an analog output on a SynqNet node.

node	a handle to a SynqNet node object.
channel	the index of the analog output channel (with respect to the node).
state	the desired state of the analog output.
wait	determines what happens if two output functions are called in short succession. See Overview of Motor I/O: Output Waits.

Return Values	
MPIMessageOK	

See Also

meiSqNodeAnalogOutGet | Accessing Analog Data | MEISqNodeInfoIo

meiSqNodeDigitalIn

Declaration

```
long meiSqNodeDigitalIn(MEISqNode      node,
                        long           bitStart,
                        long           bitCount,
                        unsigned long  *state );
```

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

Description

meiSqNodeDigitalIn gets the current state of multiple digital inputs on the specified SynqNet node.

node	a handle to a SynqNet node object.
bitStart	the first bit.
bitCount	the number of bits to be read.
***state**	a pointer to where the current state of the input bits is written to by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code will get the current state of digital inputs 3, 4, and 5 of Node 0 (in the example network these bits spread over segments 0 and 1):

```
long x;
meiSqNodeDigitalIn( sqNode0, 3, 3, &x );
```

See Also

Accessing Digital I/O | SQIO-DIN32DOUT32: Input Pinouts

meiSqNodeDigitalInPtr

Declaration

```
long meiSqNodeDigitalInPtr(MEISqNode   sqNode,
                           long        bit,
                           long        *mask,
                           long        **ptr);
```

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

Description

meiSqNodeDigitalInPtr gets a controller memory pointer and mask of a digital input (**bit** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

sqNode	a handle to a SynqNet node object.
bit	an index to a node's digital input bit.
***mask**	a pointer to a bitwise mask that corresponds to the specified bit.
****ptr**	a pointer to a controller address that contains the digital input bit.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record bit 5 on node 1.

```
long mask;
long * pointer;
meiSqNodeDigitalInPtr( sqNode1, 5, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalOutPtr
meiSqNodeAnalogInPtr
meiSqNodeAnalogOutPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeDigitalOutPtr

Declaration

```
long meiSqNodeDigitalOutPtr(MEISqNode   sqNode,
                            long        bit,
                            long        *mask,
                            long        **ptr);
```

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

Description

meiSqNodeDigitalOutPtr gets a controller memory pointer and mask of a digital output (**bit** on the SynqNet node **sqNode**) and writes them to the locations pointed to by **ptr** and **mask**. The memory pointer and mask obtained by this function can be used when setting up the data recorder or user limits.

sqNode	a handle to a SynqNet node object.
bit	an index to a node's digital input bit.
***mask**	a pointer to a bitwise mask that corresponds to the specified bit.
****ptr**	a pointer to a controller address that contains the digital output bit.

Return Values	
MPIMessageOK	
MPIMessageaARG_INVALID	

Sample Code

The sample code below shows how to set up the data recorder to record bit 5 on node 1.

```
long mask;
long * pointer;
meiSqNodeDigitalOutPtr( sqNode1, 5, &mask, &pointer );

void * points[MEIXmpMaxRecSize];
points[0] = pointer;
long pointCount = 1;
mpiRecorderRecordConfig(
   recorder,
   MPIRecorderRecordTypePOINT,
   pointCount,
   points );
```

See Also

meiSqNodeDigitalInPtr
meiSqNodeAnalogInPtr
meiSqNodeAnalogOutPtr

MEIMotorEventConfig
mpiRecorderRecordConfig

meiSqNodeDigitalOutGet

Declaration

```
long meiSqNodeDigitalOutGet(MEISqNode       node,
                            long            bitStart,
                            long            bitCount,
                            unsigned  long  *state);
```

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

Description

meiSqNodeDigitalOutGet reads the current state of the multiple digital output bits on the specified SynqNet node.

node	a handle to a SynqNet node object.
bitStart	the first bit.
bitCount	the number of bits to be read.
***state**	a pointer to where the current state of the output bits is written to by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeDigitalOutSet | Accessing Digital I/O | SQIO-DIN32DOUT32: Output Pinouts

meiSqNodeDigitalOutSet

Declaration

```
long meiSqNodeDigitalOutSet(MEISqNode       node,
                            long            bitStart,
                            long            bitCount,
                            unsigned  long  state,
                            MPI_BOOL        wait);
```

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

Description

meiSqNodeDigitalOutSet changes the state of multiple digital outputs on the specified SynqNet node.

node	a handle to a SynqNet node object.
bitStart	the first bit.
bitCount	the number of bits to be changed.
state	a pointer to where the current state of the output bits is written to by this function.
wait	Boolean flag indicating if the new output state is applied immediately or if a wait is inserted so that any previous output set is transmitted over SynqNet and applied to the output before this function. You should be able to use TRUE for this argument in most applications.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code will set the top 16 bits and clear the bottom 16 bits of Node 1:

```
long x = 0xFFFF0000;
meiSqNodeDigitalOutSet( sqNode1, 0, 32, &x, TRUE );
```

The following MPI code will only set the fifth bit on a node 0 (the second bit of the segment 1):

```
meiSqNodeDigitalOutSet( sqNode0, 5, 1, 1, TRUE );
```

The following MPI code with ***wait*** true will ensure that the physical pin will definitely generate a short pulse.

```
meiSqNodeDigitalOutSet( sqNode1, 0, 1, 1, /*wait*/ 1 );
meiSqNodeDigitalOutSet( sqNode1, 0, 1, 0, /*wait*/ 1 );
```

The following MPI code shows the use of not introducing a ***wait***. This code is updating two bits on the same node. Both calls will immediately exit and since the timing of these two bits in this application are not related, this code can execute faster.

```
meiSqNodeDigitalOutSet( sqNode1, 7, 1, 1, /*wait*/ 0 );
meiSqNodeDigitalOutSet( sqNode1, 8, 1, 1, /*wait*/ 0 );
```

See Also

meiSqNodeDigitalOutGet | Accessing Digital I/O | SQIO-DIN32DOUT32: Output Pinouts

meiSqNodeSegmentAnalogIn

Declaration

```
long meiSqNodeSegmentAnalogIn(MEISqNode    node,
                              long         segment,
                              long         channel,
                              long         *state);
```

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

Description

meiSqNodeSegmentAnalogIn gets the current state of an analog input on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
channel	the index of the analog input channel (with respect to the slice). For more information, please see the Overview of MPI I/O.
***state**	a pointer to where the current state of the input is written by this function.

Return Values	
MPIMessageOK	

See Also

Accessing Analog Data | MEISqNodeInfoIo | Overview of MPI I/O

meiSqNodeSegmentAnalogOutGet

Declaration

```
long meiSqNodeSegmentAnalogOutGet(MEISqNode    node,
                                  long         segment,
                                  long         channel,
                                  long         *state);
```

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

Description

meiSqNodeSegmentAnalogOutGet gets the current state of an analog output on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
channel	the index of the analog output channel (with respect to the slice). For more information, please see the Overview of MPI I/O.
***state**	a pointer to where the current state of the output is written by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeSegmentAnalogOutSet | Accessing Analog Data | MEISqNodeInfoIo

Overview of MPI I/O

meiSqNodeSegmentAnalogOutSet

Declaration

```
long meiSqNodeSegmentAnalogOutSet(MEISqNode    node,
                                  long         segment,
                                  long         channel,
                                  long         state,
                                  MPI_BOOL     wait);
```

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

Description

meiSqNodeSegmentAnalogOutSet changes the current state of an analog output on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
channel	the index of the analog input channel (with respect to the slice). For more information, please see the Overview of MPI I/O.
state	the desired state of the analog output.
wait	determines what happens if two output functions are called in short succession. See Overview of Motor I/O: Output Waits.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code sets the analog output Number 2 on Segment 1 of Node 1 to +5V.

```
meiSqNodeSegmentAnalogOutSet( sqNode1, 1, 2, 0x3FFF );
```

See Also

meiSqNodeSegmentAnalogOutGet | Accessing Analog Data | MEISqNodeInfoIo

Overview of MPI I/O

meiSqNodeSegmentInfo

Declaration

```
long meiSqNodeSegmentInfo(MEISqNode              node,
                          long                   segment,
                          MEISqNodeSegmentInfo   *info);
```

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

Description

meiSqNodeSegmentInfo reads the constant data about a segment on a SynqNet node and fills in the structure pointer by the ***info*** argument.

node	a handle to a SynqNet node object
segment	the index of the slice / module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
***info**	a pointer to a structure that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code shows how to read the number of digital inputs on Slice 1 of Node 0.

```
MEISqNodeSegmentInfo segmentInfo;
meiSqNodeSegmentInfo( sqNode0, 1, &segmentInfo );

long x = segmentInfo.digitalInCount;
```

See Also

MEISqNodeSegmentInfo | What Information is Available About Each I/O Segment? | Overview of MPI I/O

meiSqNodeSegmentDigitalIn

Declaration

```
long meiSqNodeSegmentDigitalIn(MEISqNode      node,
                               long           segment,
                               long           bitStart,
                               long           bitCount,
                               unsigned long  *state);
```

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

Description

meiSqNodeSegmentDigitalIn gets the current state of multiple digital inputs on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
bitStart	the first bit
bitCount	the number of bits to be read.
***state**	a pointer to a long word that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code will get the current state of all the digital inputs (the following slice has 8 inputs) on segment 1 of node 0:

```
long x;
meiSqNodeSegmentDigitalIn( sqNode0, 1, 0, 8, &x );
```

See Also

Accessing Digital I/O

Overview of MPI I/O

meiSqNodeSegmentDigitalOutGet

Declaration

```
long meiSqNodeSegmentDigitalOutGet(MEISqNode     node,
                                   long          segment,
                                   long          bitStart,
                                   long          bitCount,
                                   unsigned long *state);
```

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

Description

meiSqNodeSegmentDigitalOutGet changes the state of multiple digital outputs on the specified slice on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
bitStart	the first bit
bitCount	the number of bits to be read.
state	a pointer to where the current state of the output bits is written to by this function.

Return Values	
MPIMessageOK	

See Also

meiSqNodeSegmentDigitalOutSet | Accessing Digital I/O | Overview of MPI I/O

meiSqNodeSegmentDigitalOutSet

Declaration

```
long meiSqNodeSegmentDigitalOutSet(MEISqNode      node,
                                   long           segment,
                                   long           bitStart,
                                   long           bitCount,
                                   unsigned long  state,
                                   MPI_BOOL       wait);
```

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

Description

meiSqNodeSegmentDigitalOutSet sets the current state of the multiple digital output bits on the specified slice/module on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
bitStart	the first bit
bitCount	the number of bits to be read.
state	a pointer to where the current state of the output bits is written to by this function.
wait	a Boolean flag indicating if the new output state is applied immediately or a wait is inserted so that any previous output set is transmitted over SynqNet. You should be able to use TRUE for this argument in most applications.

Return Values	
MPIMessageOK	

See Also

meiSqNodeSegmentDigitalOutGet | Accessing Digital I/O | Overview of MPI I/O

meiSqNodeSegmentMemoryGet

Declaration

```
long  meiSqNodeSegmentMemoryGet(MEISqNode  sqNode,
                                long       segment,
                                long       memoryStart,
                                long       memoryCount,
                                char       *data);
```

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

Description

meiSqNodeSegmentMemoryGet gets the current value of the specified slice memory registers from the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
memoryStart	the first memory register to get.
memoryCount	the number of memory registers to get.
***data**	a pointer to the array of memory registers that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

Here is the code to read the first ten memory parameters on slice 1.

```
char memory[10];
meiSqNodeSegementMemoryGet( sqNode, 1, 0, 10, memory );
```

See Also

Overview of MPI I/O | meiSqNodeSliceMemorySet

meiSqNodeSegmentMemorySet

Declaration

```
long  meiSqNodeSegmentMemorySet(MEISqNode  sqNode,
                                long       segment,
                                long       memoryStart,
                                long       memoryCount,
                                char       *data);
```

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

Description

meiSqNodeSegmentMemorySet changes the current value of the specified slice memory registers on the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
memoryStart	the first memory register to set.
memoryCount	the number of memory registers to set.
***data**	a pointer to the array of memory registers that be written to the segment by this function.

Return Values	
MPIMessageOK	

Sample Code

Here is the code read the first 2 memory parameters on slice 1.

