Platform Objects

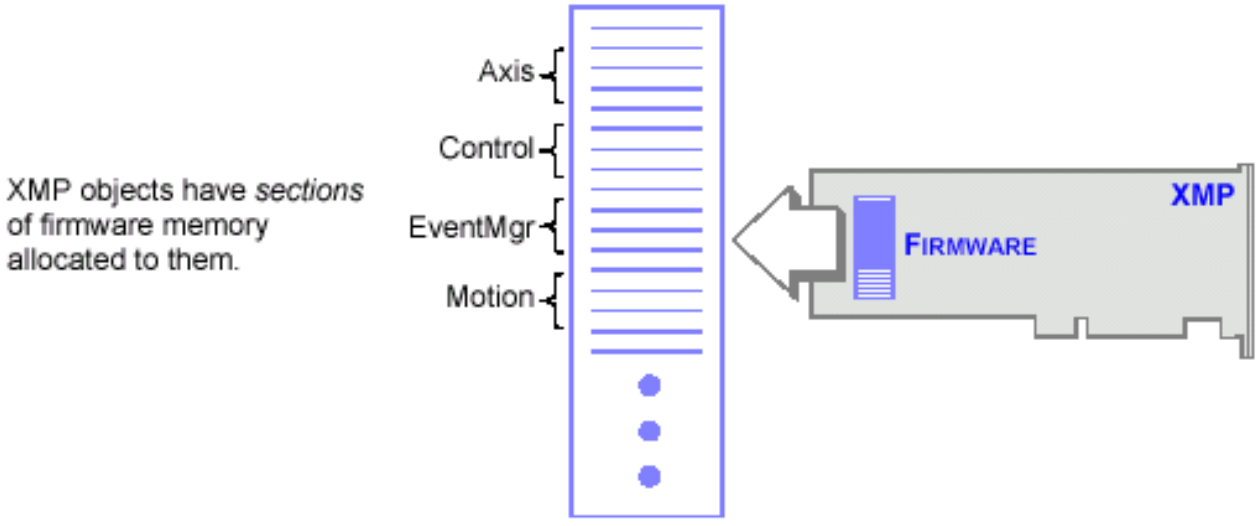
Introduction

The **Platform** module provides a common interface to platform-specific functionality, such as memory allocation, resource locking, interrupts, signalling, and others.

The Platform object provides low-level platform-specific functionality and depends upon the combination of the operating system and the C compiler used for development. The Platform module was written to provide platform-independent access functions for use by the MPI. Unless your application needs to be written for compatibility with different platforms, MEI encourages the use of OS-specific functions. If an MEIPlatform object handle is required, one should obtain this handle from the MPIControl method meiControlPlatform.

WARNING!
Do NOT attempt to use the (intentionally undocumented) method, meiPlatformCreate(). Using this method will interfere with the inner workings of the MPI.

The mei*Object*Give/Take(...) methods all use the meiPlatformLockGive**/**Take(...) methods. When you take a lock, you take exclusive access to the resource (i.e., the section of XMP firmware memory associated with that Object). When you give a lock, you release (give up) that exclusive access. Think of it as TakeAccessOf and GiveUpAccess.



Methods

meiPlatform**Alloc**	Allocate system memory.
meiPlatform**AssertSet**	Set an assertion handling function to be used by the MPI library.

meiPlatform**Atof**	Convert a numeric string to a double.
meiPlatform**Atol**	Convert a numeric string to a long.
meiPlatform**ExmpTempGet**	
meiPlatform**ExmpTempInit**	
meiPlatform**FileClose**	Close a file handle.
meiPlatform**FileOpen**	Open a file handle.
meiPlatform**FileRead**	Read data from a file handle created by meiPlatformFileOpen.
meiPlatform**FileWrite**	Writes data to a file whose handle was created by meiPlatformFileOpen.
meiPlatform**FirmwareAddress16To32**	
meiPlatform**FirmwareAddress32To16**	
meiPlatform**FirmwareAddress32To64**	
meiPlatform**FirmwareAddress64To32**	
meiPlatform**Free**	Free system memory.
meiPlatform**HostAddress32To64**	
meiPlatform**HostAddress64To32**	
meiPlatform**Key**	Return an input character if an input character is available.
meiPlatform**MemoryGet64**	
meiPlatform**MemorySet64**	
meiPlatform**MemoryToFirmware**	Convert a host memory address to a controller memory address.
meiPlatform**MemoryToHost**	Covert a controller memory address to a host memory address.
meiPlatform**Sleep**	Put the current thread to sleep for the number of milliseconds specified.
meiPlatform**ProcessId**	Return the process identification number of the current process.
meiPlatform**TimerCount**	Write to ticks the current timer count.
meiPlatform**TimerFrequency**	Write to frequency the timer frequency of the current platform.
meiPlatform**Trace**	Display printf(...)-style trace information
meiPlatform**TraceEol**	Set the end-of-line (eol) to be used by meiPlatformTrace(...).
meiPlatform**TraceFile**	Redirect trace output.
meiPlatform**TraceFunction**	Display the trace output.
meiPlatform**Word64Orient**	

