Message Objects

Introduction

The **Message** module manages text strings associated with the error/warning code return values that can be returned from MPI methods. Each method return value has a unique value which can be decoded using the Message methods and macros.

For example, a return value of MEISynqNetMessagePLL_ERROR has a defined value of 0x192E. Passing this value to the mpiMessage(...) function returns the following text string:

"SynqNet: node PLL unable to lock with drive"

If MEISynqNetMessagePLL_ERROR was the last return value received from an MPI method, the string returned from mpiMessage(...) would also contain extended message information. The extended message information provides greater insight into the problem that has just occurred. In this case, the extended message information would indicate which node had the PLL locking problem:

"SynqNet: node PLL unable to lock with drive : Node 3"

Notice that the extended message information is delimited from the more generic message by a colon surrounded by spaces " : "

Extended message information may not always be returned. Extended information is only available for the very last return value from the MPI -- if a second (non-zero) return value has been returned between the time when the first return value was returned and the call to mpiMessage(...), then no extended information would be returned. In addition, not all messages return extended error information.

Methods

Configuration and Information Methods

mpiMessage	
mpiMessage**Function**	Associate message text with a function

Data Types

MPIMessage / MEIMessage

MPIMessage**Function**

Macros

mpiMessage**ID**

mpiMessage**MODULE**

mpiMessage**NUMBER**

mpiMessage

Declaration

```
const char *mpiMessage(long  messageId,
                       char  *messageText)
```

Required Header: stdmpi.h

Description

mpiMessage returns the text message associated with ***messageId*** and copies the text to ***messageText*** if ***messageText*** is not NULL.

If "messageText" is	Then
NULL	*Message* returns a pointer to message text resident in the library
not NULL	*Message* returns the pointer ***messageText***, which must point to a buffer large enough to hold the message text (that will be copied into it)

Return Values	
NULL	if messageId is invalid
Pointer to an empty string (" ")	if the messge utility is not enabled or the message function pointer associated with *messageId* is NULL[1].
Pointer to a non-zero-length string	if the mpiMessage successsfully retrieves the message associated with *messageId*.

1 - *ObjectCreate(...)* methods register the appropriate message function associated with *Object's* module. If the MPI returns a certain ***messageId***, then the appropriate message function should already be loaded and mpiMessage should return the appropriate error code.

Only an application whose job is to translate message IDs specified by some source other than the MPI (user or a log file) to error messages will need to call the create methods of all MPI objects. MEI recommends using the message.exe utility for this task, since it is already designed to manage these issues.

See Also

mpiMessageFunction | message.exe utility

mpiMessageFunction

Declaration

```
long mpiMessageFunction(MPIModuleId        moduleId,
                        MPIMessageFunction function)
```

Required Header: stdmpi.h

Description

mpiMessageFunction registers ***function*** as the function to be called by mpiMessage(...) (in order to obtain the text for a message associated with module ***moduleId***).

MessageFunction is typically called internally by object create methods. Applications generally do not need to call *MessageFunction* directly.

Return Values	
MPIMessageOK	

See Also

mpiMessage

MPIMessage / MEIMessage

Definition: MPIMotorMessage

```
typedef enum {
    MPIMessageOK,

    MPIMessageARG_INVALID,
    MPIMessagePARAM_INVALID,
    MPIMessageHANDLE_INVALID,

    MPIMessageNO_MEMORY,

    MPIMessageOBJECT_FREED,
    MPIMessageOBJECT_NOT_ENABLED,
    MPIMessageOBJECT_NOT_FOUND,
    MPIMessageOBJECT_ON_LIST,
    MPIMessageOBJECT_IN_USE,

    MPIMessageTIMEOUT,

    MPIMessageUNSUPPORTED,
    MPIMessageFATAL_ERROR,

    MPIMessageFILE_CLOSE_ERROR,
    MPIMessageFILE_OPEN_ERROR,
    MPIMessageFILE_READ_ERROR,
    MPIMessageFILE_WRITE_ERROR,
    MPIMessageFILE_MISMATCH,
} MPIMessage;
```

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

Description

MPIMessageOK

OK

MPIMessageARG_INVALID

Argument invalid

This message code is returned by a method when one or more arguments fail an integrity check. The following integrity checks are made, depending on the argument types:
- Pointers and addresses are not NULL.
- Memory addresses are within a valid range.
- Object maps have a valid number of members.

Possible Causes:
Number of axes on the motion commanded exceeds MEIXmpMAX_COORD_AXES.