```
char memory[10];
meiSqNodeSegementMemoryGet( sqNode, 1, 0, 10, memory );
```

See Also

Overview of MPI I/O | meiSqNodeSliceMemoryGet

meiSqNodeSegmentParamDefault

Declaration

```
long  meiSqNodeSegmentParamDefault(MEISqNode  sqNode);
```

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

Description

meiSqNodeSegmentParamDefault overwrites the current set of parameters with the default set of parameters.

sqNode	a handle to a SqNode object

Return Values	
MPIMessageOK	

Sample Code

```
meiSqNodeSegementMemorySet( sqNode );
```

See Also

Overview of MPI I/O | meiSqNodeSegmentParamStore | meiSqNodeSegmentParamClear

meiSqNodeSegmentParamStore

Declaration

```
long  meiSqNodeSegmentParamStore(MEISqNode  sqNode);
```

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

Description

meiSqNodeSegmentParamStore copies the current working set of parameters to the set of stored parameters held in non-volatile memory. This set of parameters will subsequently be used from a node power-on or reset.

sqNode	a handle to a SqNode object.

Return Values	
MPIMessageOK	

Sample Code

```
meiSqNodeSegementMemoryStore( sqNode );
```

See Also

Overview of MPI I/O | meiSqNodeSegmentParamDefault | meiSqNodeSegmentParamClear

meiSqNodeSegmentParamClear

Declaration

```
long  meiSqNodeSegmentParamClear(MEISqNode  sqNode);
```

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

Description

meiSqNodeSegmentParamClear function will delete any slice parameters previously stored on non-volatile memory. The next time the node powers up, the default parameters will be loaded on the slices.

sqNode	a handle to a SqNode object.

Return Values	
MPIMessageOK	

Sample Code

```
meiSqNodeSegementMemoryClear( sqNode );
```

See Also

Overview of MPI I/O | meiSqNodeSegmentParamStore | meiSqNodeSegmentParamDefault

meiSqNodeSegmentParamGet

Declaration

```
long  meiSqNodeSegmentParamGet(MEISqNode  sqNode,
                               long       segment,
                               long       paramStart,
                               long       paramCount,
                               char       *params);
```

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

Description

meiSqNodeSegmentParamGet gets the current value of the specified slice parameters from the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
paramStart	the first parameter to get.
paramCount	the number of parameters to get.
***params**	a pointer to the array of parameters that will be filled in by this function.

Return Values	
MPIMessageOK	

Sample Code

For example to get the first two slice parameters on slice 1.

```
char parameters[2];
meiSqNodeSegmentParamGet( sqNode0, 1, 0, 2, parameters );
```

See Also

meiSqNodeSliceParamSet | Overview of MPI I/O

meiSqNodeSegmentParamSet

Declaration

```
long  meiSqNodeSegmentParamSet(MEISqNode  sqNode,
                               long       segment,
                               long       paramStart,
                               long       paramCount,
                               char       *params);
```

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

Description

meiSqNodeSegmentParamSet gets the current value of the specified slice parameters from the sqNode.

sqNode	a handle to a SqNode object.
segment	the index of the slice / module attached to this SynqNet node.
paramStart	the first parameter to set.
paramCount	the number of parameters to set.
***params**	a pointer to the array of parameters that be written to the segment by this function.

Return Values	
MPIMessageOK	

Sample Code

For example to get the first two slice parameters on slice 1.

```
char parameters[2]; = { '\x00', '\x0F' };
meiSqNodeSegmentParamSet( sqNode0, 1, 0, 2, parameters );
```

See Also

Overview of MPI I/O | meiSqNodeSliceParamGet

meiSqNodeSegmentUserDataGet

Declaration

```
long meiSqNodeSegmentUserDataGet(MEISqNode                  node,
                                 long                       segment,
                                 MEISqNodeSegmentUserData   *data);
```

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

Description

Modules attached to a SQID node have a small section of non-volatile memory that can be used for any purpose by the user. **meiSqNodeSegmentUserDataGet** gets the user data from a segment on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
data	a pointer to where the user data is written by this function.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code reads the contents on the non-volatile memory on segment 0 of node 1:

```
MEISqNodeSegmentUserData nodeData;

meiSqNodeSegmentUserDataGet( sqNode1, 0, &nodeData );
```

See Also

meiSqNodeSegmentUserDataSet | MPI Overview I/O: User Data | Overview of MPI I/O

meiSqNodeSegmentUserDataSet

Declaration

```
long meiSqNodeSegmentUserDataSet(MEISqNode                  node,
                                 long                       segment,
                                 MEISqNodeSegmentUserData   *data);
```

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

Description

Modules attached to a SQID node have a small section of non-volatile memory that can be used for any purpose by the user. **meiSqNodeSegmentUserDataSet** changes the user data stored on a segment on a SynqNet node.

node	a handle to a SynqNet node object.
segment	the index of the slice/module attached to this SynqNet node. For more information, please see the Overview of MPI I/O.
data	a pointer to the new user data.

Return Values	
MPIMessageOK	

Sample Code

The following MPI code changes the contents on the non-volatile memory to a string:

```
MEISqNodeSegmentUserData nodeData;

strncpy( nodeData.data, “test” , MEISqNodeUserDATA_CHAR_MAX );

meiSqNodeSegmentUserDataSet( sqNode1, 0,&nodeData );
```

See Also

meiSqNodeSegmentUserDataGet | MPI Overview I/O: User Data | Overview of MPI I/O

meiSqNodeDownload

Declaration

```
long meiSqNodeDownload(MEISqNode                  node,
                       MEISqNodeDownloadParams    *params);
```

Required Header: stdmei.h

Description

meiSqNodeDownload SqNodeDownload reads a binary image from a file and writes it into a SynqNet node's non-volatile storage. SynqNet nodes may support one or more drive interfaces. SqNodeDownload can also write binary images to a drives' non-volatile storage if the drive interface hardware supports a communication channel to the drive processor. The drive interface(s) for a SynqNet node are indexed by a number (0, 1, 2, etc.).

The SynqNet node binary files are node specific. Please see the Node Binary Files: Product Table.

The SynqNet drive binary files are drive specific. The SqNodeLib includes the drive specific code necessary to support various hardware download protocols. Please see the drive manufacturer's documentation for details. Use meiSqNodeInfo(...), to determine if the SynqNet node supports a drive interface and it's type.

The binary download process requires a significant amount of time, probably between 5 to 30 seconds, depending on the node/drive type and file size. A callback function pointer is provided in the MEISqNodeDownloadParams structure for the calling application to monitor the download progress.

node	a handle to a SynqNet node object
***params**	a pointer to the download parameters structure.

Return Values	
MPIMessageOK	

See Also

meiSqNodeInfo | meiSynqNetInfo | MEISqNodeDownloadParams | MEISqNodeChannel | MEISqNodeCallback

meiSqNodeFlashErase

Declaration

```
long meiSqNodeFlashErase(MEISqNode     sqNode);
```

Required Header: stdmei.h

Description

meiSqNodeFlashErase brings the SynqNet network down to discovery mode, sends a service command down to the node that erases its runtime flash, and leaves the network down. The next time the network is brought up to Synq mode the node will be running off its boot image.

node	a handle to a SynqNet node object.

Return Values	
MPIMessageOK	

See Also

meiSqNodeNetworkObjectNext

Declaration

```
long meiSqNodeNetworkObjectNext(MEISqNode              node,
                                MEINetworkPort         port,
                                MEINetworkObjectInfo   *info);
```

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

Description

meiSqNodeNetworkObjectNext gets the information for the neighboring network object (device) connected to the specified port and writes the information into the structure pointed to by ***info***.

NOTE: This info.type value may be MEINetworkObjectTypeNONE if there is nothing connected to the given port.

node	a handle to a SynqNet node object.
port	specifies the node's IN or OUT port.
***info**	a pointer to the next object's info structure.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

meiSynqNetNetworkObjectNext | MEINetworkObjectInfo

Version Utility

meiSqNodeStatusClear

Declaration

```
long meiSqNodeStatusClear(MEISqNode     node);
```

Required Header: stdmei.h

Description

meiSqNodeStatusClear clears node CRC errors on all ports, clears node Packet errors, clears node ioAbort state, and resets SqNode events.

node	a handle to a SynqNet node object

Return Values	
MPIMessageOK	

See Also

MEIEventTypeSQNODE

meiSqNodeVerify

Declaration

```
long meiSqNodeVerify(MEISqNode                  node,
                     MEISqNodeDownloadParams    *params);
```

Required Header: stdmei.h

Description

meiSqNodeVerify verifies that the runtime image on a sqNode matches the data contained in a provided image file.

node	a handle to SqNode object.
***params**	a pointer to parameters used in the verify routine.

Return Values	
MPIMessageOK	
MEISqNodeMessageVERIFY_FAIL	

See Also

meiSqNodeEventNotifyGet

Declaration

```
long meiSqNodeEventNotifyGet(MEISqNode       node,
                             MPIEventMask    *eventMask,
                             void            *external);
```

Required Header: stdmei.h

Description

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

Use the event mask macros mpiEventMaskGET(...), mpiEventMaskBitGET(...), etc. to decode the eventMask.

The event notification data in external is in addition to the event notification data in eventMask. If external is NULL, the event notification data will not be copied to the external pointer.

Remarks

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

node	a handle to a SynqNet node object
***eventMask**	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
***external**	pointer to external

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MPI/MEIEventType | MEIEventNotifyData | MEIEventStatusInfo

meiSqNodeEventNotifySet

Declaration

```
long meiSqNodeEventNotifySet(MEISqNode       node,
                             MPIEventMask    eventMask,
                             void            *external);
```

Required Header: stdmei.h

Description

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

Use the event mask macros meiEventMaskSQNODE(...), mpiEventMaskSET(...), mpiEventMaskBitSET(...), mpiEventMaskCLEAR(...), etc. to create the eventMask.

The event notification data in external is in addition to the event notification data in eventMask. If external is NULL, the event notification data will not be copied to the external pointer.

Remarks

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

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

node	a handle to a SynqNet node object
eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
***external**	pointer to external

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEI/MPIEventType | MEIEventNotifyData | MEIEventStatusInfo

meiSqNodeEventReset

Declaration

```
long meiSqNodeEventReset(MEISqNode      sqNode,
                         MPIEventMask   eventMask);
```

Required Header: stdmei.h

Description

meiSqNodeEventReset is a method used to reset events that have been latched on a node. Events that can be reset by this method include:

See MEIEventType.

/* SqNode events */
MEIEventTypeSQNODE_IO_ABORT,
MEIEventTypeSQNODE_NODE_DISABLE,
MEIEventTypeSQNODE_NODE_ALARM,
MEIEventTypeSQNODE_ANALOG_POWER_FAULT,
MEIEventTypeSQNODE_USER_FAULT,
MEIEventTypeSQNODE_NODE_FAILURE,

sqNode	a handle to a SynqNet node object
eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

mpiControlEventReset | mpiMotionEventReset | mpiMotorEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | mpiAxisEventReset

Event Notification Methods

meiSqNodeMemory

Declaration

```
long meiSqNodeMemory(MEISqNode     node,
                     void          **memory);
```

Required Header: stdmei.h

Description

meiSqNodeMemory writes an address (that can be used to access SqNode memory) to the contents of memory. This address (or an address calculated from it) can be passed as the src argument to mpiSqNodeMemoryGet(...) or the dst argument to mpiSqNodeMemorySet(...).

node	a handle to a SynqNet node object
****memory**	a pointer to an SqNode memory address.

Return Values	
MPIMessageOK	

See Also

meiSqNodeMemoryGet | meiSqNodeMemorySet

meiSqNodeMemoryGet

Declaration

```
long meiSqNodeMemoryGet(MEISqNode     node,
                        void          *dst,
                        const void    *src,
                        long          count);
```

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

Description

meiSqNodeMemoryGet reads count bytes of an SqNode's memory, starting from address ***src*** and writes it to application memory, starting at address ***dst***.

node	a handle to a SynqNet node object
***dst**	pointer to the destination address in application memory
***src**	pointer to the source address in SqNode memory
count	number of bytes to copy

Return Values	
MPIMessageOK	

See Also

meiSqNodeMemory | meiSqNodeMemorySet

meiSqNodeMemorySet

Declaration