Data Types

MEIPlatform**BoardType**
MEIPlatform**FileMode**
MEIPlatform**Info**
MEIPlatform**Message**

Constants

MEIPlatform**ControlCountMax**
MEIPlatform**InfoCHAR_MAX**

meiPlatformAlloc

Declaration

```
void meiPlatformAlloc(long   size)
```

Required Header: stdmei.h

Description

meiPlatformAlloc allocates system memory. meiPlatformAlloc will return NULL upon failure to allocate memory.

size	the number of bytes to allocate.

See Also

meiPlatformFree

meiPlatformAssertSet

Declaration

```
void meiPlatformAssertSet(void (MPI_DEF2 *func)
                          (const char *file, long line));
```

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

Description

meiPlatformAssertSet sets an assertion handling function to be used by the MPI library.

When an assertion occurs, the filename and line number where that assertion happened will be passed to the assertion handling function. If no assertion handling function is set, information about the assertion will be reported to stdout and the C function, exit() will be called with an argument of 1.

An MPI library assertion should never occur. If it does, please contact MEI's technical support.

NOTE: meiPlatformAssertSet is only fully implemented for Windows operating systems.

platform	the handle to the controller's platform object. This should be obtained from meiControlPlatform(...).
firmware	the controller memory address to be converted.
\host	the location where the host memory address will be written.

See Also

meiControlPlatform

meiPlatformAtof

Declaration

```
double meiPlatformAtof(const char *ascii)
```

Required Header: stdmei.h

Description

meiPlatformAtof converts a numeric string to a double. This function returns the converted value as a double.

***ascii**	string to be converted

Returns
Converted the numeric text string ascii to a ***long*** and returned it.

See Also

meiPlatformAtol

meiPlatformAtol

Declaration

```
long meiPlatformAtol(const char *ascii)
```

Required Header: stdmei.h

Description

meiPlatformAtol converts a numeric string to a long. This function returns the converted value as a long.

***ascii**	string to be converted

Returns
Converted the numeric text string ascii to a ***long*** and returned it

See Also

meiPlatformAtof

meiPlatformExmpTempGet

Declaration

```
long meiPlatformExmpTempGet(MEIPlatform    platform,
                            long*          temp)
```

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

Description

meiPlatformExmpTempGet retrieves the internal temperature of the eXMP-SynqNet controller. This function should be used to poll the temperature to be sure it is within the allowable range. Hardware problems such as a broken fan, could cause the temperature of the eXMP to rise to a potentially unsafe level. When an unsafe temperature level has been detected, a shutdown routine should be executed by the motion application.

meiPlatformExmpTempInit(...) must be called once before polling with an meiPlatformExmpTempGet (...).

NOTE: This function is only supported on systems using an eXMP-SynqNet.

platform	the handle to the Platform object.
temp	the temperature (in Celsius) of the eXMP-SynqNet is returned in this variable.

See Also

meiPlatformExmpTempInit

eXMP-SynqNet Hardware

meiPlatformExmpTempInit

Declaration

```
long meiPlatformExmpTempInit(MEIPlatform    platform)
```

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

Description

meiPlatformExmpTempInit initializes the temperature reading routine on the eXMP-SynqNet controller. meiPlatformExmpTempInit must be called once before polling with an meiPlatformExmpTempGet(...).

NOTE: This function is only supported on systems using an eXMP-SynqNet.

platform	the handle to the Platform object.

