

# Trace Objects

## Introduction

Use the **Trace** module to selectively produce trace output on a global and/or per-object basis for your application. You can specify the types of trace output when an application is linked, or dynamically (by using a debugger).

**NOTE:** You can also define your own trace function, using [meiPlatformTraceFunction\(...\)](#). For example, you could define your own function to send traces to a circular memory buffer.

The format of the trace output is determined by **printf(...)**-like trace macros located in MPI library source. The trace macros are of the form **meiTrace#(mask, format, arg ...)**, where **format** and the **args** determine the trace output, and where **#** indicates the total number of arguments following the **format** argument (because macros cannot take variable numbers of arguments).

The placement and content of the **meiTrace(...)** macros in the MPI library source is the responsibility of whomever maintains the library. Because trace can be added as desired, it is often useful to leave trace statements in the library source code rather than remove them, as is similarly done with debug **printf(...)** statements. It is also useful to define per-object trace output types so that the volume of trace output is set to a manageable level.

The Trace module interface is declared in the **XMP\include\trace.h** header file. In order for your application to use Trace functions, you must build your application with the **MEI\_TRACE** conditional-compile symbol defined.

**NOTE: Debug and DebugSingle are the only MPI library configurations that will produce trace output.**

To install trace, simply install the DLL for either the Debug or DebugSingle configuration. The Debug and DebugSingle configurations of the MPI library are built with the **MEI\_TRACE** compile-time symbol defined.

By default, trace output is sent to standard error. However, to send trace output to a file, your application can call the [meiTraceFile](#)(char \*fileName) function.

To obtain the current global trace mask, call [meiTraceGet](#).

To modify the global trace mask, call [meiTraceSet](#).

To obtain an object's trace mask, call [meiObjectTraceGet](#) (defined in `stdmei.h`).

To modify an object's trace mask, call [meiObjectTraceSet](#).

**See Also:**[Trace Masks](#)[Global Trace Outputs](#)[Per-Object Trace Outputs](#)

## Methods

### Configuration and Information Methods

<a href="#">meiTraceEol</a>	Set the end-of-line character to be used by Trace
<a href="#">meiTraceFile</a>	Send trace output to a file
<a href="#">meiTraceFunction</a>	sets function used to display a trace buffer
<a href="#">meiTraceGet</a>	Get global trace mask
<a href="#">meiTraceMaskBits</a>	Convert the trace mask into an array of trace bits.
<a href="#">meiTraceMsg</a>	Convert the message identification value into a string.
<a href="#">meiTraceMsgFunction</a>	Set a module's trace message function.
<a href="#">meiTraceSet</a>	Set global trace mask

## Data Types

[MEITrace](#)[MEITraceFunction](#)[MEITraceMask](#)

## Constants

[MEITraceMaskGLOBAL](#)

# meiTraceFile

## Declaration

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

**Required Header:** stdmei.h

## Description

**meiTraceFile** causes trace output to be sent to the file *fileName*. By default, trace output goes to standard output. Note that if *fileName* is Null, trace output still goes to standard output.

### **WARNING!**

Be careful, you can easily run out of disk space. To save disk space, use a circular file type instead of regular file type.

### Return Values

**MPIMessageOK**

if *TraceFile* successfully causes trace output to be sent to the file

## See Also

# meiTraceGet

## Declaration

```
MEITraceMask meiTraceGet(void)
```

**Required Header:** stdmei.h

## Description

**meiTraceGet** returns the current global trace mask for the application.

### Returns

The global trace mask

## See Also

[meiTraceSet](#)