```
long meiSqNodeMemorySet(MEISqNode    node,
                        void         *dst,
                        const  void  *src,
                        long         count);
```

Required Header: stdmei.h

Description

meiSqNodeMemorySet reads count bytes of application memory, starting from address ***src*** and writes it to an SqNode's memory, starting at address ***dst***.

node	a handle to a SynqNet node object
***dst**	pointer to the destination address in SqNode memory
***src**	pointer to the source address in application memory
count	number of bytes to copy

Return Values	
MPIMessageOK	

See Also

meiSqNodeMemory | meiSqNodeMemoryGet

meiSynqNetControl

Declaration

```
MPIControl meiSqNodeControl(MEISqNode node);
```

Required Header: stdmei.h

Description

meiSqNodeControl returns a handle to the control object associated with the SqNode object.

node	a handle to a SynqNet node object

Return Values	
MPIControl	a handle to a control object
MPIHandleVOID	if node is not valid

See Also

meiSqNodeCreate | mpiControlCreate

meiSqNodeNumber

Declaration

```
long meiSqNodeNumber(MEISqNode    node,
                     long         *number);
```

Required Header: stdmei.h

Description

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

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

Return Values	
MPIMessageOK	

See Also

meiSynqNetInfo | meiSynqNetNumber

RMBAnalogInRange

Definition

```
typedef enum {
    RMBAnalogInRange10V,
    RMBAnalogInRange5V,
    RMBAnalogInRange2_5V,
    RMBAnalogInRange1_25V,
} RMBAnalogInRange;
```

Change History: Added in the 03.02.00

Description

RMBAnalogInRange enumeration has the four analog input ranges that can be configured on RMB_10V2 nodes.

RMBAnalogInRange10V	The analog voltage range is within ±10V
RMBAnalogInRange5V	The analog voltage range is within ±5V
RMBAnalogInRange2_5V	The analog voltage range is within ±2.5V
RMBAnalogInRange1_25V	The analog voltage range is within ±1.25V

See Also

Analog Inputs on RMB Nodes

MEISqNodeChannel

Definition

```
typedef enum MEISqNodeChannel {
    MEISqNodeChannelDRIVE0,
    MEISqNodeChannelDRIVE1,
    MEISqNodeChannelDRIVE2,
    MEISqNodeChannelDRIVE3,
    MEISqNodeChannelDRIVE4,
    MEISqNodeChannelDRIVE5,
    MEISqNodeChannelDRIVE6,
    MEISqNodeChannelDRIVE7,
    MEISqNodeChannelNODE,
} MEISqNodeChannel;
```

Description

MEISqNodeChannel is an enumeration of communication interfaces to a node. All SynqNet nodes support a single NODE channel to the network interface device. SynqNet nodes may support one or more drive channels to a drive processor. DRIVE channels are indexed by an enumeration (DRIVE0, DRIVE1, DRIVE2, etc.).

MEISqNodeChannelDRIVE0	interface to drive number 0
MEISqNodeChannelDRIVE1	interface to drive number 1
MEISqNodeChannelDRIVE2	interface to drive number 2
MEISqNodeChannelDRIVE3	interface to drive number 3
MEISqNodeChannelDRIVE4	interface to drive number 4
MEISqNodeChannelDRIVE5	interface to drive number 5
MEISqNodeChannelDRIVE6	interface to drive number 6
MEISqNodeChannelDRIVE7	interface to drive number 7
MEISqNodeChannelNODE	interface to the node device

See Also

meiSqNodeDownload

MEISqNodeCmdHeader

Definition

```
typedef struct MEISqNodeCmdHeader {
    MEISqNodeChannel    channel;    /* internal node destination */
    MEISqNodeMemory     memory;
    MEISqNodeDataSize   size;
    MEISqNodeCmdType    type;       /* read/write command */
} MEISqNodeCmdHeader;
```

Description

MEISqNodeCmdHeader specifies the service command communication interface to the device, the memory region on the device to access, the data size, and type.

channel	Communication interface to a device. See MEISqNodeChannel.
memory	The memory region to access. See MEISqNodeMemory.
size	The length of data to send or receive. See MEISqNodeDataSize.
type	The service command action (read or write). See MEISqNodeCmdType.

See Also

MEISqNodeCommand

MEISqNodeCmdType

Definition

```
typedef enum MEISqNodeCmdType {
    MEISqNodeCmdTypeREAD,
    MEISqNodeCmdTypeWRITE,
} MEISqNodeCmdType;
```

Description

MEISqNodeCmdType is an enumeration of service command types to send to a node or drive.

MEISqNodeCmdTypeREAD	read data
MEISqNodeCmdTypeWRITE	write data

See Also

MEISqNodeCmdHeader

MEISqNodeCommand

Definition

```
typedef struct MEISqNodeCommand {
    MEISqNodeCmdHeader  header;
    unsigned long       address;  /* command resgisters */
    unsigned long       data;     /* command data */
} MEISqNodeCommand;
```

Description

MEISqNodeCommand specifies the service command. It includes a header structure (channel, memory, size, and type), a destination address, and the data.

header	A structure that specifies the channel, memory region, and data size. See MEISqNodeCmdHeader.
address	A memory location to read or write the data.
data	The command data to send.

See Also

MEISqNodeResponse

MEISqNodeConfig

Definition

```
typedef struct MEISqNodeConfig {
    char                                userLabel[MEIObjectLabelCharMAX
+1];
                                        /* +1 for NULL terminator */
    MEISqNodeConfigAlarm                nodeAlarm;
    MEISqNodeConfigIoAbort              ioAbort;
    MEISqNodeConfigPacketError          upStreamError;
    MEISqNodeConfigPacketError          downStreamError;
    MEISqNodeConfigUserFault            userFault;

    MEISqNodeConfigControlLatency       controlLatency;
    MEISqNodeFeedbackSecondary          feedbackSecondary
                                        [MEISqNodeMaxFEEDBACK_SECONDARY];
} MEISqNodeConfig;
```

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

Description

MEISqNodeConfig specifies the SynqNet node configurations.

userLabel	Consists of 16 characters that are used to label the sqNode object for user identification purposes. The userLabel field is NOT used by the controller.
nodeAlarm	A structure to configure a SynqNet node's trigger conditions for the Node Alarm output bit. The node alarm circuit is node specific, but is intended to notify users when the node has a problem. The nodeAlarm occurs on an ioAbort, DedicatedInAMP_FAULT (one per motor/drive) or an FPGA fails to operate with run-time code. See MEISqNodeConfigIoAbort and MEISqNodeConfigNodeAlarm for the trigger configurations.
ioAbort	A structure to configure a SynqNet node's trigger conditions for an I/O Abort action. When an ioAbort is triggered, the SynqNet node's outputs are disabled (set to the power-on condition). **NOTE**: The outputs are disabled on all SynqNet partner nodes, with the exception of the remote motion block's (rmb's) user outputs. For the rmb only, a user limit must be configured to disable the rmb's user output when an ioAbort occurs. See MEISqNodeConfigIoAbort for the trigger configurations.

upStreamError	A structure used to configure the fault and failure limits for the upstream SynqNet packets. The controller keeps track of how many bad packets are received from the Node and performs the appropriate actions when the fault and fail limits are reached. See MEISqNodeConfigPacketError for appropriate ranges and resulting actions. **NOTE**: Saving the upStreamError values to non-volatile flash memory is currently not supported. These values need to be set after each controller reset or power on.
downStreamError	A structure used to configure the fault and failure limits for the downstream SynqNet packets. The node keeps track of how many bad packets are received from the controller and performs the appropriate actions when the fault and fail limits are reached. See MEISqNodeConfigPacketError for appropriate ranges and resulting actions.
userFault	A structure to configure the trigger conditions for a SynqNet node user fault. When a user fault is triggered, a node ioAbort and/or an action on each motor will occur. See MEISqNodeConfigUserFault for the trigger configurations.
feedbackSecondary	A structure to configure the secondary encoder resources on the node. See MEISqNodeFeedbackSecondary for more information.
controlLatency	MEISqNodeConfigControlLatency structure to configure the minimum and maximum control latency limits.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet | MEISqNodeConfigPacketError

MEISqNodeConfigAlarm

Definition

```
typedef struct MEISqNodeConfigAlarm {
    unsigned long mask;  /* One bit per drive/motor. Triggered by
                            the MEIMotorDedicatedInAMP_FAULT input. */
    MPI_BOOL      notCyclicEnable;  /* allow nodeAlarm to be asserted
when
                                       the node is not in cyclic mode */
    MPI_BOOL      ioAbortEnable;  /* allow ioAbort to assert nodeAlarm */
    MPI_BOOL      ioFaultEnable;  /* allow ioFault to assert nodeAlarm */
} MEISqNodeConfigAlarm;
```

Change History: Modified in the 03.04.00.

Description

MEISqNodeConfigAlarm specifies the input trigger for the SynqNet node alarm output. The input triggers are the MEIMotorDedicatedInAMP_FAULT bits for each motor/drive interface.

mask	Each bit in the mask represents a motor or drive interface. For example, a value of 0x3 will trigger the node alarm output when either motor 0's OR motor 1's MEIMotorDedicatedInAMP_FAULT bit is TRUE.
notCyclicEnable	This Boolean variable is used to specify whether or not a node can receive an alarm when it is not in cyclic mode. TRUE = node alarm can be asserted in any mode. FALSE = node alarm can only be asserted in cyclic mode.
ioAbortEnable	This Boolean variable is used to specify the effect an I/O abort will have on the node alarm output. TRUE = an I/O abort will trigger a node alarm. FALSE = an I/O abort will not trigger a node alarm.
ioFaultEnable	This Boolean variable is used to specify the effect an I/O fault will have on the node alarm output. TRUE = an I/O fault will trigger a node alarm. FALSE = an I/O fault will not trigger a node alarm.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet

MEISqNodeConfigControlLatency

Definition

```
typedef struct MEISqNodeConfigControlLatency {
   double minimum;  /* uS  - if > 0, this will insure
                       a minimum latency value */
   double maximum;  /* uS  - if > 0, an error will occur
                       if this value is exceeded */
} MEISqNodeConfigControlLatency;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeConfigControlLatency is a structure that contains boundary values that ensure that the node's Control Latency falls within a given range. The SynqNet configuration can be read with meiSqNodeConfigGet(...) and can be written with meiSqNodeConfigSet(...).

minimum	Lower limit for the control latency configuration. This will ensure that a minimum node control latency will be used in the SynqNet timing calculations. Set to zero to disable minimum node control latency.
maximum	Upper limit for the control latency configuration. This will force mpiControlInit(...) or meiSynqNetInit(...) to return MEISynqNetMessageNODE_LATENCY_EXCEEDED if this value is exceeded. Set to zero to disable maximum node control latency check.

