Node Objects

Introduction

A **Node** object manages an individual SERCOS node. A SERCOS node may be a drive or I/O module.

Methods

Create, Delete, Validate Methods

mpiNode**Create**	Create Node object
mpiNode**Delete**	Delete Node object
mpiNode**Validate**	Validate Node object

Configuration and Information Methods

mpiNode**ConfigGet**	Get Node config
mpiNode**ConfigSet**	Set Node config
mpiNode**FlashConfigGet**	Get flash configuration for Node
mpiNode**FlashConfigSet**	Set flash configuration for Node
mpiNode**IdnDataGet**	
mpiNode**IdnDataSet**	
mpiNode**IdnFieldGet** / meiNode**IdnFieldGet**	
mpiNode**IdnFieldSet** / meiNode**IdnFieldSet**	
mpiNode**ServiceProcedure**	Execute the procedure in the node described by idn
mpiNode**Status**	Get the status of a Node

Memory Methods

mpiNode**Memory**	Get address of Node memory

Relational Methods

mpiNode**IdnListGet**	Get IdnList associated with a Node
mpiNode**IdnListSet**	Set IdnList associated with a Node
mpiNode**Number**	Get index number of Node
mpiNode**Sercos**	

Other Methods

meiNode**OperationMode**	
meiNode**RealTimeControlWordGet**	
meiNode**RealTimeControlWordSet**	Write the real time control word to a node.
meiNode**RealTimeStatusWordGet**	
meiNode**TelegramType**	

Data Types

MPINode**Config** / MEINode**Config**
MPINode**IdnListType**
MEINode**IdnCyclic**
MEINode**IdnData**
MPINode**Mode**
MEINode**RealTimeBit**
MEINode**RealTimeWord**

Macros

mpiNode**IdnDataGET**
mpiNode**IdnGET**
mpiNode**IdnSET**

mpiNodeCreate

Declaration

```
const mpiNodeCreate(MPISercos sercos,
                    long      number)
```

Required Header

stdmpi.h

Description

NodeCreate creates a node object associated with the node identified by ***number*** located on a SERCOS ring, ***sercos***. NodeCreate is the equivalent of a C++ constructor.

Return Values

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

See Also

mpiNodeValidate | mpiNodeDelete

mpiNodeDelete

Declaration `long mpiNodeDelete(MPINode node)`

Required Header stdmpi.h

Description **NodeDelete** deletes a Node object and invalidates its handle (***node***). *NodeDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	if *NodeDelete* successfully deletes the Node object and invalidates its handle

See Also mpiNodeCreate | mpiNodeValidate

mpiNodeValidate

Declaration `long mpiNodeValidate(MPINode node)`

Required Header stdmpi.h

Description **NodeValidate** validates the Node object and its handle (***node***).

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

See Also mpiNodeCreate | mpiNodeDelete

mpiNodeConfigGet

Declaration

```
long mpiNodeConfigGet(MPINode        node,
                      MPINodeConfig  *config,
                      void           *external)
```

Required Header

stdmpi.h

Description

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

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

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

Return Values

MPIMessageOK	if *NodeConfigGet* successfully writes the Node object's configuration to the structure(s)

See Also

MEINodeConfig | mpiNodeConfigSet

mpiNodeConfigSet

Declaration

```
long mpiNodeFlashConfigSet(MPINode        node,
                           void           *flash,
                           MPINodeConfig  *config,
                           void           *external)
```

Required Header

stdmpi.h

Description

NodeFlashConfigSet sets a Node object's (***node***) flash configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL). The Node object's flash configuration information in external is in addition to the Node object's flash configuration information in ***config***, i.e, the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

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

Return Values

MPIMessageOK	if *NodeFlashConfigSet* successfully sets the Node object's flash configuration using data from the structure(s) flash is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

MEINodeConfig | MEIFlash | mpiNodeConfigGet

mpiNodeFlashConfigGet

Declaration

```
long mpiNodeFlashConfigGet(MPINode        node,
                           void           *flash,
                           MPINodeConfig  *config,
                           void           *external)
```

Required Header

stdmpi.h

Description

NodeFlashConfigGet gets a Node object's (***node***) flash configuration and writes it in the structure pointed to by ***config***, and also writes it in the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

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

Return Values

MPIMessageOK	if *NodeFlashConfigGet* successfully writes the Node object's flash configuration to the structure(s) ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

MEINodeConfig | MEIFlash | mpiNodeFlashConfigSet

mpiNodeFlashConfigSet

Declaration

```
long mpiNodeFlashConfigSet(MPINode        node,
                           void           *flash,
                           MPINodeConfig  *config,
                           void           *external)
```