# meiTraceSet

## Declaration

```
MEITraceMask meiTraceSet (MEITraceMask mask)
```

**Required Header:** stdmei.h

## Description

**meiTraceSet** sets the global trace mask to *mask*.

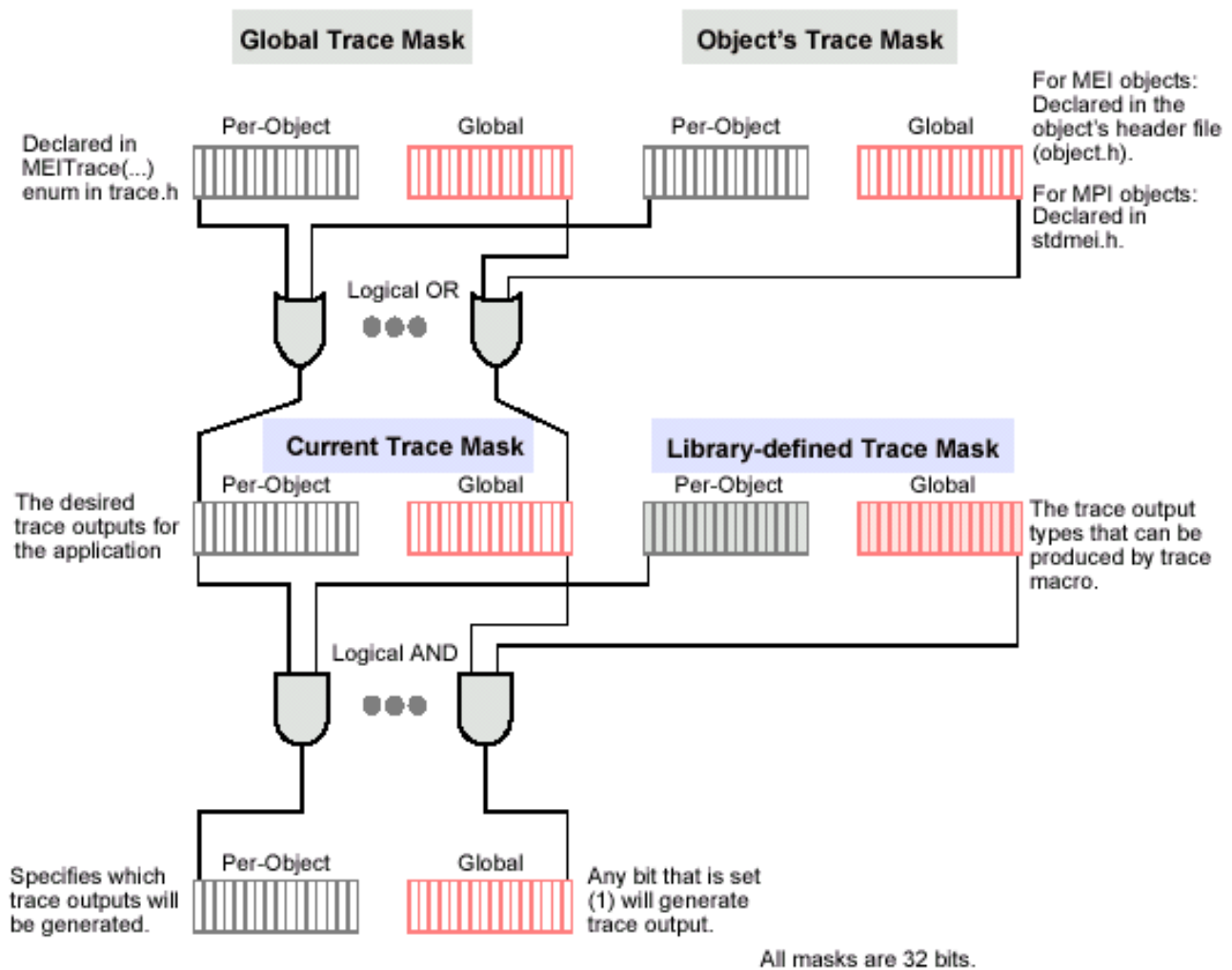
<i>If "traceMask" is</i>	<b>Then</b>
<b>MEITraceALL</b>	all global categories of trace will be enabled
<b>MEITraceNONE</b>	all categories of trace will be disabled

## Returns

The value of the previous *global trace mask*

## See Also

[meiTraceGet](#) | [MEITrace](#)



[Return to Trace Object's page](#)

# meiTraceEol

## Declaration

```
char meiTraceEol(char eol)
```

**Required Header:** stdmei.h

## Description

**meiTraceEol** function simply calls **meiPlatformTraceEol(...)**, which sets the end-of-line character that will be used by **meiPlatformTrace(...)**. By default, **meiPlatformTrace(...)** will append a newline character ('\n') to the messages that it displays. The **meiPlatformTraceEol(...)** function allows your application to set the default end-of-line character.

### Returns

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

## See Also

[meiPlatformTrace](#) | [meiPlatformTraceEol](#)

# meiTraceFunction

## Declaration

```
MEITraceFunction meiTraceFunction(MEITraceFunction traceFunction)
```

**Required Header:** stdmei.h

## Description

**meiTraceFunction** sets the function used to display a trace buffer.

Front end to meiPlatformTraceFunction(). If traceFunction is NULL (default), then trace functions is fprintf(MEIPlatformTraceSTREAM) (default stdout).