See Also

MEISqNodeConfig | meiSqNodeConfigSet | meiSqNodeConfigGet | MEISynqNetTiming

Node Control Latency | Cable Length Discovery Uncertainty Factor | SynqNet Cable Length

MEISqNodeConfigIoAbort

Definition

```
typedef struct MEISqNodeConfigIoAbort {
  MEISqNodeConfigTrigger  synqLost;    /* communication error */
  MEISqNodeConfigTrigger  nodeDisable; /* external input */
  MEISqNodeConfigTrigger  powerFault;  /* analog power failure */
  MPI_BOOL                userFault;   /* TRUE = user fault causes ioabort */
} MEISqNodeConfigIoAbort;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeConfigIoAbort specifies the SynqNet node configurations to generate an I/O Abort action.

When an ioAbort is triggered, the SynqNet node's outputs are disabled (set to the power-on condition) and all axes on motion supervisors associated with the node are aborted and enter the error state.

NOTE: The outputs are disabled on all SynqNet partner nodes, with the exception of the remote motion block's (rmb's) user outputs. For the rmb only, a user limit must be configured to disable the rmb's user output when an ioAbort occurs.

When the I/O Abort conditions are cleared, the states of the axes may be cleared with a call to mpiMotionAction(..., MPIActionRESET). The IoAbort is triggered when any one or more of the following enabled configurations occur.

synqLost	Occurs when a SynqNet node drops out of SYNQ (cyclic) mode to SYNQ_LOST mode. See MEISqNodeConfigTrigger.
nodeDisable	An input bit to the SynqNet node. The node disable circuit is node specific, but is intended to shutdown the node via the IoAbort. See MEISqNodeConfigTrigger.
powerFault	An input bit to the SynqNet node. The power fault circuit is node specific, but is usually connected to an analog power monitor. Typically, when the DAC power or other analog component power is either too high or drops below a threshold, the power fault is triggered. Please see the node/drive manufacturer's documentation for details. See MEISqNodeConfigTrigger.
userFault	A user configurable trigger condition. A value of TRUE enables the trigger, FALSE disables the trigger.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet | mpiMotionAction

MEISqNodeConfigPacketError

Definition

```
typedef struct MEISqNodeConfigPacketError {
    long faultLimit;    /* 1 - 255 */
    long failLimit;     /* 1 - 255 */
} MEISqNodeConfigPacketError;
```

Description

MEISqNodeConfigPacketError specifies the limit conditions for SynqNet node packet rate errors.

faultLimit	Packet error rate limit to generate a fault. When the faultLimit is reached, the node will attempt to recover by switching the port used for data transmission. Valid range is 1 to 255. The default value is 12, which allows for a normal packet rate. The value saturates at 255.
failLimit	Packet error rate limit to generate a failure. When the failLimit is reached, the node will drop to the SYNQ_LOST state and disable its outputs. Valid range is 1 to 255. The default value is 12, which allows for a normal packet rate. The value saturates at 255.

See Also

meiSqNodeConfigGet | meiSqNodeConfigSet

MEISqNodeConfigTrigger

Definition

```
typedef struct MEISqNodeConfigTrigger {
    MPI_BOOL   enable;
    MPI_BOOL   invert;
} MEISqNodeConfigTrigger;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeConfigTrigger specifies trigger configurations.

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

See Also

MEISqNodeConfigIoAbort

MEISqNodeConfigUserFault

Definition

```
typedef struct MEISqNodeConfigUserFault {
    long            *addr;    /* firmware addr */
    unsigned long   mask;
    unsigned long   pattern;
} MEISqNodeConfigUserFault;
```

Description

MEISqNodeConfigUserFault specifies the trigger conditions for a user defined input. The trigger condition can be configured for any controller address. When the masked value at the specified addr matches the pattern, the user fault is active. The user fault triggers a SynqNet node IoAbort if the userFault flag in MEISqNodeConfigIoAbort{} is enabled. The user fault also triggers an action for all the motors associated with the node. The userFaultAction is specified in the MEIMotorConfig{} structure.

***addr**	A pointer to a controller address.
mask	A bit mask ANDed with the value at the controller address.
pattern	A bit pattern compared to the masked value at the controller address. When the masked value equals the pattern, the user trigger is TRUE.

See Also

MEISqNodeConfigIoAbort | MEIMotorConfig | meiSqNodeConfigGet | meiSqNodeConfigSet

MEISqNodeDataSize

Definition

```
typedef enum MEISqNodeDataSize {  /* read/write data width */
    MEISqNodeDataSize8BIT,
    MEISqNodeDataSize16BIT,
    MEISqNodeDataSize24BIT,
    MEISqNodeDataSize32BIT,
} MEISqNodeDataSize
```

Description

MEISqNodeDataSize is an enumeration of service command data lengths. The data length is in units of bits.

MEISqNodeDataSize8BIT	8 bit data length
MEISqNodeDataSize16BIT	16 bit data length
MEISqNodeDataSize24BIT	24 bit data length
MEISqNodeDataSize32BIT	32 bit data length

See Also

meiSqNodeCommand | MEISqNodeCmdHeader

MEISqNodeDownloadParams

Definition

```
typedef struct MEISqNodeDownloadParams {
    char                *filename;
    MEISqNodeChannel    channel;
    MEISqNodeCallback   callback;
} MEISqNodeDownloadParams;
```

Description

MEISqNodeDownloadParams specifies the parameters for downloading a binary image to a SynqNet node.

***filename**	A pointer to a file name. The file contains a header and binary code/data. Files are node/drive specific. Please see the Node Binary Files: Product Table or the drive manufacturer's documentation for the drive binary files.
channel	A communication interface to a node's logic device or drive processor. See MEISqNodeChannel.
callback	A pointer to a callback function, to monitor the download progress. See MEISqNodeCallback.

See Also

meiSqNodeDownload

MEISqNodeDriveInfo

Definition

```
typedef struct MEISqNodeDriveInfo {
    char    firmwareVersion[MEISqNodeDriveParamMAX_STRING_LENGTH];
} MEISqNodeDriveInfo;
```

Description

MEISqNodeDriveInfo contains information about a specified drive.

firmwareVersion	A string containing drive firmware version information that is retrieved from the Drive Processor on the Node.

See Also

meiSqNodeDownload

MEISqNodeDriveMonitor

Definition

```
typedef struct MEISqNodeDriveMonitor {
    MEISqNodeDriveMonitorDataType    type;
    MEISqNodeDriveMonitorData        data;
} MEISqNodeDriveMonitor;
```

Description

MEISqNodeDriveMonitor specifies the data to be placed in the monitor field by the drive.

type	The drive data is selected by its type. See MEISqNodeDriveMonitorDataType.
data	The location of the drive data. See MEISqNodeDriveMonitorData.

See Also

MEISqNodeMonitorValue | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorConfig

Definition

```
typedef struct MEISqNodeDriveMonitorConfig {
    MEISqNodeDriveMonitor    monitorA;
    MEISqNodeDriveMonitor    monitorB;
    MEISqNodeDriveMonitor    monitorC;
}MEISqNodeDriveMonitorConfig;
```

Description

MEISqNodeDriveMonitorConfig specifies the configuration for the drive monitor fields.

monitorA	configuration for drive monitor A
monitorB	configuration for drive monitor B
monitorC	configuration for drive monitor C

See Also

MEISqNodeDriveMonitor | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorData

Definition

```
typedef union {
    long    index;     /* the values for these
                       parameters are drive specific    */
    long    address;   /* and can be found in the
                       appropriate drive modules */
} MEISqNodeDriveMonitorData;
```

Description

MEISqNodeDriveMonitorData specifies the location of the monitor data. Drive data can be specified by either an index or an address. The location is drive specific. Please see the drive manufacturer's documentation.

index	A drive specific value to select a monitor data field from a table.
address	A drive specific memory address to select the monitor data.

See Also

meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorDataType

Definition

```
typedef enum MEISqNodeDriveMonitorDataType {
    MEISqNodeDriveMonitorDataTypeINDEX,
    MEISqNodeDriveMonitorDataTypeADDRESS,
    MEISqNodeDriveMonitorDataTypeFIXED_CONFIG,
} MEISqNodeDriveMonitorDataType;
```

Change History: Modified in the 03.04.00; added MEISqNodeDriveMonitorDataTypeFIXED_CONFIG.

Description

MEISqNodeDriveMonitorDataType is an enumeration of monitor data selection types.

MEISqNodeDriveMonitorDataTypeINDEX	Select monitor data using an index to a table.
MEISqNodeDriveMonitorDataTypeADDRESS	Select monitor data using an address.
MEISqNodeDriveMonitorDataTypeFIXED_CONFIG	Monitor data is not selectable. The monitor configuration is hard coded by the node/drive.

See Also

meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeDriveMonitorInfo

Definition

```
typedef struct MEISqNodeDriveMonitorInfo{
   long                                monitorCount;
   MEISqNodeMonitorLocation            location[MEISqNodeMonitorValueIndexLAST];
   long                                configCount;
   const MEISqNodeMonitorConfigInfo    *configInfo; /* array of configuration
                                                       information of size
                                                       configCount, this
                                                       info is drive specific */
} MEISqNodeDriveMonitorInfo;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeDriveMonitorInfo is a structure that contains information about the monitor fields. This structure is useful for determining the number of supported monitor fields, their locations and how to decode them. It is also useful for determining the number of configurable monitor field data inputs and the reading the monitor data-specific information. The monitor field data is SynqNet drive-specific. Not all drives support monitors.

NOTE: SynqNet RMBs (Remote Motion Blocks) or I/O nodes do not support monitors.

monitorCount	the number of supported monitor fields.
location	an array of MEISqNodeMonitorLocation structures, with length equal to the monitorCount. Each MEISqNodeMonitorLocation structure contains information to decode each monitor field.
configCount	the number of configurable monitor field data inputs.
***configInfo**	a pointer to an array of MEISqNodeMonitorConfigInfo structures, with length equal to **configCount**. Each MEISqNodeMonitorConfigInfo structure contains information about the configurable monitor field data.

Sample Code

```
MEISqNodeDriveMonitorInfo monInfo;

returnValue =
   meiSqNodeDriveMonitorInfo(sqNode,
                             driveIndex,
                             &monInfo);

if(returnValue == MPIMessageOK && monInfo.monitorCount) {
   printf("Monitor Information:\n");
   printf("\rAvailable Monitors A");
   if(monInfo.monitorCount > 1) {
      printf(", B");
   }

   if(monInfo.monitorCount > 2) {
      printf(", C");
   }

   printf("\n");
}

if(monInfo.configCount) {
   printf("\nMonitor Configurations:\n");
   for(i = 0; i < monInfo.configCount; i++) {
       if(monInfo.configInfo[i].type == MEISqNodeDriveMonitorDataTypeINDEX)
          printf("\r\r Index %d", monInfo.configInfo[i].value);
       }
       else {
          printf("\r\r Drive address 0x%x", monInfo.configInfo[i].value);
       }
       printf(" %s\n",monInfo.configInfo[i].description);
   }
}
```

See Also

MEISqNodeMonitorConfigInfo | MEISqNodeMonitorLocation | MEISqNodeMonitorValueIndex | meiSqNodeDriveMonitorInfo

MEISqNodeDriveParamCallback

Definition

```
typedef void (*MEISqNodeDriveParamCallback)
             (MEISqNodeDriveParamCallbackType  type,
              const char                       *name,
              long                             number,
              const char                       *value);
```

Change History: Modified in the 03.03.00

Description

In the **MEISqNodeDriveParamCallback** structure, the function's pointer type defines a function that can be passed to the meiSqNodeDriveMapParamFileSet(...) function. The meiSqNodeDriveMapParamFileSet(...) function will call this type of function to report progress or warnings. A NULL value for the callback pointer will disable the callback feature.

type	the type of event that caused the callback function to be called.
name	name of the drive parameter.
number	drive parameter index.
value	the value of the drive parameter.

See Also

meiSqNodeDriveMapParamFileSet

MEISqNodeDriveParamCallbackType

Definition

```
typedef enum {
    MEISqNodeDriveParamCallbackTypeCHANGED,
    MEISqNodeDriveParamCallbackTypeSET_FAILED,
    MEISqNodeDriveParamCallbackTypeSKIPPING_READ_ONLY,
} MEISqNodeDriveParamCallbackType;
```

Change History: Modified in the 03.03.00

Description

The **MEISqNodeDriveParamCallbackType** enumeration is used by the MEISqNodeDriveParamCallback function to describe the type of event that caused the callback function to be called.

MEISqNodeDriveParamCallbackTypeCHANGED	This indicates that the new drive parameter value is different to the current parameter value.
MEISqNodeDriveParamCallbackTypeSET_FAILED	The drive parameter that was set has failed.
MEISqNodeDriveParamCallbackTypeSKIPPING_READ_ONLY	This indicates that the drive paramerter could not be set because it has read only attributes.

See Also

MEISqNodeDriveParamCallback

MEISqNodeFeedbackSecondary

Definition

