Platform Objects

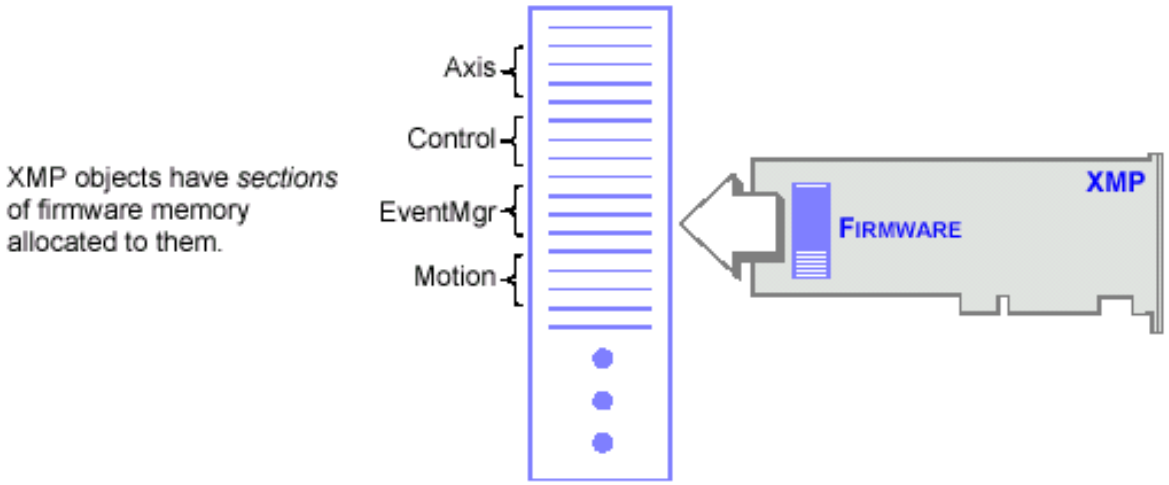
Introduction

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

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

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

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



Methods

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**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**Free**	Free system memory.
meiPlatform**Key**	Return an input character if an input character is available.
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.

Data Types

MEIPlatform**BoardType**

MEIPlatform**FileMode**

MEIPlatform**Message**

meiPlatformAlloc

Declaration

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

Required Header

stdmei.h

Description

PlatformAlloc 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_DECL2 *func)
                          (const char *file, long line));
```

Required Header

stdmei.h

Description

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

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

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

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

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

meiPlatformAtof

Declaration

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

Required Header stdmei.h

Description **PlatformAtof** 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 **PlatformAtol** 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

meiPlatformFileClose

Declaration

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

Required Header

stdmei.h

Description

PlatformFileClose 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

PlatformFileOpen 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

PlatformFileRead 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 **PlatformFileWrite** 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

meiPlatformFree

Declaration

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

Required Header

stdmei.h

Description

PlatformFree 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

meiPlatformKey

Declaration

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

Required Header

stdmei.h

Description

PlatformKey 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

meiPlatformMemoryToFirmware

Declaration

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

Required Header

stdmei.h

Description

PlatformMemoryToFirmware 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,
                             void           *firmware,
                             void           **host)
```

Required Header

stdmei.h

Description

PlatformMemoryToHost 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

PlatformSleep 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 **PlatformProcessId** returns the process identification number of the current process.

See Also

meiPlatformTimerCount

Declaration

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

Required Header

stdmei.h

Description

PlatformTimerCount 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

PlatformTimerFrequency 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

PlatformTrace 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.

Return Values

MPIMessageOK	if *PlatformTrace* successfully executes

See Also

meiPlatformTraceEol

Declaration

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

Required Header

stdmei.h

Description

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

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

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

Return Values	
MPIMessageOK	if *PlatformTraceFile* successfully closes any *previously opened* trace file and redirects trace output to ***fileName***

See Also

meiPlatformTrace

meiPlatformTraceFunction

Declaration

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

Required Header

stdmei.h

Description

PlatformTraceFunction 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

MEIPlatformBoardType

MEIPlatformBoardType

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

Description

PlatformBoardType 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

MEIPlatformFileMode

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

Description

PlatformFileMode 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

MEIPlatformMessage

MEIPlatformMessage

```
typedef enum {

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

Description

MEIPlatformMessagePLATFORM_INVALID

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

MEIPlatformMessageDEVICE_INVALID

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

MEIPlatformMessageDEVICE_ERROR

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

MEIPlatformMessageDEVICE_MAP_ERROR

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

See Also