See Also

meiPlatformExmpTempGet

eXMP-SynqNet Hardware

meiPlatformFileClose

Declaration

```
long meiPlatformFileClose(long   file)
```

Required Header: stdmei.h

Description

meiPlatformFileClose closes a file handle. meiPlatformFileClose is a platform independent replacement for the C function fclose().

file	the file handle to be closed.

See Also

meiPlatformFileOpen

meiPlatformFileOpen

Declaration

```
long meiPlatformFileOpen(const  char            *fileName,
                         MEIPlatformFileMode    mode)
```

Required Header: stdmei.h

Description

meiPlatformFileOpen opens a file handle. meiPlatformFileOpen is a platform independent replacement for the C function fopen().

fileName	the name of the file to open.
mode	the access mode used to open the file. Different MEIPlatformFileMode values may be or'ed together to produce the desired mode. For example: `MEIPlatformFileModeREAD | MEIPlatformFileModeBINARY)`

Returns
meiPlatformFileOpen returns the newly created file handle.

See Also

meiPlatformFileClose | meiPlatformFileRead | meiPlatformFileWrite | MEIPlatformFileMode

meiPlatformFileRead

Declaration

```
long meiPlatformFileRead(long   file,
                         char   *buffer,
                         long   byteCount)
```

Required Header: stdmei.h

Description

meiPlatformFileRead reads data from a file handle created by meiPlatformFileOpen. meiPlatformFileRead is a platform independent replacement for the C function fread().

file	the handle of the file from which data will be read.
buffer	the storage location where data from the file will be written to.
byteCount	the number of bytes to read from the file.

See Also

meiPlatformFileOpen | meiPlatformFileWrite

meiPlatformFileWrite

Declaration

```
long meiPlatformFileWrite(long          file,
                          const  char   *buffer,
                          long          byteCount)
```

Required Header: stdmei.h

Description

meiPlatformFileRead writes data to a file whose handle was created by meiPlatformFileOpen. meiPlatformFileWrite is a platform independent replacement for hte C function fwrite().

file	the handle of the file to which data will be written.
buffer	the location of the data to be written to the file.
byteCount	the number of bytes to be written to the file.

See Also

meiPlatformFileOpen | meiPlatformFileRead

meiPlatformFirmwareAddress16To32

Declaration

```
long meiPlatformFirmwareAddress16To32(MEIPlatform  platform,
                                      void         **firmwareAddress)
```

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

Description

meiPlatformFirmwareAddress16To32 takes a firmware address that is in the 16-bit space of the XMP and converts it to the 32-bit space. Conversion for an XMP is done by dividing the ***firmwareAddress*** by 2. No conversion is done for a ZMP.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress32To16

meiPlatformFirmwareAddress32To16

Declaration

```
long meiPlatformFirmwareAddress32To16(MEIPlatform  platform,
                                      void         **firmwareAddress,
                                      long         lowWord)
```

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

Description

meiPlatformFirmwareAddress32To16 takes a firmware address that is in the 32-bit space of the XMP and converts it to the 16-bit space.
Set ***lowWord*** to 1 to tell the function to return the 16-bit address of the lower 16-bit word.
Set ***lowWord*** to 0 to get the 16-bit address of the upper 16-bit word.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.
lowWord	**lowWord set to 1** tells the function to return the 16-bit address of the lower 16-bit word. **lowWord set to 0** gets the 16-bit address of the upper 16 bit word.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress16To32

meiPlatformFirmwareAddress32To64

Declaration

```
/* calculates base address of 64 bit entity given a pointer to
one of the 32 bit words within the 64 bit entity */
long meiPlatformFirmwareAddress32To64(MEIPlatform  platform,
                                      void         **firmwareAddress,
                                                    /* address of 32 bit entity */
                                      long         lowWord)
                                                   /* set to 1 if given address
                                                   is the low word */
```

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

Description

meiPlatformFirmwareAddress32To64 takes a pointer to a 32-bit word within a 64-bit entity and returns a pointer to the base address of the 64-bit entitiy. Set ***lowWord*** to 1 if the given address points to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.
lowWord	Set lowWord to 1 if the given address points to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress64To32

meiPlatformFirmwareAddress64To32

Declaration

```
long meiPlatformFirmwareAddress64To32(MEIPlatform  platform,
                                      void         **firmwareAddress,
                                      long         lowWord)
```