```
typedef struct MEISqNodeFeedbackSecondary {
    long    motorIndex;
} MEISqNodeFeedbackSecondary;
```

Description

MEISqNodeFeedbackSecondary allows for configuration of the secondary feedback resources on a SynqNet node.

motorIndex	Indicates motorIndex on the node to which the secondary feedback resource is mapped. This value is MEISqNodeNOT_AVAILABLE if the secondary feedback resource does not exist on the node hardware

See Also

MEISqNodeConfig

MEISqNodeFilename

Definition

```
typedef struct MEISqNodeFileName{
    char  fileName[MEISqNodeFILENAME_MAX];
}MEISqNodeFileName;
```

Description

MEISqNodeFilename is used in methods that retrieve filenames from the MPI.

fileName	String containing the name of an SqNode image file.

See Also

MEISqNodeFpgaType

Definition

```
typedef enum MEISqNodeFpgaType {
    MEISqNodeFpgaTypeBOOT,
    MEISqNodeFpgaTypeRUN_TIME,
} MEISqNodeFpgaType
```

Description

MEISqNodeFpgaType is an enumeration of FPGA types.

MEISqNodeFpgaTypeBOOT	The FPGA is operating with a boot image. The boot image only supports basic SynqNet communication. Use meiSqNodeDownload(...) to download the runtime image to the SynqNet node.
MEISqNodeFpgaTypeRUN_TIME	The FPGA is operating with a runtime image.

See Also

meiSqNodeInfo | MEISqNodeInfoFpga | meiSqNodeDownload

MEISqNodeInfo

Definition

```
typedef struct MEISqNodeInfo {
    long                    motorCount;
    long                    driveCount;
    long                    motorOffset;
    long                    feedbackSecondaryCount;
    MEISqNodeInfoId         id;
    MEISqNodeInfoFpga       fpga;
    MEISqNodeInfoNetwork    network;
    MEISqNodeInfoIo         io;
} MEISqNodeInfo;
```

Description

MEISqNodeInfo contains static data stored for the SynqNet node. The motor objects are indexed sequentially across all the SynqNet nodes associated with each network. Each motor on a controller has a unique number.

motorCount	The number of motors that the SynqNet node supports.
driveCount	The number of drives interfaces that the SynqNet node supports.
motorOffset	The starting number for the first motor on the SynqNet node.
feedbackSecondaryCount	The number of auxillary feedbacks on the node.
id	A structure that contains identification data for the SynqNet node. See MEISqNodeInfoId.
fpga	A structure that contains identification data for the SynqNet node FPGA. See MEISqNodeInfoFpga.
network	A structure that contains network interface information for the SynqNet node. See MEISqNodeInfoNetwork.
io	A structure that returns how many of each type of node I/O this node supports.

See Also

meiSqNodeInfo

MEISqNodeInfoId

Definition

```
typedef struct MEISqNodeInfoId {
    unsigned  long   nodeType;   /* product/mfg code   */
    char             *nodeName;  /* product/mfg string */
    unsigned  long   option;     /* product option code*/
    unsigned  long   switchId;   /* rotary switch id   */
    unsigned  long   unique;     /* unique id code     */

    MPI_BOOL         exactMatch; /* TRUE/FALSE         */

    char             serialNumber[MEISqNodeID_CHAR_MAX];
    char             modelNumber[MEISqNodeID_CHAR_MAX];
    char             manufacturerData[MEISqNodeManufacturerDATA_CHAR_MAX];
} MEISqNodeInfoId;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeInfoId contains identification data for the SynqNet node.

All nodes by all manufacturers will have ***nodeType*** and ***unique*** numbers that should generate a unique identification for each node on the SynqNet network.. Although some node manufacturers may opt to leave the ***serialNumber*** and ***modelNumber*** fields blank, you can still identify and distinguish a node by comparing the ***nodeType*** and ***unique*** numbers. The ***nodeType*** number is also represented by a unique text string *nodeName*.

nodeType	A 32 bit value that identifies the node hardware. The upper 16 bits represent the manufacturer of the SynqNet node hardware. Each manufacturer has a unique value. The lower 16 bits represent the SynqNet node product type. The SynqNet node manufacturer determines a unique value to track a product series. Typically, the node type value is displayed in hex.
***nodeName**	A string that represents the SynqNet nodeType. The nodeName string matches the name of the SqNodeLib node specific header file.
option	The product option code within a product series.
switchId	If a node/drive have an physical address switch on its faceplate, switchId will contain the value to which the switch is set. If an ID switch is not supported by a node, this value will be set to -1 (0xFFFFFFFF).

unique	A 32 bit value that identifies the node. It is an unsigned long. The SynqNet node manufacturer determines this unique value to track a single product. This is useful to determine when individual nodes of the same type are switched or replaced on a SynqNet network. **NOTE**: It is possible for a manufacturer to use the same unique identification number for two nodes of different models. The combination of SqNode.Name (or nodeType) and SqNode.UniqueId will be unique for any given code.
exactMatch	A string that tells you if the node is running under a matched or unmatched classification. The value of meiSqNodeInfo.id.exactMatch is TRUE when all ID components have been matched to a supported configuration. The value is FALSE when running with a default (unmatched) configuration.
serialNumber	A string that represents the SynqNet node serial number. For a given node type, the serial number is unique. The SynqNet node manufacturer determines the serial number to track an individual unit.
modelNumber	A string that represents the SynqNet node model number. The SynqNet node manufacturer determines the model number.
manfacturerData	A string containing Manufacturer-specific data which is stored on the node at time of production.

See Also

meiSqNodeInfo | MEISqNodeInfoFpga

MEISqNodeInfoIo

Definition

```
typedef struct MEISqNodeInfoIo {
   long  digitalInCount;
   long  digitalOutCount;
   long  analogInCount;
   long  analogOutCount;
   long  segmentCount;
   long  maxWait;
} MEISqNodeInfoIo;
```

Change History: Modified in the 03.02.00

Description

MEISqNodeInfoIo lists the number of digital and analog inputs that are supported by a SynqNet node.

digitalInCount	The number of digital inputs on a SynqNet node.
digitalOutCount	The number of digital outputs on a SynqNet node.
analogInCount	The number of analog inputs on a SynqNet node.
analogOutCount	The number of analog outputs on a SynqNet node.
segmentCount	The total number of segments on a SynqNet node.
maxWait	This is the maximum amount of time between when the output bit is set in software and the hardware state takes effect. See Output Waits.

See Also

meiSqNodeInfo | meiSqNodeSegmentDigitalOutGet | meiSqNodeSegmentDigitalOutSet | meiSqNodeSegmentAnalogOutGet | meiSqNodeSegmentAnalogOutSet | What I/O does a nodes support?

MEISqNodeInfoFpga

Definition

```
typedef struct MEISqNodeInfoFpga {
    MEISqNodeFpgaType          type;
    unsigned           long    vendorDevice;
    unsigned           long    version;
    unsigned           long    branchVersion;
    MPI_BOOL                   defaultVersion; /* TRUE/FALSE */
} MEISqNodeInfoFpga;
```

Change History: Modified in the 03.03.00

Description

MEISqNodeInfoFpga contains identification data for the SynqNet node FPGA.

type	The FPGA type. See MEISqNodeFpgaType.
vendorDevice	A 32 bit value that identifies the FPGA image. The upper 16 bits represent the manufacturer of the SynqNet node network interface device. Each manufacturer has a unique vendor value. The lower 16 bits represent the SynqNet node network interface component. The device is typically an FPGA (could be an ASIC). If the device is an FPGA, the vendorDevice information is stored in the FPGA binary image. Each device for a particular vendor has a unique device value. Typically, the vendorDevice value is displayed in hexadecimal format.
version	A 32-bit value that represents the revision of the device. The upper 16 bits (SqMac Version), represent the SynqNet network interface revision. The lower 16 bits (Node Version), represent the device revision. Typically, the version value is displayed in hexadecimal format. Ex: 0x02400344 SqMac Version: 0240 Node Version: 0344

branchVersion	A 32-bit value that identifies the branch from an existing version (MajorMinor) or from another Branch. The upper 16 bits (SqMac Branch Version), represent the SynqNet network interface branch revision. The lower 16 bits (Node Branch Version), represent the device branch revision. Ex: 0x01010102 SqMac Branch Version: 0101 Node Branch Version: 0102 **NOTE**: The FPGA branch version was added in FPGAs with SqMac version 0x0230 (and greater). If the SqMac version is 0x0230 (or greater), then the branch version will properly show the revision information. If the SqMac version is less than 0x0230, then the branch version will show 0xFF.
defaultVersion	Indicates if the default version of the SqNode FPGA image is loaded on this node. The defaultVersion defines the version of the SynqNet node FPGA image that was built and tested with the current version of the MPI.

See Also

meiSqNodeInfo | MEISqNodeInfoId | MPI/SynqNet FPGA Compatibility Check

MEISqNodeInfoNetwork

Definition

```
typedef struct MEISqNodeInfoNetwork {
    long    number;
    long    inPorts;
    long    outPorts;
} MEISqNodeInfoNetwork;
```

Description

MEISqNodeInfoNetwork structure contains information about the SynqNet node's network interface.

Remarks

The labeling convention for IN and OUT ports is for convenience. The hardware ports are identical. During SynqNet initialization, the node are discovered based on the OUT to IN port connections.

number	An index to a SynqNet network associated with a controller.
inPorts	The number of SynqNet IN port network interfaces.
outPorts	The number of SynqNet OUT port network interfaces.

See Also

meiSqNodeInfo

MEISqNodeMemory

Definition

```
typedef enum MEISqNodeMemory {
    MEISqNodeMemoryDATA,        /* node/drive processor RAM */
    MEISqNodeMemoryPROGRAM,     /* drive processor program memory */
    MEISqNodeMemoryIO,          /* drive I/O memory */
    MEISqNodeMemoryDRIVE,       /* direct command to drive */
} MEISqNodeMemory;
```

Description

MEISqNodeMemory is an enumeration of drive region types to access with a service command.

MEISqNodeMemoryDATA	node/drive processor data memory
MEISqNodeMemoryPROGRAM	drive processor program memory
MEISqNodeMemoryIO	drive I/O memory
MEISqNodeMemoryDRIVE	direct command to drive processor