### Return Values

<b>handle</b>	to previous Trace function
<b>NULL</b>	otherwise

## See Also



# meiTraceMaskBits

## Declaration

```
long meiTraceMaskBits(long    mask,  
                      long    *bitCount,  
                      long    *bit);
```

**Required Header:** stdmei.h

## Description

**meiTraceMaskBits** converts the trace mask into an array of trace bits and the length of the array.

<b>mask</b>	A bit mask of enumerated trace bits.
<b>*bitCount</b>	A pointer to a long containing the number of trace bits enabled in the mask. This value is also the length of the bit array.
<b>*bit</b>	A pointer to an array of longs containing the enumerated trace bits. Each array member contains one trace bit enumerated value.

## See Also

[MEITrace](#) | [meiTraceGet](#) | [meiTraceSet](#)

# meiTraceMsg

## Declaration

```
const char *meiTraceMsg(long    messageId,  
                        char     *messageText );
```

**Required Header:** stdmei.h

## Description

**meiTraceMsg** converts the message identification value into a string pointed to by *messageText*.

<b>messageId</b>	a message identification value.
<b>*messageText</b>	a pointer to a character string containing the text for the messageId.

## See Also

[meiTraceMsgFunction](#)

# meiTraceMsgFunction

## Declaration

```
long meiTraceMsgFunction(MPIModuleId      moduleId,  
                        MEITraceMsgFunction function);
```

**Required Header:** stdmei.h

## Description

**meiTraceMsgFunction** sets a module's trace message function.

<b>moduleId</b>	an enumerated module identification value
<b>function</b>	a pointer to a trace message function.

## See Also

# MEITrace

## Definition

```
typedef enum {
    MEITraceNONE = 0,
    MEITraceFIRST = 0x0001,
    MEITraceFUNCTION_ENTRY = (int) MEITraceFIRST << 0,
    MEITraceFUNCTION_RETURN = (int) MEITraceFIRST << 1,
    MEITraceMEMORY_ALLOC = (int) MEITraceFIRST << 2,
    MEITraceMEMORY_FREE = (int) MEITraceFIRST << 3,
    MEITraceMEMORY_GET = (int) MEITraceFIRST << 4,
    MEITraceMEMORY_SET = (int) MEITraceFIRST << 5,
    MEITraceVALIDATE = (int) MEITraceFIRST << 6,
    MEITraceLOCK_GIVE = (int) MEITraceFIRST << 7,
    MEITraceLOCK_TAKE = (int) MEITraceFIRST << 8,
    MEITraceEVENT = (int) MEITraceFIRST << 9,

    MEITraceALL = (int) ((MEITraceLAST << 1) - 1)
} MEITrace;
```

## Description

**MEITrace** is an enumeration of generic trace bits that can be used to enable/disable library trace statement output for objects throughout the MPI.

<b>MEITraceFUNCTION_ENTRY</b>	Trace the entry into all methods.
<b>MEITraceFUNCTION_RETURN</b>	Trace the return from all methods.
<b>MEITraceMEMORY_ALLOC</b>	Enables trace statements for all host memory allocations.
<b>MEITraceMEMORY_FREE</b>	Enables trace statements for all host memory de-allocations.
<b>MEITraceMEMORY_GET</b>	Enables trace statements for all controller memory reads.
<b>MEITraceMEMORY_SET</b>	Enables trace statements for all controller memory writes.
<b>MEITraceVALIDATE</b>	Enables trace statements for all function parameter validations.
<b>MEITraceLOCK_GIVE</b>	Enables trace statements for all IPC lock releases.
<b>MEITraceLOCK_TAKE</b>	Enables trace statements for all IPC lock takes.
<b>MEITraceEVENT</b>	Enables trace statements for all MPI Events.

## See Also

[Trace Object](#) | [Trace.exe utility](#)

# MEITraceFunction

## Definition

```
typedef long (*MEITraceFunction) (const char *buffer);
```

## Description

Definition for a trace function interface. **MEITraceFunction** can be used to define a custom trace output routine. MEITraceFunction function must take a pointer to a buffer as a parameter and must return a long.

## See Also

[meiTraceFunction](#)

# MEITraceMask

## Definition

```
typedef unsigned long MEITraceMask;
```

## Description

**MEITraceMask** is a bit mask used to enable/disable library trace statement output.

## See Also

[meiTraceGet](#) | [meiTraceSet](#) | [MEITraceMaskGLOBAL](#)

# MEITraceMsgFunction

## Definition

```
typedef long(*MEITraceMsgFunction)(const char *buffer);
```

## Description

**MEITraceMsgFunction** is the type definition for the callback function used by `meiTrace(...)`. 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

[meiTraceMsgFunction](#) | [MEITrace](#)



# MEITraceMaskGLOBAL

## Definition

```
extern MEITraceMask MEITraceMaskGLOBAL;
```

## Description

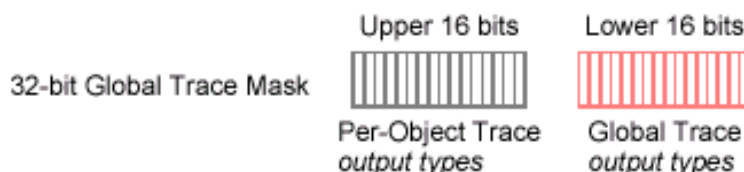
**MEITraceMaskGLOBAL** is a non-object specific MPI Trace mask variable used for library wide Trace bits.