Required Header

stdmpi.h

Description

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

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

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

Return Values

MPIMessageOK	if *NodeFlashConfigSet* successfully sets the Node object's flash configuration using data from the structure(s) ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

MEINodeConfig | MEIFlash | mpiNodeFlashConfigGet

meiNodeIdnDataGet / mpiNodeIdnDataGET

meiNodeIdnDataGet

Declaration

```
long meiNodeIdnDataGet(MPIControl          control,
                       MEIMotorTypeInfo    *motorInfo,
                       long                count,
                       MEINodeIdnData      *idnData)
```

Required Header stdmei.h

Description **NodeIdnDataGet** reads the idn data and writes it to the structure pointed to by ***idnData***. The data is read from the controller associated with the control object and the nodeNumber specified in the structure pointed to by ***motorInfo***.

control	a handle to a Control object
***motorInfo**	a pointer to a MEIMotorTypeInfo structure
count	number of MEINodeIdnData structures to read
***idnData**	a pointer to an array of MEINodeIdnData structures

Return Values	
MPIMessageOK	if *NodeIdnDataGet* successfully reads the idn data.

meiNodeIdnDataGET

Declaration

```
#define    mpiNodeIdnDataGET(node, idn)
           mpiNodeIdnFieldGet((node), (idn), MPIIdnFieldDATA)
```

Required Header stdmpi.h

Description **NodeIdnDataGET** reads the data field from an ***idn*** located on a ***node***, and writes them into the idn object.

See Also meiNodeIdnDataSet

meiNodeIdnDataSet / mpiNodeIdnDataSET

meiNodeIdnDataSet

Declaration

```
long meiNodeIdnDataSet(MPIControl          control,
                       MEIMotorTypeInfo    *motorInfo,
                       long                count,
                       MEINodeIdnData      *idnData)
```

Required Header stdmei.h

Description **NodeIdnDataSet** writes the idn data from the structure pointed to by ***idnData***. The data is written to the controller associated with the control object and the nodeNumber specified in the structure pointed to by ***motorInfo***.

control	a handle to a Control object
***motorInfo**	a pointer to a MEIMotorTypeInfo structure
count	number of MEINodeIdnData structures to read
***idnData**	a pointer to an array of MEINodeIdnData structures

Return Values	
MPIMessageOK	if *NodeIdnDataSet* successfully writes the idn data.

mpiNodeIdnDataSET

Declaration

```
#define    mpiNodeIdnDataSET(node, idn)
           mpiNodeIdnFieldSet((node), (idn), MPIIdnFieldDATA)
```

Required Header stdmpi.h

Description **NodeIdnDataSET** writes the data fields from an idn object to the idn located on a node.

See Also meiNodeIdnDataGet | meiNodeIdnDataGET

mpiNodeIdnFieldGet / meiNodeIdnFieldGet

mpiNodeIdnFieldGet

Declaration

```
long mpiNodeIdnFieldGet(MPINode     node,
                        MPIIdn      idn,
                        MPIIdnField field)
```

Required Header stdmpi.h

Description **NodeIdnFieldGet** reads the specified ***field*** from an ***idn*** located on a ***node*** and writes it into the idn object.

node	a handle to the Node object
idn	a handle to an Idn object
field	enumeration corresponding to each element in the structure MPIIdnElement

Return Values	
MPIMessageOK	if *NodeIdnFieldGet* successfully reads the specified field.

meiNodeIdnFieldGet

Declaration

```
long meiNodeIdnFieldGet(MPINode     node,
                        MPIIdn      idn,
                        MPIIdnField field)
```

Required Header stdmei.h

Description **NodeIdnFieldGet** reads the specified ***field*** from the ***node*** into the ***idn***.

Return Values	
MPIMessageOK	if *NodeIdnFieldGet* successfully reads the specified field.

See Also mpiNodeIdnFieldSet

mpiNodeIdnFieldSet / meiNodeIdnFieldSet

mpiNodeIdnFieldSet

Declaration

```
long mpiNodeIdnFieldSet(MPIIdn        idn,
                        MPIIdnField   field,
                        void          *value)
```

Required Header stdmpi.h

Description **NodeIdnFieldSet** writes a ***value*** to a specified ***field*** located in the ***idn*** on a node. The node is determined by the association between the idn object and the node object.

idn	a handle to an Idn object
***value**	a pointer to a void

Return Values	
MPIMessageOK	if *NodeIdnFieldSet* successfully writes the specified field.