See Also

MEISqNodeCmdHeader | meiSqNodeCommand

MEISqNodeMessage

Definition

```
typedef enum {
    MEISqNodeMessageINVALID,
    MEISqNodeMessageNODE_INVALID,
    MEISqNodeMessageSTATE_ERROR,
    MEISqNodeMessageCONFIG_NETWORK_MISMATCH,
    MEISqNodeMessageMAP_CONFIG_MISMATCH,
    MEISqNodeMessageNOT_IN_CONFIG_FILE,
    MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID,

    MEISqNodeMessageRESPONSE_TIMEOUT,
    MEISqNodeMessageREADY_TIMEOUT,
    MEISqNodeMessageSRVC_ERROR,
    MEISqNodeMessageSRVC_UNSUPPORTED,
    MEISqNodeMessageSRVC_CHANNEL_INVALID,

    MEISqNodeMessageCMD_NOT_SUPPORTED,
    MEISqNodeMessageDISCOVERY_FAILURE,
    MEISqNodeMessageDISPATCH_ERROR,
    MEISqNodeMessageINIT_FAILURE,
    MEISqNodeMessageINTERFACE_ERROR1,
    MEISqNodeMessageFILE_NODE_MISMATCH,
    MEISqNodeMessageFILE_INVALID,
    MEISqNodeMessageINVALID_HEADER,
    MEISqNodeMessageDOWNLOAD_FAIL,
    MEISqNodeMessageVERIFY_FAIL,
    MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED,
    MEISqNodeMessageVERIFY_NOT_SUPPORTED,
    MEISqNodeMessageBOOT_ROM_INVALID,
    MEISqNodeMessageINVALID_TABLE,
    MEISqNodeMessageINVALID_STR_LEN,
    MEISqNodeMessageFEEDBACK_MAP_INVAILD,
    MEISqNodeMessageNODE_FAILURE,
    MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT,

    MEISqNodeMessageIO_MODULE_INCOMPATIBILITY,
    MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED,
    MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED,
    MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED,
    MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED,
    MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED,

    MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT,
```

```
    MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MANY_SLICES,
    MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH,
    MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT,
    MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH,
    MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR,
    MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR,
    MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE,
    MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE,
    MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE,
    MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED,
    MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE,
    MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE,
    MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT,
    MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE,
    MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA,
    MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED,
    MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA,
    MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST,
    MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER,
    MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE,
    MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE,
    MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT,
    MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT,
    MEISqNodeMessageIO_SLICE_EEPROM_FORMAT,
    MEISqNodeMessageIO_SLICE_TOO_MUCH_IO,

    MEISqNodeMessageBOOT_FILE_NOT_FOUND,
    MEISqNodeMessagePARAM_READ_ONLY,
    MEISqNodeMessagePARAM_LOCKED,
    MEISqNodeMessageMONITOR_INDEX,
    MEISqNodeMessageMONITOR_ADDRESS,
    MEISqNodeMessageMONITOR_NA,

    MEISqNodeMessageCOMMUTATION_INIT_FAILURE,
    MEISqNodeMessagePOSITION_CLEAR_FAILURE,
    MEISqNodeMessageNODE_EEPROM_SET,
} MEISqNodeMessage;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00. Modified in the 03.02.00

Description

MEISqNodeMessage is an enumeration of SynqNet node error messages that can be returned by the MPI library.

MEISqNodeMessageINVALID

The SqNode type is out of range. This message code is returned by SynqNet node methods if the node type is not a member of the SQNodeLibNodeType enumeration.

MEISqNodeMessageNODE_INVALID

The SynqNet Node number is out of range. This message code is returned if the given node number is less than zero, or greater than or equal to MEISynqNetMaxNODE_COUNT.

MEISqNodeMessageSTATE_ERROR

Some methods can only be executed in SYNQ state. This message will be returned when the network is not in the expected network state (i.e ASYNQ state).

MEISqNodeMessageCONFIG_NETWORK_MISMATCH

The type of map file specified in meiSqNodeDriveMapParamFileSet(...) does not match the type of drive found on the network.

MEISqNodeMessageMAP_CONFIG_MISMATCH

The parameter name or number specified in meiSqNodeDriveMapParamFileSet(...) was not valid for the specified drive.

MEISqNodeMessageNOT_IN_CONFIG_FILE

The parameter name or number specified in meiSqNodeDriveMapParamFileSet(...) was not found.

MEISqNodeMessageCONFIG_FILE_FORMAT_INVALID

A file with an incorrect format was used in meiSqNodeDriveMapParamFileSet(...).

MEISqNodeMessageRESPONSE_TIMEOUT

The drive/node did not respond to the service command issued in a reasonable amount of time.

MEISqNodeMessageREADY_TIMEOUT

The node/drive did not complete the hand shaking for the service command.

MEISqNodeMessageSRVC_ERROR

There was an error in carrying out the service command issued.

MEISqNodeMessageSRVC_UNSUPPORTED

The service command issued is either not supported or recognized by the drive.

MEISqNodeMessageSRVC_CHANNEL_INVALID

Invalid service channel specified. See MEISqNodeCmdHeader.

MEISqNodeMessageCMD_NOT_SUPPORTED

The service command is not supported by the node.

MEISqNodeMessageDISCOVERY_FAILURE

Unable to discover node rescources.

MEISqNodeMessageDISPATCH_ERROR

Is the default error code returned when a node specific routine has failed. Check the node FPGA version to verify whether or not it is correct.

MEISqNodeMessageINIT_FAILURE

A node specific initialization routine was unable to successfully complete its routine. Verify that the node FPGA is the default version for your MPI version. See MEISqNodeInfoFpga.

MEISqNodeMessageINTERFACE_ERROR1

This is an outdated node, which does not support the current discovery routine.

MEISqNodeMessageFILE_NODE_MISMATCH

Node type does not match the file provided for download.

MEISqNodeMessageFILE_INVALID

The file provided for download was not found or was corrupted.

MEISqNodeMessageINVALID_HEADER

The header information in the download image is invalid. Please verify firmware file to be correct and retry download. If firmware file is correct please contact firmware manufacturer.

MEISqNodeMessageDOWNLOAD_FAIL

Node firmware download failed. Verify that the firmware file is correct and retry the download.

NOTE: A network reset my be required.

MEISqNodeMessageVERIFY_FAIL

The node FPGA firmware does not match the FPGA image file.

MEISqNodeMessageDOWNLOAD_NOT_SUPPORTED

The downloading of the node firmware (FPGA) image is not supported for this node.

MEISqNodeMessageVERIFY_NOT_SUPPORTED

The Node specified for verification does not support the upload of the FPGA image. Therefore, the image cannot be verified.

MEISqNodeMessageBOOT_ROM_INVALID

The SqNode Boot Rom identification or version is not recognized by the MPI.

MEISqNodeMessageINVALID_TABLE

Invalid resource table in node module. This is a fatal error within the MPI. Please verify MPI and node FPGA versions to be correct and then contact MEI's Technical Support.

MEISqNodeMessageINVALID_STR_LEN

An attempt to write information to the node has failed due to an invalid string length.

MEISqNodeMessageFEEDBACK_MAP_INVAILD

Returned from meiSqNodeConfigSet(...) when the given secondary encoder (*n*) is not mappable to the motor on the node specified by MEISqNodeFeedbackSecondary[n].motorIndex.

MEISqNodeMessageNODE_FAILURE

An attempt was made to access a SynqNet node that has a node failure event active. SynqNet Node Failure describes the details.

MEISqNodeMessageEXCEEDED_MAXIMUM_SYNQNET_PACKET_LIMIT

When initializing the SynqNet network one of the node requires a SynqNet packet larger that what can be supported.

MEISqNodeMessageIO_MODULE_INCOMPATIBILITY

Two modules attached to a SQID node are incompatible. This error message code is returned when initializing a SQID node. Different types of I/O module may be incompatible and will not work on the same SQID node.

MEISqNodeMessageIO_MODULE_EEPROM_NOT_PROGRAMMED

The EEPROM on one of the modules attached to a SQID node has not been programmed.

MEISqNodeMessageIO_MODULE_COUNT_EXCEEDED

The maximum number of I/O that can be supported by a SQID node has been exceeded.

MEISqNodeMessageIO_MODULE_LENGTH_CHECK_FAILED

When initializing an SQIO node, checks are performed to confirm that the node actually supports the correct number of I/O. This error is returned when one of these tests fails. An error will occur when the length of at least one of the inter module (SPI) buses did not match the length calculated from data held in the module EEPROMs. This error message can be returned by MPI functions that reset the SynqNet Network such as, mpiControlReset(...), mpiControlInit(...), meiSynqNetInit(...). This error can ONLY be generated by SQIO nodes. This fault can also be caused by a poor electrical connection between the SQID and the I/O modules. If the modules are firmly connected and the error persists then you will need to contact your supplier of the I/O Modules to correct the hardware.

MEISqNodeMessageIO_MODULE_3_3V_BUS_CURRENT_EXCEEDED

During the initialization of the modules attached to a SQID node, the maximum current that can be drawn from the inter module 3.3V bus exceeds the allowable current.

MEISqNodeMessageIO_MODULE_24V_BUS_CURRENT_EXCEEDED

During the initialization of the modules attached to a SQID node, the maximum current that can be drawn from the inter module 5V bus exceeds the allowable current.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT

An error was encountered while initalizing the Slice I/O node.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_TOO_MAY_SLICES

Slice I/O nodes can only support 32 slices attached to the network adapter. This error is returned when more than 32 slices are detected.

MEISqNodeMessageIO_SLICE_INITIALIZATION_FAULT_VENDOR_MISMATCH

A slice is attached to the node that is not supplied by MEI. You can only use slices supplied by MEI.

MEISqNodeMessageIO_SLICE_INITIALIZATION_TIMEOUT

When initializing a Slice I/O node the initialization routine failed to complete within the expected time.

MEISqNodeMessageIO_SLICE_TOPOLOGY_MISMATCH

When attempting to restore the slice parameters to the attached slices of a Slice I/O node, the arrangement of slices did not match the expected arrangement. If you wish to clear the previous slice parameters, you can stop this error by calling meiSqNodeSegmentParamClear(...).

MEISqNodeMessageIO_SLICE_SERVICE_RECEIVE_ERROR

When attempting to access data on Slice I/O, a communication fault was detected. The message received from the slice was badly formed, errors include CRC and missing start/stop bits.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MANY_CHAR

When attempting to access data on Slice I/O, a communication fault was detected. The slice returned too many characters when responding to this request.

MEISqNodeMessageIO_SLICE_SERVICE_BUS_ERROR_CODE

When attempting to access data on Slice I/O, a communication fault was detected. The slice returned an error code.

MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_FAULT_CODE

When attempting to access data on Slice I/O, a communication fault was detected. An unknown fault was detected when accessing this slice.

MEISqNodeMessageIO_SLICE_SERVICE_RESOURCE_UNAVAILABLE

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_NOT_SUPPORTED

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the action requested on this data. For example, a read or write.

MEISqNodeMessageIO_SLICE_SERVICE_INVALID_ATTRIBUTE_VALUE

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_ALREADY_IN_MODE

When attempting to access data on Slice I/O, a communication fault was detected. The slice is already in the requested mode.

MEISqNodeMessageIO_SLICE_SERVICE_STATE_CONFLICT

When attempting to access data on Slice I/O, a communication fault was detected. The slice can not perform the requested action in its current state.

MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SETTABLE

When attempting to access data on Slice I/O, a communication fault was detected. You cannot modify the data on this slice.

MEISqNodeMessageIO_SLICE_SERVICE_NOT_ENOUGH_DATA

When attempting to access data on Slice I/O, a communication fault was detected. Not enough data was supplied to the slice for this operation.

MEISqNodeMessageIO_SLICE_SERVICE_ATTRIBUTE_NOT_SUPPORTED

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_DATA

When attempting to access data on Slice I/O, a communication fault was detected. Too much data was supplied to the slice for this operation.

MEISqNodeMessageIO_SLICE_SERVICE_OBJECT_DOES_NOT_EXIST

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the requested data.

MEISqNodeMessageIO_SLICE_SERVICE_INVALID_PARAMETER

When attempting to access data on Slice I/O, a communication fault was detected. The slice does not support the specified parameter.

MEISqNodeMessageIO_SLICE_SERVICE_STORE_OPERATION_FAILURE

When attempting to access data on Slice I/O, a communication fault was detected. The slice failed during a store operation.

MEISqNodeMessageIO_SLICE_SERVICE_UNKNOWN_ERROR_CODE

When attempting to access data on Slice I/O, a communication fault was detected. The error code from the slice was not recognized.

MEISqNodeMessageIO_SLICE_SERVICE_TIMEOUT

When attempting to access data on Slice I/O, a communication fault was detected. The operation on the slice exceeded the timeout threshold.

MEISqNodeMessageIO_SLICE_SERVICE_RESPONSE_FORMAT

When attempting to access data on Slice I/O, a communication fault was detected. The response from the slice was not formatted correctly.

MEISqNodeMessageIO_SLICE_SERVICE_EEPROM_FORMAT

The data held on the EEPROM on the network adaptor was not formatted correctly.

MEISqNodeMessageIO_SLICE_SERVICE_TOO_MUCH_IO

When starting the slice node, too many I/O slices were found. You should remove some slices to stop this error message from being generated.

MEISqNodeMessageBOOT_FILE_NOT_FOUND

The boot file "kollmorgen_ember.a00" was not found. When downloading drive images to Kollmorgen CD, DASA, and PicoDAD drives, a boot file is downloaded to the drive prior to the actual drive image. This boot file needs to be located in the same directory as the drive's image file that is provided for download.

MEISqNodeMessagePARAM_READ_ONLY

The drive parameter that the user is attempting to set is read only.

MEISqNodeMessagePARAM_LOCKED

The drive parameter that the user is attempting to set is not accessible. SelSFDParam must be set to 0, otherwise the SFD motor parameters will be used.

MEISqNodeMessageMONITOR_INDEX

Drive does not support the configuring of Monitors through indexing.

MEISqNodeMessageMONITOR_ADDRESS

Drive does not support the configuring of Monitors through addressing.

MEISqNodeMessageMONITOR_NA

The hardware support for SynqNet node monitors is not available. This message is returned by meiSqNodeDriveMonitorInfo(…), if the node/drive hardware does not support monitor fields. To avoid this problem, do not use the monitor methods if the SynqNet node hardware does not support it. The monitor field data is SynqNet drive-specific. All drives do not support monitors. SynqNet RMBs (Remote Motion Blocks) or I/O nodes do not support monitors.

MEISqNodeMessageCOMMUTATION_INIT_FAILURE

Motor commutation initialization failed. This message is returned from sgdsCommutationInit(…) or sgdzBsCommutationInit(…) if the commutation initialization fails.

MEISqNodeMessagePOSITION_CLEAR_FAILURE

SynqNet drive did not clear the position as requested. This message is returned from sgdsEncoderPositionClear(…) or sgdzBsEncoderPositionClear(…) if the SynqNet drive does not respond to the service command request to clear the feedback position.

MEISqNodeMessageNODE_EEPROM_SET

SynqNet node EERPOM was modified by the MPI. This message is returned from mpiControlInit(…) or meiSynqNetInit(…) if the Kollmorgen S300, S600 or S1800 drive's EEPROM was not properly configured at the factory. The MPI will attempt to discover the drive's configuration and set the appropriate option number in the node's EEPROM.

See Also

meiSqNodeDriveMapParamFileSet | meiSqNodeConfigSet

MEISqNodeMonitorConfigInfo

Definition

```
typedef struct MEISqNodeMonitorConfigInfo{
   MEISqNodeDriveMonitorDataType  type;
   long                           value;
   const char                     *description;
} MEISqNodeMonitorConfigInfo;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeMonitorConfigInfo is a structure that contains information about the configurable monitor field data. This structure is useful for determining the monitor selection type, its selection value, and a brief text description. The monitor field data is SynqNet drive-specific. Not all drives support monitors.