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

Description

meiPlatformFirmwareAddress64To32 takes a pointer to a 64-bit entity and returns a pointer to one of the 32-bit words within the 64-bit entity. Set ***lowWord*** to 1 to get a pointer to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	Pointer to a valid firmware address from the firmware's perspective.
lowWord	Set lowWord to 1 to get a pointer to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformFirmwareAddress32To64

meiPlatformFree

Declaration

```
long meiPlatformFree(void   *alloc,
                     long   size)
```

Required Header: stdmei.h

Description

meiPlatformFree frees system memory.

alloc	the address of the memory to free.
size	the number of bytes to free. This must match the size specified for meiPlatformAlloc when alloc was allocated.

See Also

meiPlatformAlloc

meiPlatformHostAddress32To64

Declaration

```
/* calculates base address of 64 bit entity given a pointer to
one of the 32 bit words within the 64 bit entity */
long meiPlatformHostAddress32To64(MEIPlatform  platform,
                                  void         **firmwareAddress,
                                               /* address of 32 bit entity */
                                  long         lowWord)
                                               /* set to 1 if given address
                                               is the low word */
```

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

Description

meiPlatformHostAddress32To64 takes a pointer to a 32-bit word within a 64-bit entity and returns a pointer to the base address of the 64-bit entitiy. Set ***lowWord*** to 1 if the given address points to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	a pointer to a valid firmware address from the host's perspective.
lowWord	set lowWord to 1 if the given address points to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformHostAddress64To32

meiPlatformHostAddress64To32

Declaration

```
long meiPlatformHostAddress64To32(MEIPlatform  platform,
                                  void         **firmwareAddress,
                                               /* address of
                                               generic 64entity */
                                  long         lowWord)
```

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

Description

meiPlatformHostAddress64To32 takes a pointer to a 64-bit entity and returns a pointer to one of the 32-bit words within the 64-bit entity. Set ***lowWord*** to 1 to get a pointer to the ***lowWord*** of the 64-bit entity.

platform	MEIPlatform object
****firmwareAddress**	a pointer to a valid firmware address from the host's perspective.
lowWord	set lowWord to 1 to get a pointer to the lowWord of the 64-bit entity.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	Returned if no XMP or ZMP controller is found.

See Also

meiPlatformHostAddress32To64

meiPlatformKey

Declaration

```
long meiPlatformKey(MPIWait wait)
```

Required Header: stdmei.h

Description

meiPlatformKey returns an input character (typically a keystroke) if an input character is available.

If an input charater is not available, *PlatformKey* waits ***wait*** milliseconds for an input character to become available.

NOTE:
meiPlatformKey is not fully implemented for all operating sytems. For example, in VentureCom's RTX Windows Extensions, a keystroke will be simulated after 10 seconds.

If "wait" is	*Then*
MPIWaitFOREVER (-1)	*PlatformKey* will wait for an input character forever
MPIWaitPOLL (0)	*PlatformKey* will return immediately
a value (not -1 or 0)	*PlatformKey* will wait for an input character for ***wait*** milliseconds

Return Values	
-1	if no input character was available
0	*PlatformKey* has read a non-zero character (typically a function key or other non-ASCII value), and **meiPlatformKey(...)** should be called AGAIN immediately to receive that non-zero character
a value (not -1 or 0) (an ASCII character)	(typically a keystroke) if an input character is available

See Also

meiSqNodeCreate | meiSqNodeValidate

meiPlatformMemoryGet64

Declaration

```
long meiPlatformMemoryGet64(MEIPlatform  platform,
                            void         *dst,
                            const void   *src)
```

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

Description

meiPlatformMemoryGet64 gets (reads) 8 bytes of platform memory (starting at address ***src***) to application memory (starting at address ***dst***).

This function should be used to get/read any 64-bit data from the controller. This function can take up to one foreground cycle to complete. This function will take a platform lock blocking all other tasks from accessing the controller.

platform	the handle to the controller's platform object.
***dst**	address of host data storage area. Storage MUST be two words (64 bits).
***src**	address of data (in host space) on the controller to be read.

Returns	
MPIMessageOK	
MEIPlatformMessageDEVICE_ERROR	The board type is not an XMP or ZMP controller. 64-bit data is only supported on XMP and ZMP controllers.
MEIPlatformMessageCOPY64_FAILURE	The 64-bit value could not be read successfully.
MPIMessageFATAL_ERROR	The platform type does not exist.