## See Also

[MEITraceMask](#)

# Global Trace Outputs

There is a global 32-bit trace mask: the low 16 bits are the global trace output types, while the upper 16 bits are the per-object trace output types. Each object has a similar trace mask. The upper 16 bits of the global trace mask are not defined, but can be used to set the per-object output types for all objects. To enable all trace output types for all objects, set the global trace mask to all 1s (i.e., -1).



The **MEITrace{...}** enum (declared in trace.h) specifies the global types of trace output, i.e., the types of trace output that can be produced by any object or module. The **MEITrace{...}** enum defines constants that you use together as a bit mask. You specify the desired trace output as a combination (logical OR) of **MEITrace{...}** constants.

There are 16 possible types of global trace output, with 12 global trace outputs defined.

Output Type	Displays
MEITraceFUNCTION_ENTRY	Function name & calling parameters upon entry to function
MEITraceFUNCTION_RETURN	Function name, calling parameters & return value upon exit from function
MEITraceMEMORY_ALLOC	The Address & byte count when memory is dynamically allocated
MEITraceMEMORY_FREE	The Address & byte count when dynamically allocated memory is freed
MEITraceMEMORY_GET	Source address, destination address, byte count when reading XMP firmware memory
MEITraceMEMORY_SET	Source address, destination address, byte count when writing XMP firmware memory
MEITraceVALIDATE	Results of object validation
MEITraceLOCK_GIVE	When a resource lock is released
MEITraceLOCK_TAKE	When a resource lock is waited for & obtained

MEITraceEVENT	When an XMP event is received
MEITraceALL	All global trace outputs (lower 16 bits)
MEIModuleTraceALL	All per-object trace outputs (upper 16 bits)

[Return to Trace Object's page](#)

# Per-Object Trace Outputs

There are 16 possible types of per-object trace output. Each object can declare up to 16 of its own trace output types. MPI modules declare per-object trace output types in `stdmei.h`. MEI modules declare per-object trace output types in the module header file.

Output Type	Displays
MEIMotionTraceSTATUS	Status of the Motion Supervisor
MEINotifyTrcaeTHREAD	When a thread goes to sleep or wakes up
MEISequenceTraceLOAD	When a batch of new commands are sent to the XMP Program Sequencer
MEIConfigTracePROGRESS	Displays "." as it executes (used by config utility)
MEIRecorderTraceRECORD_GET	When the Recorder gets records from the XMP
MEIRecorderTraceSTATUS	The number of data records available in the XMP

**NOTE:** The first 5 output types overlap in the mask.

[Return to Trace Object's page](#)

# Trace Masks

Every MPI object contains an [MEITraceMask](#) and every process contains a single global MEITraceMask. An MEITraceMask consists of bits, where each bit corresponds to a single trace category. A trace category is a specific type of debug information that you want to be displayed by the MPI library. A trace category can be either global (applying to all MPI objects) or object-specific (applying only to a specific MPI object).

Trace Category	Is
global	declared by the MEITrace{...} enum in trace.h.
object-specific (for MPI objects)	declared in stdmei.h.
object-specific (for MEI objects)	declared in the object header file (for MEI objects). Note that the trace mask bits for object-specific trace categories overlap.

An object will produce trace output for a trace category when the logical **OR** of the **global trace mask** and the **object's trace mask** has the bit set that corresponds to the trace category.

If the global trace mask has all of its bits set, then all objects will display trace output for all trace categories.

If an object's trace mask has **all** of its bits set, then that object will display trace output for all trace categories, but a different object of the same type might produce less or no trace output depending on the setting of its trace mask. The setting of the global and object trace masks is under the control of your application.

The trace mask is derived in 2 steps:

1. The **global trace mask** is logically **ORed** with the **object trace mask**. This yields the **current trace mask**, representing the desired trace output types as specified by the application.
2. The **current trace mask** (from step1) is logically **ANDed** with a **library-defined trace mask** (that describes the trace output types for which the trace macro should produce output). If the result of the AND is non-zero, trace output will be produced using the *format* and the *args* [from `meiTrace#(mask, format, arg ...`