NOTE: SynqNet RMBs (Remote Motion Blocks) or I/O nodes do not support monitors.

type	Monitor field data is selected by an index (MEISqNodeDriveMonitorDataTypeINDEX) or a drive address (MEISqNodeDriveMonitorDataTypeADDRESS).
value	Monitor index or drive address to select the monitor field data.
***description**	A string to describe the monitor field data.

Sample Code

```
MEISqNodeDriveMonitorInfo monInfo;


returnValue =
   meiSqNodeDriveMonitorInfo(sqNode,
                             driveIndex,
                             &monInfo);

printf("\nMonitor Configurations:\n");
if(monInfo.configCount) {
   for(i = 0; i < monInfo.configCount; i++) {
       if(monInfo.configInfo[i].type == MEISqNodeDriveMonitorDataTypeINDEX) {
          printf("\r\r Index %d", monInfo.configInfo[i].value);
       }
       else {
          printf("\r\r Drive address 0x%x", monInfo.configInfo[i].value);
```

```
        }
        printf(" %s\n",monInfo.configInfo[i].description);
    }
}
else {
     printf("Monitors are not Configurable.\n");
     }
```

See Also

MEISqNodeDriveMonitorDataType | MEISqNodeDriveMonitorInfo | meiSqNodeDriveMonitorInfo | MEISqNodeMonitorValueIndex

MEISqNodeMonitorLocation

Definition

```
typedef struct MEISqNodeMonitorLocation {
   long            *addr;
   unsigned  long  mask;
   unsigned  long  rightShift;
} MEISqNodeMonitorLocation;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeMonitorLocation is a structure to decode the monitor field. It contains a controller address, bit mask, and bit shift information for the specific monitor field: MEISqNodeMonitorValueIndexA, MEISqNodeMonitorValueIndexB, or MEISqNodeMonitorValueIndexC.

addr	address for the drive monitor field
mask	bit mask to apply to the monitor field
rightShift	bit shift to apply to monitor field

Sample Code

Read and display the supported monitor field information.

```
MEIMotorConfig MEISqNodeDriveMonitorInfo monInfo;

returnValue =
meiSqNodeDriveMonitorInfo(sqNode,
                          driveIndex,
                          &monInfo);

if(returnValue == MPIMessageOK && monInfo.monitorCount) {
   printf("Monitor Information:\n");
   printf("\rAvailable Monitors A");
   if(monInfo.monitorCount > 1) {
   printf(", B");
   }
   if(monInfo.monitorCount > 2) {
```

```
   printf(", C");
   }
   printf("\n\nMonitor Locations:\n");
   for(i = 0; i < monInfo.monitorCount; i++) {
   printf(" %c Controller address 0x%x, mask 0x%x, right shift %d\n",
   'A'+ i, monInfo.location[i].addr, monInfo.location[i].mask,
           monInfo.location[i].rightShift);
  }
}
```

See Also

MEISqNodeMonitorConfigInfo | MEISqNodeDriveMonitorInfo | meiSqNodeDriveMonitorInfo | MEISqNodeMonitorValueIndex

MEISqNodeMonitorValue

Definition

```
typedef struct MEISqNodeMonitorValue {
    long    count;
    long    monitor[MEISqNodeMonitorValueIndexLAST];
} MEISqNodeMonitorValue;
```

Description

MEISqNodeMonitorValue contains the data for the monitor fields read by the meiSqNodeDriveMonitor (...) method.

count	The number of monitor fields read. This specifies the size of the monitor array.
monitor	An array of monitor data fields. Each field is indexed by the MEISqNodeMonitorValueIndex enumeration.

See Also

meiSqNodeDriveMonitor | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeMonitorValueIndex

Definition

```
typedef enum MEISqNodeMonitorValueIndex {
    MEISqNodeMonitorValueIndexA,
    MEISqNodeMonitorValueIndexB,
    MEISqNodeMonitorValueIndexC,
} MEISqNodeMonitorValueIndex;
```

Change History: Modified in the 03.04.00.

Description

MEISqNodeMonitorValueIndex is an enumeration of indices to node monitor values.

MEISqNodeMonitorValueIndexA	Index to node monitor value A.
MEISqNodeMonitorValueIndexB	Index to node monitor value B.
MEISqNodeMonitorValueIndexC	Index to node monitor value C.

See Also

meiSqNodeDriveMonitor | meiSqNodeDriveMonitorConfigGet | meiSqNodeDriveMonitorConfigSet

MEISqNodeResponse

Definition

```
typedef struct MEISqNodeResponse {
    unsigned long    data;    /* response data */
} MEISqNodeResponse;
```

Description

MEISqNodeResponse contains the service command response data.

data	The response information from a service command. The data field is only valid for MEISqNodeCmdTypeREAD command types.

See Also

meiSqNodeCommand

MEISqNodeSegmentInfo

Definition

```
#define MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH 0x20
#define MEISqNodeSegmentInfoMODEL_NAME_LENGTH    0x20
#define MEISqNodeSegmentInfoMANUFACTURER_LENGTH  0x10

typedef struct MEISqNodeSegmentInfo {
   long  id;
   long  option;
   char  serialNumber[MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH];
   char  modelName[MEISqNodeSegmentInfoMODEL_NAME_LENGTH];
   long  digitalInCount;
   long  digitalOutCount;
   long  analogInCount;
   long  analogOutCount;
   long  version;
   long  paramCount;
   long  memoryCount;
   char  manufacturerData[MEISqNodeManufacturerDATA_CHAR_MAX];
} MEISqNodeSegmentInfo;
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfo contains data about the I/O that is supported by a segment (slice or module) attached to a SynqNet node.

id	This field contains a 32-bit number that uniquely identifies this kind of segment. For modules attached to a SQID node, the top 16 bits are the manufacturer code and the bottom 16 bits are to product code.
option	The option code for the segment. For slices attached to a Slice network adaptor this field is always zero.
serialNumber	The serial number of this segment. **NOTE**: Slice-I/O nodes do NOT report serial numbers.
modelName	A text string giving the model name of this module.

digitalInCount	The total number of digital inputs on this segment.
digitalOutCount	The total number of digital outputs on this segment.
analogInCount	The total number of analog inputs on this segment.
analogOutCount	The total number of analog outputs on this segment.
version	The version of the segment. For modules attached to a SQID node, this field is always zero.
paramCount	The total number segment parameters supported by this segment. For modules attached to a SQID node this field is always zero.
memoryCount	The total number of memory bytes available on this segment. For modules attached to a SQID node, this field is always zero.
manufacturerData	A series of characters programmed into the node during manufacturing. For slices attached to a Slice network adaptor, this field is always zero.

See Also

MEISqNodeConfigIoAbort

MEISqNodeSegmentUserData

Definition

```
typedef struct MEISqNodeSegmentUserData {
        char   data[MEISqNodeSegmentUserDATA_CHAR_MAX];
} MEISqNodeSegmentUserData;
```

Change History: Modified in the 03.02.00

Description

Modules attached to a SQID node have a small section of non-volatile memory that can be used for any purpose by the user. The **MEISqNodeSegmentUserData** structure holds a copy of this data.

data	Up to 16 bytes of data.

See Also

meiSqNodeSegmentUserDataGet | meiSqNodeSegmentUserDataSet | MPI Overview I/O: User Data

MEISqNodeStatus

Definition

```
typedef struct MEISqNodeStatus {
    MEISqNodeStatusPacketError   upStreamError;
    MEISqNodeStatusPacketError   downStreamError;
    MEISqNodeStatusCrcError      crcError;
    MPIEventMask                 eventMask;
    MPIEventStatusIoFaults       ioFaults;
} MEISqNodeStatus;
```

Change History: Modified in the 03.04.00.

Description

MEISqNodeStatus contains error counters and the ***eventMask*** for a SynqNet node.

upStreamError	The rate and count of bad synqNet messages received by the controller from the Node. See MEISqNodeStatusPacketError.
downStreamError	The rate and count of bad synqNet messages received by the Node from the controller. See MEISqNodeStatusPacketError.
crcError	Counters for the CRC errors. See MEISqNodeStatusCrcError.
eventMask	Array that defines the event mask bits. The array is defined as: `typedef MPIEventMaskELEMENT_TYPE MPIEEventMask [MPIEventMaskELEMENTS]` The bits are defined by the MPI/MEIEventType enumerations.
ioFaults	Flags indicating the source of any I/O faults. See MEISqNodeStatusIoFaults.

See Also

meiSqNodeStatus | meiSynqNetStatus | MEISqNodeConfig | MEISqNodeStatusIoFaults

MEISqNodeStatusCrcError

Definition

```
typedef struct MEISqNodeStatusCrcError {
    long    port[MEINetworkPortLAST];
} MEISqNodeStatusCrcError;
```

Description

MEISqNodeStatusCrcError contains CRC error counters for each network port. The CRC error counters are helpful for diagnosing data integrity problems. The counter increments for any CRC error on any packet received at that port (whether the packet is addressed to the node or not). The CRC error counters are cleared during network initialization.

port	An array of CRC error counters. Each network port has one CRC error counter. The valid range is 0 to 255. The value saturates at 255. A value of -1 indicates the controller was unable to read the data from the node due to a failed connection.

See Also

meiSqNodeStatus | MEINetworkPort

MEISqNodeStatusIoFaults

Definition

```
typedef struct MEISqNodeStatusIoFaults {
   MPI_BOOL  digitalIn;
   MPI_BOOL  digitalOut;
   MPI_BOOL  analogIn;
   MPI_BOOL  analogOut;
} MEISqNodeStatusIoFaults;
```

Change History: Added in the 03.04.00.

Description

MEISqNodeStatusIoFaults contains flags that indicate the source of any I/O fault.

digitalIn	This flag indicates that there is a fault with using the digital inputs.
digitalOut	This flag indicates that there is a fault with using the digital outputs.
analogIn	This flag indicates that there is a fault with using the analog inputs.
analogOut	This flag indicates that there is a fault with using the analog outputs.

See Also

MEISqNodeStatus | meiSqNodeStatus

I/O Faults

MEISqNodeStatusPacketError

Definition

```
typedef struct MEISqNodeStatusPacketError {
    long    rate;
    long    count;
} MEISqNodeStatusPacketError;
```

Description

MEISqNodeStatusPacketError contains packet error counters and rate counters. Each SynqNet node has a packet error counter and a packet error rate counter. Packets addressed to a node are checked for integrity.