See Also mpiNodeIdnFieldGet

meiNodeIdnFieldSet

Declaration

```
long meiNodeIdnFieldSet(MPINode     node,
                        MPIIdn      idn,
                        MPIIdnField field)
```

Required Header stdmei.h

Description **NodeIdnFieldSet** writes the specified ***field*** to the ***node*** from the ***idn*** object.

Return Values	
MPIMessageOK	if *NodeIdnFieldSet* successfully writes the specified field.

See Also mpiNodeIdnFieldGet | meiNodeIdnFieldGet

mpiNodeServiceProcedure

Declaration

```
long mpiNodeServiceProcedure(MPINode                  node,
                             MPISercosProcedureAction action,
                             MPIIdn                   idn,
                             unsigned long            *status)
```

Required Header stdmpi.h

Description **NodeServiceProcedure** executes the procedure in the ***node*** described by ***idn***. The extent of the execution of the procedure is defined by the action parameter (see the MPISercosProcedureAction structure for the different actions).

Return Values	
MPIMessageOK	if sucessful in executing action
MPISercosMessagePROCEDURE_ERROR	if an error occurs in the drive during execution
MPISercosMessagePROCEDURE_DATA_INVALID	if procedure data is invalid
MPISercosMessagePROCEDURE_TIMEOUT	if a timeout occurs and the procedure has not completed.

See Also

mpiNodeStatus

Declaration

```
long mpiNodeStatus(MPINode        node,
                   MPINodeStatus  *status,
                   void           *external)
```

Required Header

stdmpi.h

Description

NodeStatus writes a Node's (***node***) status into the structure pointed to by ***status***, and also into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

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

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

Return Values

MPIMessageOK	if *NodeStatus* successfully writes the Node's status to the structure(s)

See Also

mpiNodeMemory

Declaration

```
long mpiNodeMemory(MPINode    node,
                   void       **memory)
```

Required Header

stdmpi.h

Description

NodeMemory writes an address [that is used to access Node (***node***) memory] to the contents of ***memory***.

Return Values

MPIMessageOK	if *NodeMemory* successfully writes the Node memory address to the contents of ***memory***

See Also

mpiNodeIdnListGet

Declaration

```
long mpiNodeIdnListGet(MPINode            node,
                       MPINodeIdnListType type,
                       MPIIdnList         idnList)
```

Required Header

stdmpi.h

Description

NodeIdnListGet gets the Node's (***node***) Idn list specified by ***type*** and writes it to ***idnList***. The existing Idn list at ***idnList*** (if any) will be deleted.

Return Values	
MPIMessageOK	if *NodeIdnListGet* successfully gets and writes the Node's Idn list to ***idnList***

See Also

mpiNodeIdnListSet

mpiNodeIdnListSet

Declaration

```
long mpiNodeIdnListSet(MPINode            node,
                       MPINodeIdnListType type,
                       MPIIdnList         idnlist)
```

Required Header

stdmpi.h

Description

NodeIdnListSet sets the Node's (***node***) Idn list (specified by ***type***) using data from ***idnList***. The existing Idn list of the Node (if any) will be deleted.

Return Values	
MPIMessageOK	if *NodeIdnListSet* successfully sets the Node's Idn list using data from ***idnList***

See Also

mpiNodeIdnListGet

mpiNodeNumber

Declaration

```
long mpiNodeNumber(MPINode node,
                   long    *number)
```

Required Header

stdmpi.h

Description

NodeNumber writes the index of a Node object (***node***, on the motion controller that the Node object is associated with) to the location pointed to by ***number***.

Return Values

MPIMessageOK	if *NodeNumber* successfully writes the Node object's index to the location

See Also

mpiNodeSercos

Declaration `const MPISercos mpiNodeSercos(MPINode node)`

Required Header stdmpi.h

Description **NodeSercos** returns a handle to Sercos object passed to the Node Object in the mpiNodeCreate() function call.

Return Values	
MPIMessageOK	if successful

See Also mpiNodeCreate

meiNodeOperationMode

Declaration

```
long meiNodeOperationMode(MPINode node,
                          long    *operationMode)
```

Required Header

stdmei.h

Description

NodeOperationMode returns in ***operationMode*** the sercos defined operation mode based on the MEI-defined operation mode.