Sample Code

```
...
/* axisPosition.actual is defined as MEIInt64 which is 64 bits */

if (returnValue == MPIMessageOK) {
  returnValue = meiPlatformMemoryGet64(axis->platform,
                                       &axis->axisPosition.actual,
                                       &axis->Axis->ActualPosition);
}

if (returnValue == MPIMessageOK) {
  *actual = (double)axis->axisPosition.actual;
}
...
```

See Also

meiPlatformMemorySet64

meiPlatformMemorySet64

Declaration

```
long meiPlatformMemorySet64(MEIPlatform  platform,
                            void         *dst,
                            const void   *src)
```

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

Description

meiPlatformMemorySet64 sets (writes) 8 bytes of application memory (starting at address ***src***) to platform memory (starting at address ***dst***).

This function should be used to set/write any 64-bit data to the controller. This function can take up to one foreground cycle to complete. This function will take a platform lock blocking all other tasks from accessing the controller.

platform	the handle to the controller's platform object.
dst	address of data (in host space) on the controller to be written.
src	address of host data storage area. Storage MUST be two words (64 bits).

Returns	
MPIMessageOK	
MPIMessageFATAL_ERROR	The platform type does not exist.

Sample Code

```
fMEIInt64 newTargetPos;


returnValue = meiPlatformMemoryGet64(axis->platform,
                                     &newTargetPos,
                                     &axis->Axis->TC.CommandPosition);

if (returnValue == MPIMessageOK) {
    returnValue = meiPlatformMemorySet64(axis->platform,
                                         &axis->Axis->TC.TargetPosition,
                                         &newTargetPos);
}
```

See Also

meiPlatformMemoryGet64

meiPlatformMemoryToFirmware

Declaration

```
long meiPlatformMemoryToFirmware(MEIPlatform   platform,
                                 const  void   *host,
                                 void          **firmware)
```

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

Description

meiPlatformMemoryToFirmware converts a host memory address to a controller memory address.

platform	the handle to the controller's platform object. This should be obtained from meiControlPlatform().
host	the host memory address to be converted.
firmware	the location where the controller's memory address will be written.

See Also

meiPlatformMemoryToHost | meiControlPlatform

meiPlatformMemoryToHost

Declaration

```
long meiPlatformMemoryToHost(MEIPlatform   platform,
                             const  void   *firmware,
                             void          **host)
```

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

Description

meiPlatformMemoryToHost coverts a controller memory address to a host memory address.

platform	the handle to the controller's platform object. This should be obtained from meiControlPlatform(...).
firmware	the controller memory address to be converted.
host	the location where the host memory address will be written.

See Also

meiPlatformMemoryToFirmware | meiControlPlatform

meiPlatformSleep

Declaration

```
void meiPlatformSleep(long   milliseconds)
```

Required Header: stdmei.h

Description

meiPlatformSleep puts the current thread to sleep for the number of ***milliseconds*** specified.

NOTE:
Different platforms have different time slice "quanta" (minimum sleep resolution) for threads. For example, Windows NT, 2000, and XP have a quanta of 10ms. For example, even if meiPlatformSleep (2) is specified, the actual sleep period will essentially be equivalent to meiPlatformSleep(10).

milliseconds	the number of milliseconds for which to put the current thread to sleep

Returns
Converted the numeric text string ascii to a ***long*** and returned it.

See Also

meiPlatformProcessId

Declaration

```
long meiPlatformProcessId(void)
```

Required Header: stdmei.h

Description

meiPlatformProcessId returns the process identification number of the current process.

See Also

meiPlatformTimerCount

Declaration

```
long meiPlatformTimerCount(long  *ticks)
```

Required Header: stdmei.h

Description

meiPlatformTimerCount reads the host CPU's timer value and writes it into the contents of a long pointed to by ***ticks***. The resolution of the platform timer can be determined with meiPlatformTimerFrequency(...).

***ticks**	a pointer to the timer value for the host CPU.

See Also

meiPlatformTimerFrequency

meiPlatformTimerFrequency

Declaration

```
long meiPlatformTimerFrequency(long  *frequency)
```

Required Header: stdmei.h

Description

meiPlatformTimerFrequency reads the host CPU's timer frequency and writes it into the contents of a long pointed to by ***frequency***. The platform timer value can be read with meiPlatformTimerCount(...).