MPIMessagePARAM_INVALID

Parameter invalid.

Parameters are the data inside an argument. This message code is returned by a method when one or more parameters fail an integrity check. The following integrity checks are made, depending on the parameter types:
- Pointers and addresses in structures are not NULL.
- Memory addresses in structures are within a valid range.
- Object numbers are within a valid range.
- Values in structures are within a valid range.

Please see possible causes for receiving this message.

MPIMessageHANDLE_INVALID

An object handle is not valid. This message code is returned by a method when the object handle argument is NULL. To correct this problem, create an object using an object create method. For example, to create an axis object use mpiAxisCreate(...).

MPIMessageNO_MEMORY

Memory is not available. This message code is returned by a method when an object, thread or buffer is not created due to a memory allocation failure. All host memory allocation occurs through meiPlatformAlloc(...). To correct this problem, check your host computer resources and remove unused components or add more memory. Note: Some methods may make several memory allocations or may call other methods that allocate memory. If a memory allocation fails, the previous allocations will not be freed.

MPIMessageOBJECT_FREED

The object has been freed. This message code is set within the object when the object is deleted. This message code is used internally. This message code is returned by an object delete method if the object has already been deleted. To prevent this problem, make sure to delete objects only one time.

MPIMessageOBJECT_NOT_ENABLED

The object is not active in the controller. This message code is returned by a method if real-time data or a handshake is required between the MPI and the controller, and the specified object is not enabled in the controller. To correct this problem, use mpiControlConfigSet(...) to enable the object, by setting the object count to greater than the specified object number. For example, to enable motor objects 0 to 3, set motorCount to 4.

MPIMessageOBJECT_NOT_FOUND

The object is not found on an object list. This message code is returned by a method when an object does not exist on the specified list or the object list does not exist. This message code can be returned from methods that insert or remove objects from a list. To correct this problem, specify an object that exists on the list or specify a different object list containing the object. To prevent object creation/ deletion problems, delete objects in the reverse order they were created.

MPIMessageOBJECT_ON_LIST

The object is a member of a list. This message code is returned when a method tries to add an object to a list and the object already exists on the list. It is also returned when a method tries to delete an object when the object exists on a list. For example, if an axis object is a member of a list of axis objects or an axis object is associated with a motion object, calling mpiAxisDelete(...) will return the object on list message code. To correct this problem when adding an object to a list, check to make sure the object is not already a member. To correct this problem when deleting an object, remove the object from the list and delete the parent object. To prevent object creation/deletion problems, delete objects in the reverse order they were created.

MPIMessageOBJECT_IN_USE

This message is returned when an operation cannot be completed because the object is currently in use. For example, when threadDelete(…) from the apputil library attempts to delete a thread that is currently running, MPIMessageOBJECT_IN_USE will be returned. To correct this problem, make sure the object is not in use before you attempt the operation.

MPIMessageTIMEOUT

The wait time has expired. This message code is returned by a method waiting for a response from the controller, a hardware resource, or a semaphore lock and the maximum wait time value expired. The library uses timeout values to prevent lock-up conditions when unexpected behavior occurs. To correct this problem, check the hardware health, power, and network connections.

MPIMessageUNSUPPORTED

The software or controller does not support a feature. This message code is returned by a method if the MPI library, controller, or network device does not support a feature.

Possible Causes:
Unsupported motion type specified.

List of supported motion types:
MPIMotionTypePT
MPIMotionTypePTF
MPIMotionTypePVT
MPIMotionTypePVTF
MPIMotionTypeSPLINE
MPIMotionTypeBESSEL
MPIMotionTypeBSPLINE
MPIMotionTypeBSPLINE2
MPIMotionTypeS_CURVE
MPIMotionTypeS_CURVE_JERK
MPIMotionTypeTRAPEZOIDAL
MPIMotionTypeVELOCITY
MPIMotionTypeVELOCITY_JERK
MEIMotionTypeFRAME

See MPIMotionType/MEIMotionType.

MPIMessageFATAL_ERROR

The software or hardware failed. This message code is returned by a method when an inexplicable data value is read from the controller or host memory. This message code indicates a serious problem with the host computer, memory, or controller. Contact an MEI applications engineer if you receive this message code.

MPIMessageFILE_CLOSE_ERROR

An error occurred when closing a file. This message code is returned by a method that fails to close a file. Internally, files are closed using meiPlatformClose(...) which makes a platform specific call. File closing errors indicate a missing file, bad file, or other file system problems.