Return Values

MPIMessageOK	if successful

See Also

meiNodeRealTimeControlWordGet

Declaration

```
long meiNodeRealTimeControlWordGet(MPINode node,
                                   long    *realTimeControlWord)
```

Required Header

stdmei.h

Description

NodeRealTimeControlWordGet reads the real time control word from a ***node*** and writes it into the location pointed to by ***realTimeControlWord***.

node	a handle to a Node object
***realTimeControlWord**	a pointer to a long

Return Values

MPIMessageOK	if *NodeRealTimeControlWordGet* successfully reads the real time control word.

See Also

mpiNodeRealTimeControlWordSet

meiNodeRealTimeControlWordSet

Declaration

```
long meiNodeRealTimeControlWordSet(MPINode node,
                                   long    realTimeControlWord)
```

Required Header

stdmei.h

Description

NodeRealTimeControlWordSet writes the real time control word to a node.

node	a handle to a Node object
***realTimeControlWord**	a long value

Return Values

MPIMessageOK	if *NodeRealTimeControlWordSet* successfully writes the real time control word.

See Also

mpiNodeRealTimeControlWordGet

meiNodeRealTimeStatusWordGet

Declaration

```
long meiNodeRealTimeStatusWordGet(MPINode node,
                                  long    *realTimeStatusWord)
```

Required Header

stdmei.h

Description

NodeRealTimeStatusWordGet reads the real time status word from a ***node*** and writes it into the location pointed to by ***realTimeStatusWord***.

node	a handle to a Node object
***realTimeStatuslWord**	a point to a long

Return Values

MPIMessageOK	if *NodeRealTimeStatusWordGet* successfully reads the real time status word.

See Also

meiNodeTelegramType

Declaration

```
long meiNodeTelegramType(MPINode node,
                         long    *telegramType)
```

Required Header

stdmei.h

Description

NodeTelegramType reads the telegram type from a ***node*** and writes it into the location pointed to by ***telegramType***.

node	a handle to a Node object
***telegramType**	a pointer to a long

Return Values

MPIMessageOK	if *NodeTelegramType* successfully reads the telegram type.

See Also

MPINodeConfig / MEINodeConfig

MPINodeConfig

```
typedef struct MPINodeConfig {
    long        address;        /* SERCOS address 1 ..254 */
    long        motorNumber;
    long        filterNumber;
    MPINodeMode mode;
} MPINodeConfig;
```

Description

address	In SERCOS each Node has a unique address on the ring(1 - 254).
motorNumber	The number of the MPIMotor associated with the SERCOS Node (1:1).
filterNumber	The number of the Filter associeated with the SERCOS Node (1:1).
mode	The position control algorithm used by the node.

MEINodeConfig

```
typedef struct MEINodeConfig {
    long                Enabled;
    long                StatusInvertMask;
    MEINodeIdnCyclic    At[MEIXmpNodeCyclicCountMAX];
    MEINodeRealTimeBit  RealTimeStatus[MEIXmpNodeRealTimeBitCountMAX];
    MEINodeRealTimeWord RealTimeStatusWord;
    long                ControlInvertMask;
    MEINodeIdnCyclic    Mdt[MEIXmpNodeCyclicCountMAX];
    MEINodeRealTimeBit  RealTimeControl[MEIXmpNodeRealTimeBitCountMAX];
    long                HostRealTimeControlWord;
    MEINodeRealTimeWord RealTimeControlWord;
} MEINodeConfig;
```

Description

NodeConfig is the configuration data that defines the operation of a Node.

Enabled	this variable reports whether or not the Node is enabled.
StatusInvertMask	a bit mask that is used to identify which status bits are inverted.
At	a list of IDNs that make up the AT data for a Node.
RealTimeStatus	in the RealTimeStatusWord there are 2 bits that can be assigned through IDN's to report certain events.
RealTimeStatusWord	located in the AT, this word is a bit mask that is used to report the status of a node.
ControlInvertMask	a bit mask that is used to identify which control bits are inverted.
Md	a list of IDNs that make up the MDT data that is sent to a Node.
RealTimeControl	in the RealTimeControlWord there are 2 bits that can be assigned through IDNs to trigger certain actions or report status from the controller to the Node.

HostRealTimeControlWord	
RealTimeControlWord	located in the MDT, this word is a bit mask that is used to control the node and report controller status.