***frequency**	a pointer to the timer frequency for the host CPU.

See Also

meiPlatformTimerCount

meiPlatformTrace

Declaration

```
long meiPlatformTrace(const char *format, ...)
```

Required Header: stdmei.h

Description

meiPlatformTrace displays **printf(...)**-style trace information. An *end-of-line* character will be appended to the output, newline by default.

Library modules call **meiTrace#(...)**, a macro which can be conditionally compiled to call **meiPlatformTrace(...)** (by defining the symbol MEI_TRACE when building the library).

Otherwise, calls to **meiTrace#(...)** are removed by the C preprocessor.

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

See Also

meiPlatformTraceEol

Declaration

```
char meiPlatformTraceEol(char eol)
```

Required Header: stdmei.h

Description

The **meiPlatformTraceEol** function sets the *end-of-line* (***eol***) character that will be used by meiPlatformTrace(...).

milliseconds	the number of milliseconds for which to put the current thread to sleep

Returns
The previous end-of-line character used by meiPlatformTrace(...)

See Also

meiPlatformTrace

meiPlatformTraceFile

Declaration

```
long meiPlatformTraceFile(const char *fileName)
```

Required Header: stdmei.h

Description

meiPlatformTraceFile redirects trace output to ***fileName***, after first closing any previously opened trace file. If no trace file has been explicitly opened, trace output will go to standard output.

Return Values	
MPIMessageOK	

See Also

meiPlatformTrace

meiPlatformTraceFunction

Declaration

```
MEITraceFunction meiPlatformTraceFunction(MEITraceFunction traceFunction)
```

Required Header: stdmei.h

Description

meiPlatformTraceFunction displays the trace output using ***traceFunction***, and replaces the internal function that was called by meiPlatformTrace(...) to display the trace output. Use *PlatformTraceFunction* to enable your application to take control of the display of trace output.

Return Values	
the previous *traceFunction*	if there is a previous function
NULL	if no ***traceFunction*** has been specified (the default trace function is used)

See Also

meiPlatformTrace

meiPlatformWord64Orient

Declaration

```
long meiPlatformWord64Orient(MEIPlatform  platform,
                             MEIInt64     *dst,
                             MEIInt64     *src)
```

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

This function is used to orient the two 32-bit words within a 64-bit entity. This function is needed when reading/writing 64-bit entites from/to the controller where the Get/Set64 functions cannot be used. The function is needed because the XMP is little endian and the ZMP is big endian. The low level MPI read/write functions already deal with byte swapping the bytes within a 32-bit word when reading/writing from/to the controller. The problem is that the generalized low level functions do not know which data is 64-bit data and cannot automatically swap the words. The Get/Set64 function do know that the data being written/read is 64 bits and automatically swaps the 32-bit words when needed.

This function will do nothing to the data for an XMP, but will swap the two 32-bit words for a ZMP. This function should be used when writing general code that could run on either an XMP or a ZMP controller.

platform	
dst	a pointer to 64-bit entity where oriented data will be written. Data should be on the host, not the controller.
src	a pointer to 64-bit entity to be oriented. Data should be on the host, not the controller.

Returns
MPIMessageOK

See Also

MEIPlatformBoardType

Definition

```
typedef enum {
    MEIPlatformBoardTypeUNKNOWN,
    MEIPlatformBoardTypeXMP,
    MEIPlatformBoardTypeZMP,
} MEIPlatformBoardType;
```

Description

MEIPlatformBoardType is the type of motion controller card that is being used.

MEIPlatformBoardTypeUNKNOWN	Board is not being recognized.
MEIPlatformBoardTypeXMP	An XMP Motion Controller board has been recognized.
MEIPlatformBoardTypeZMP	A ZMP Motion Controller board has been recognized.

See Also

MEIPlatformFileMode

Definition

```
typedef enum {
    MEIPlatformFileModeREAD,     /* default */
    MEIPlatformFileModeWRITE,
    MEIPlatformFileModeTEXT,     /* default */
    MEIPlatformFileModeBINARY,
    MEIPlatformFileModeTRUNC,
    MEIPlatformFileModeAPPEND,
} MEIPlatformFileMode;
```

Description

MEIPlatformFileMode is an enumeration that is used as an argument for methods that open files.