MPIMessageFILE_OPEN_ERROR

An error occurred when opening a file. This message code is returned by a method that fails to open a file. Internally, files are opened using meiPlatformOpen(...) which makes a platform specific call. File opening errors indicate a missing file, bad file, or other file system problems.

MPIMessageFILE_READ_ERROR

An error occurred when reading from a file. This message code is returned by a method that fails to read from a file. Internally, files are read using meiPlatformFileRead(...) which makes a platform specific call. File reading errors indicate a bad file or other file system problems.

MPIMessageFILE_WRITE_ERROR

An error occurred when writing to a file. This message code is returned by a method that fails to write to a file. Internally, files are written using meiPlatformFileWrite(...) which makes a platform specific call. File writing errors indicate a bad file or other file system problems.

MPIMessageFILE_MISMATCH

The file being downloaded does not match the installed hardware. For example, if a user attempts to download an XMP file to a ZMP controller or a ZMP file to an XMP controller , the MPI will return error code MPIMessageFILE_MISMATCH.

Definition: MEIMotorMessage

```
typedef enum {
    MEIMotorMessageMOTOR_NOT_ENABLED,
    MEIMotorMessageSECONDARY_ENCODER_NA,
    MEIMotorMessageHARDWARE_NOT_FOUND,
    MEIMotorMessageSTEPPER_INVALID,
    MEIMotorMessageDISABLE_ACTION_INVALID,
    MEIMotorMessagePULSE_WIDTH_INVALID,
    MEIMotorMessageFEEDBACK_REVERSAL_NA,
    MEIMotorMessageFILTER_DISABLE_NA,
} MEIMotorMessage;
```

Required Header: stdmei.h

Description

MEIMotorMessage is an enumeration of Motor error messages that can be returned by the MPI library.

MEIMotorMessageMOTOR_NOT_ENABLED

The motor number is not active in the controller. This message code is returned by MPIMotorEventConfig if the specified motor is not enabled in the controller. To correct the problem, use mpiControlConfigSet(…) to enable the motor object, by setting the motorCount to greater than the motor number. For example, to enable motor 0 to 3, set motorCount to 4.

MEIMotorMessageSECONDARY_ENCODER_NA

The motor's secondary encoder is not available. This message code is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the encoder fault trigger is configured for a secondary encoder when the hardware does not support a secondary encoder. To correct the problem, do not select the secondary encoder when configuring the encoder fault conditions.

MEIMotorMessageHARDWARE_NOT_FOUND

The motor object's hardware resource is not available. This message code is returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the node hardware for the motor is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor objects. An application should not configure a motor object if there is no mapped hardware to receive the service commands. To correct this problem, verify that all expected nodes were found. Use meiSqNodeInfo(…) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEIMotorMessageSTEPPER_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessageDISABLE_ACTION_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessagePULSE_WIDTH_INVALID

The motor object stepper pulse width is not valid. The pulse width must be no greater than 1 millisecond and no less than 100 nanoseconds.

MEIMotorMessageFEEDBACK_REVERSAL_NA

Feedback reversal is not applicable or is not supported for the feedback type specified.

MEIMotorMessageFILTER_DISABLE_NA

Disabling of filters is not applicable or is not supported for the feedback type specified.

See Also

MPIMessageFunction

Definition

```
typedef const char *(*MPIMessageFunction)(long);
```

Description

MPIMessageFunction is the type definition for the callback function used by mpiMessage(...). A default callback function is provided internally to all MPI/MEI modules, but an application can also be written to override it and provide a custom message function instead.

See Also

mpiMessage | mpiMessageFunction

mpiMessageID

Declaration

```
#define mpiMessageID(module number) \
((long)((((module) & MPIModuleIdMAX) << 8) | \ ((number) & 0xFF)))
```

Required Header: stdmpi.h

Description

mpiMessageID converts the message module value and number to a unique message identification value.

See Also

mpiMessage

mpiMessageMODULE

Declaration

```
#define mpiMessageMODULE(messageId) \
(((messageId) & (MPIModuleIdMAX << 16)) >> 16)
```

Required Header: stdmpi.h

Description

mpiMessageMODULE converts the message identification value to the message module value.

See Also

mpiMessage

mpiMessageNUMBER

Declaration

```
#define mpiMessageNUMBER(messageId) ((messageId) & 0xFF)
```

Required Header: stdmpi.h

Description

mpiMessageNUMBER converts the message identification value to the message number.

See Also

mpiMessage