See Also

mpiNodeConfigGet | mpiNodeConfigSet

MPINodeIdnListType

MPINodeIdnListType

```
typedef enum {
    MPINodeIdnListTypeINVALID,

    MPINodeIdnListTypePHASE2,
    MPINodeIdnListTypePHASE3,
    MPINodeIdnListTypeAMPLIFIER,
    MPINodeIdnListTypeMASTER_DATA,

} MPINodeIdnListType;
```

Description

MPINodeIdnListTypePHASE2	List of IDN's that must be configured in phase 2
MPINodeIdnListTypePHASE3	List of IDN's that must be configured in phase 3
MPINodeIdnListTypeAMPLIFIER	List of IDN's that are transmitted cyclically in the AT
MPINodeIdnListTypeMASTER_DATA	List of IDN's that are transmitted cyclically in the MDT

See Also

MEINodeIdnCyclic

MEINodeIdnCyclic

```
typedef struct MEINodeIdnCyclic {
    long        IDNumber;
    long        UsageFlags;     /* Gain Scheduling, etc… */
    union {
    long        *Src;           /* Host address */
    long        *Dst;           /* Host address */
    } Data;
} MEINodeIdnCyclic;
```

Description

IDNumber	the number of the IDN
UsageFlags	a bit mask that defines how the IDN is used
Data	a pointer to where the data is located

See Also

MEINodeIdnData

MEINodeIdnData

```
typedef struct MEINodeIdnData {
    MPIIdnNumber        idnNumber;
    MPIIdnData          idnData;
} MEINodeIdnData;
```

Description

idnNumber	The number of an IDN.
idnData	The data of an IDN.

See Also

mpiMotionCreate | mpiControlConfigGet

MPINodeMode

MPINodeMode

```
typedef enum {
    MPINodeModeINVALID,

    MPINodeModeNONE,

    /*   servo_command_feedback         */
    MPINodeModeOPENLOOP_TORQUE,
    MPINodeModeOPENLOOP_VELOCITY,
    MPINodeModeOPENLOOP_POSITION_MOTOR,
    MPINodeModeOPENLOOP_POSITION_EXTERNAL,
    MPINodeModeOPENLOOP_POSITION_DUAL,

    MPINodeModePOSITION_TORQUE_MOTOR,
    MPINodeModePOSITION_TORQUE_EXTERNAL,
    MPINodeModePOSITION_VELOCITY_MOTOR,
    MPINodeModePOSITION_VELOCITY_EXTERNAL,

} MPINodeMode;
```

Description

NodeMode defines the different types of control algorithms that are supported under SERCOS and can be used by a node.

See Also

MEINodeRealTimeBit

MEINodeRealTimeBit

```
typedef struct MEINodeRealTimeBit {
    union {
        long    *Src;   /* Host address */
        long    *Dst;   /* Host address */
    } Data;
    long        BitMask;
} MEINodeRealTimeBit;
```

Description

The SERCOS RealTimeBit can be in the **Status Word** or **Control Word**. This is why Data can be either a ***Src*** or ***Dst***. These pointers point to a 32-bit word. Since we are only looking at a bit, ***BitMask*** is used to mask off all the bits that are not of interest.

See Also

For more information, please refer to a SERCOS Specification Manual.

MEINodeRealTimeWord

MEINodeRealTimeWord

```
typedef struct MEINodeRealTimeWord {
    long                 InvertMask;
    MEINodeRealTimeBit   Bit[MEIXmpNodeRealTimeWordBitCountMAX];
} MEINodeRealTimeWord;
```

Description

Each Node has a **NodRealTimeWord** for Control and Status, each of which is made up of 16 (MEIXmpNodeRealTimeWordBitCountMAX) bits. The ***InvertMask*** is used to specify the polarity of each bit.

See Also

For more information, please refer to a SERCOS Specification Manual.

mpiNodeIdnDataGET

Declaration

```
#define     mpiNodeIdnDataGET(node, idn)
            mpiNodeIdnFieldGet((node), (idn), MPIIdnFieldDATA)
```

Required Header

stdmpi.h

Description

NodeIdnDataGET gets all of the fields from an ***idn*** located on a ***node***, and writes them into the idn object.

See Also

mpiNodeIdnGET

Declaration

```
#define     mpiNodeIdnGET(node, idn)
            mpiNodeIdnFieldGet((node), (idn), MPIIdnFieldALL)
```

Required Header

stdmpi.h

Description

NodeIdnGET gets all of the fields from an ***idn*** located on a ***node***, and writes them into the idn object.

See Also

mpiNodeIdnSET

Declaration

```
#define     mpiNodeIdnSET(node, idn)
            mpiNodeIdnFieldSet((node), (idn), MPIIdnFieldDATA)
```

Required Header

stdmpi.h

Description

NodeIdnSET sets all of the fields from an ***idn*** located on a ***node***, and writes them into the idn object.

See Also