MEIPlatformFileModeREAD	Open a file for read access
MEIPlatformFileModeWRITE	Open a file for write access
MEIPlatformFileModeTEXT	Open a file as text format
MEIPlatformFileModeBINARY	Open a file as binary format
MEIPlatformFileModeTRUNC	Truncate existing file or create for reading and writing
MEIPlatformFileModeAPPEND	Open existing file for appending all writes

See Also

meiPlatformFileOpen

MEIPlatformInfo

Definition

```
typedef struct MEIPlatformInfo {
   char   OSInfo[MEIPlatformInfoCHAR_MAX];
   char   CPUInfo[MEIPlatformInfoCHAR_MAX];
   long   CPUMHz;
} MEIPlatformInfo;
```

Change History: Added in the 03.02.00

Description

MEIPlatformInfo is a structure that contains read only data about the host system characteristics. It contains Operating System, CPU type, and CPU clock speed.

OSInfo	a string containing the Operating System information.
CPUInfo	a string containing the CPU information.
CPUMHz	the CPU clock speed in MHz

See Also

MEIPlatformInfoCHAR_MAX

MEIPlatformMessage

Definition

```
typedef enum {

    MEIPlatformMessagePLATFORM_INVALID,
    MEIPlatformMessageDEVICE_INVALID,
    MEIPlatformMessageDEVICE_ERROR,
    MEIPlatformMessageDEVICE_MAP_ERROR,
    MEIPlatformMessageCOPY64_FAILURE,
} MEIPlatformMessage;
```

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

Description

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

MEIPlatformMessagePLATFORM_INVALID

The platform object is not valid. This message code is returned by a platform method if the platform object handle is not valid. Most applications do not use the platform module. The MPI library uses the platform module internally. If an application needs a platform handle, use meiControlPlatform(...). Do NOT create your own platform object with meiPlatformCreate(...).

MEIPlatformMessageDEVICE_INVALID

The platform device driver is not valid. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the platform device handle is not valid. This message code comes from the lower level routines, meiPlatformInit(...) or meiPlatformDeviceClose(...). To correct the problem, make sure the device driver is installed and operational.

MEIPlatformMessageDEVICE_ERROR

The platform device failed. This message code is returned by the platform methods that fail to access a controller via a device driver. It occurs if the specified board type is not a member of the MEIPlatformBoardType enumeration. It also occurs if the device driver fails to read/write controller memory or there is an interrupt handling failure. To correct the problem, verify the platform has support for your controller and the device drive is installed and operational. Check for any resource conflicts (memory range, I/O port range, and interrupts) with other devices.

MEIPlatformMessageDEVICE_MAP_ERROR

The platform device memory mapping failed. This message code is returned by mpiControlInit(...) or mpiControlReset(...) if the controller memory could not be mapped to the operating system's memory space. To correct this problem, verify there are no memory resource conflicts. Also, make sure the host computer and operating system have enough free memory for the controller (XMP-Series requires 8 Mbytes).

MEIPlatformMessageCOPY64_FAILURE

The 64-bit read failed. This message is returned by meiPlatformMemoryGet64(…), mpiAxisCommandPositionGet(…), or mpiAxisActualPositionGet(…), if the 64-bit position data cannot be read atomically. Internally, the MPI uses an algorithm to construct the 64-bit position data via multiple 32-bit reads. If the 64-bit position value is not valid after multiple attempts, this error will be returned. If your application experiences this error message, if possible, use the equivalent 32-bit methods, mpiAxisCommandPositionGet32(…), mpiAxisActualPositionGet32(…), or contact MEI.

See Also

MEIPlatformControlCountMax

Definition

```
#define   MEIPlatformControlCountMax   (8) /* maximum number of controllers
the
                                           driver will currently support */
```

Change History: Added in the 03.04.00

Description

MEIPlatformControlCountMax defines the maximum number of controllers supported on a single system.

See Also

MEIPlatformInfoCHAR_MAX

Definition

```
#define MEIPlatformInfoCHAR_MAX (128)
```

Change History: Added in the 03.02.00

Description

MEIPlatformInfoCHAR_MAX defines the maximum number of characters for the MEIPlatformInfo strings.

See Also

MEIPlatformInfo | MEIPlatformBoardType