The packet error counters are used to monitor long-term data integrity. These counters do not trigger any fault or fail actions. The packet error counter is incremented once for each missing or invalid packet. Typically, an application will periodically read the packet error counters and store the values in a log.

The packet error rate counters are used to trigger fault recovery and/or failure shutdown. The packet error rate counter is incremented for each missing or invalid packet and is decremented for 16 consecutive valid packets. Thus, the packet error rate counters can detect large errors over short periods of time or small errors over long periods of time.

rate	The packet error rate counter. The valid range is 0 to 255. The value saturates at 255. A value of -1 indicates the controller was unable to read the data from the node due to a failed connection.
count	The packet error counter. The valid range is 0 to 255. The value saturates at 255. A value of -1 indicates the controller was unable to read the data from the node due to a failed connection.

See Also

meiSqNodeStatus | MEISqNodeConfigPacketError

MEISqNodeUserData

Definition

```
typedef struct MEISqNodeUserData {
    char    data[MEISqNodeUserDATA_CHAR_MAX];
}MEISqNodeUserData ;
```

Change History: Modified in the 03.02.00

Description

MEISqNodeUserData is used to store the user information that is located on the SqNode.

data	User information on the SqNode used for storing SqNode identification or any other useful data. Programmable string to be used by a customer to store identification-specific information. This data is not used by the MPI and is stored in the SqNode's EEPROM.

See Also

meiSqNodeUserDataGet | meiSqNodeUserDataSet

MEISqNodeConfigControlLatencyMIN_LIMIT

Definition

```
#define  MEISqNodeConfigControlLatencyMIN_LIMIT  (1000)
```

Change History: Added in the 03.04.00.

Description

MEISqNodeConfigControlLatencyMIN_LIMIT defines the minimum specifiable limit for the node's control latency.

See Also

MEISqNodeConfigControlLatencyMAX_LIMIT | MEISqNodeConfigControlLatency | meiSqNodeConfigSet | meiSqNodeConfigGet

MEISqNodeConfigControlLatencyMAX_LIMIT

Definition

```
#define  MEISqNodeConfigControlLatencyMAX_LIMIT  (1000)
```

Change History: Added in the 03.04.00.

Description

MEISqNodeConfigControlLatencyMAX_LIMIT defines

See Also

MEISqNodeConfigControlLatencyMIN_LIMIT

MEISqNodeDriveParamMAX_STRING_LENGTH

Definition

```
#define MEISqNodeDriveParamMAX_STRING_LENGTH  MEIDriveParamMAX_STRING_LENGTH
```

Change History: Modified in the 03.03.00

Description

MEISqNodeDriveParamMAX_STRING_LENGTH defines the maximum

See Also

MEISqNodeID_CHAR_MAX

Definition

```
#define   MEISqNodeID_CHAR_MAX  (30)
```

Description

MEISqNodeID_CHAR_MAX defines the maximum length (number of characters) in a sqNode identification string.

See Also

MEISqNodeFILENAME_MAX

Definition

```
#define  MEISqNodeeFILENAME_MAX  (18)
```

Description

MEISqNodeFILENAME_MAX defines the maximum size allowed for SqNode filenames.

See Also

MEISqNodeManufacturerDATA_CHAR_MAX

Definition

```
#define  MEISqNodeManufacturerDATA_CHAR_MAX  (0x10)
```

Description

MEISqNodeManufacturerDATA_CHAR_MAX defines the maximum number of characters stored in the Manufacturer's Data field.

See Also

MEISqNodeMaxFEEDBACK_SECONDARY

Definition

```
#define  MEISqNodeMaxFEEDBACK_SECONDARY  (MEISqNodeMaxMOTORS)
```

Description

MEISqNodeMaxFEEDBACK_SECONDARY defines the maximum number of secondary feedback devices per SynqNet node.

See Also

MEISqNodeConfig | MEISqNodeFeedbackSecondary

MEISqNodeMaxMOTORS

Definition

```
#define  MEISqNodeMaxMOTORS  (MEIXmpMotorsPerBlock)
```

Description

MEISqNodeMaxMOTORS defines the maximum number of motor objects supported on a single SynqNet node. (MEIXmpMotorsPerBlock = 8) This define should be used instead of the MEIXmpMotorsPerBlock definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

SqNode Objects

MEISqNodeNOT_AVAILABLE

Definition

```
#define  MEISqNodeNOT_AVAILABLE  (-1)
```

Description

MEISqNodeNOT_AVAILABLE defines a possible value for MEISqNodeConfig.feedbackSecondary[n].motorIndex.
If motorIndex = MEISqNodeNOT_AVAILABLE, then a secondary feedback device does not exist on the hardware.

See Also

MEISqNodeConfig | MEISqNodeFeedbackSecondary

MEISqNodeSEGMENT_MAX

Definition

```
#define  MEISqNodeSEGMENT_MAX (32)
```

Change History: Added in the 03.03.00

Description

MEISqNodeSEGMENT_MAX defines the maximum number of Slices or SQID Modules that can be supported.

See Also

Slice I/O | SQID

MEISqNodeSEGMENT_PARAMS_MAX

Definition

```
#define  MEISqNodeSEGMENT_PARAMS_MAX (10)
```

Change History: Added in the 03.03.00

Description

MEISqNodeSEGMENT_PARAMS_MAX defines the maximum number of parameters a Slice can support.

Sample Code

The following code shows how to read all the parameters from a slice, where you may not know how may parameters a slice supports.

```
MEISqNodeSegmentInfo segmentInfo;
meiSqNodeInfo( sqNode, 0, &segmentInfo );

char parameters[MEISqNodeSEGMENT_PARAMS_MAX];
meiSqNodeSegmentParamGet( sqNode0,
                          0,
                          0,
                          segmentInfo.paramCount,
                          parameters );
```

See Also

meiSqNodeSegmentParamSet | meiSqNodeSegmentParamGet

MEISqNodeSEGMENT_MEMORY_MAX

Definition

```
#define  MEISqNodeSEGMENT_MEMORY_MAX (30)
```

Change History: Added in the 03.03.00

Description

MEISqNodeSEGMENT_MEMORY_MAX is a macro that defines the maximum number of memory registers a Slice can support.

Sample Code

The following code shows how to read all the memory registers from a slice, where you may not know how may memory registers a slice supports.

```
MEISqNodeSegmentInfo segmentInfo;
meiSqNodeInfo( sqNode, 0, &segmentInfo );

char memory[MEISqNodeSEGMENT_MEMORY_MAX];
meiSqNodeSegmentMemoryGet( sqNode0,
                           0,
                           0,
                           segmentInfo.memoryCount,
                           memory );
```

See Also

meiSqNodeSegmentMemorySet | meiSqNodeSegmentMemoryGet

MEISqNodeSTATUS_NOT_AVAILABLE

Definition

```
#define  MEISqNodeSTATUS_NOT_AVAILABLE  (-1)
```

Description

With exception to MEISqNodeStatus.eventMaskValue, the **MEISqNodeSTATUS_NOT_AVAILABLE** value is assigned to all Node status variables when the Status is not available as a result of lost communication with the node.

See Also

MEISqNodeStatus

MEISqNodeUserDATA_CHAR_MAX

Definition

```
#define  MEISqNodeUserDATA_CHAR_MAX  (0x10)
```

Description

MEISqNodeUserDATA_CHAR_MAX defines the maximum number of characters in the User defined string, stored on the SqNode.

See Also

MEIFpgaSqNodeVersionMAX | MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEISqNodeSegmentInfoMANUFACTURER_LENGTH

Definition

```
#define MEISqNodeSegmentInfoMANUFACTURER_LENGTH          0x10
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfoMANUFACTURER_LENGTH defines the maximum number of bytes in the manufacturer data.

See Also

Overview of MPI I/O: What Information is Available About Each I/O Segment | meiSqNodeSegmentInfo

MEISqNodeSegmentInfoMODEL_NAME_LENGTH

Definition

```
#define  MEISqNodeSegmentInfoMODEL_NAME_LENGTH   0x20
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfoMODEL_NAME_LENGTH defines the maximum number of characters in the model name description.

See Also

Overview of MPI I/O: What Information is Available About Each I/O Segment | meiSqNodeSegmentInfo

MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH

Definition

```
#define  MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH   0x20
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentInfoSERIAL_NUMBER_LENGTH defines the maximum number of characters in the serial number.

See Also

Overview of MPI I/O: What Information is Available About Each I/O Segment | meiSqNodeSegmentInfo

MEISqNodeSegmentUserDATA_CHAR_MAX

Definition

```
#define MEISqNodeSegmentUserDATA_CHAR_MAX   (0x10)
```

Change History: Added in the 03.02.00

Description

MEISqNodeSegmentUserDATA_CHAR_MAX defines the maximum number of user data bytes.

See Also

MEISqNodeSegmentUserData | MPI Overview I/O: User Data

MEIDriveMapParamMAX_STRING_LENGTH

Declaration

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

Required Header: stdmei.h

Description

MEIDriveMapParamMAX_STRING_LENGTH macro defines the maximum length of a string that can be read from, or written to a drive parameter. The value is defined by a drive map constant.

See Also

MEIFPGARINCONREV

Definition

```
#define  MEIFPGARINCONREV  (0x0242)
```

Change History: Modified in the 03.04.00. Modified in the 03.02.00.

Description

MEIFPGARINCONREV defines the version of the SynqNet controller FPGA image that was built and tested with the current version of the MPI.

See Also

MEIFpgaSqMACVersionDEFAULT

Definition

```
#define MEIFpgaSqMACVersionDEFAULT    (0x0300)
```

Change History: Modified in the 03.04.00

Description

MEIFpgaSqMACVersionDEFAULT defines the version of the SqMAC FPGA image that was built and tested with the current version of the MPI. The sqMAC FPGA image is built into the SynqNet node FPGA image. This version applies to all node types.

See Also

MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqMACVersionMIN

Definition

```
#define MEIFpgaSqMACVersionMIN    (0x0300)
```

Change History: Modified in the 03.04.00

Description

MEIFpgaSqMACVersionMIN defines the minimum version of the SqMAC FPGA image that is compatible with the current version of the MPI. The sqMAC FPGA image is built into the SynqNet node FPGA image.

This version applies to all node types.

See Also

MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqMACVersionMAX

Definition

```
#define  MEIFpgaSqMACVersionMAX    (0x03FF)
```

Change History: Modified in the 03.04.00

Description

MEIFpgaSqMACVersionMAX defines the maximum version of the SqMAC FPGA image that is compatible with the current version of the MPI. The sqMAC FPGA image is built into the SynqNet node FPGA image.

This version applies to all node types.

See Also

MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqNodeVersionDEFAULT

Definition

```
#define MEIFpgaSqNodeVersionDEFAULT    (0x0400)
```

Change History: Modified in the 03.04.00. Modified in the 03.03.00. Modified in the 03.02.00.

Description

MEIFpgaSqNodeVersionDEFAULT defines the version of the SynqNet node FPGA image that was built and tested with the current version of the MPI. This is the recommended version to have loaded on all SynqNet nodes.

This version may not apply to all node types.

See Also

MEIFpgaSqNodeVersionMIN | MEIFpgaSqNodeVersionMAX | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqNodeVersionMIN

Definition

```
#define MEIFpgaSqNodeVersionMIN    (0x0400)
```

Change History: Modified in the 03.04.00.

Description

MEIFpgaSqNodeVersionMIN defines the minimum version of the SynqNet node FPGA image that is compatible with the current version of the MPI.

This version may not apply to all node types.

See Also

MEIFpgaSqNodeVersionMAX | MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check

MEIFpgaSqNodeVersionMAX

Definition

```
#define MEIFpgaSqNodeVersionMAX    (0x04FF)
```

Change History: Modified in the 03.04.00.

Description

MEIFpgaSqNodeVersionMAX defines the maximum version of the SynqNet node FPGA image that is compatible with the current version of the MPI.

This version may not apply to all node types.

See Also

MEIFpgaSqNodeVersionMIN | MEIFpgaSqNodeVersionDEFAULT | MPI/SynqNet FPGA Compatibility Check