Recorder Objects

Introduction

A **Recorder** object provides a mechanism to collect and buffer any data in the controller's memory. After a recorder is configured and started, the controller copies the data from the specified addresses to a local buffer every "N" samples. Later, the host can collect the data by polling or via interrupt-based events.

The controller supports up to 32 data recorders, which can collect data from up to a total of 32 addresses. The buffers can be dynamically allocated. A larger data recorder buffer may be required for higher sample rates, slow host computers, when running via client/server, or when a large number of data fields are being recorded.

A recorder can be started or stopped from the host application or from the controller by configuring a data recorder trigger. When the trigger conditions are met, the controller will automatically start or stop a data recorder. This is very useful for logging relevant variables during the period preceding a fault or error. Normally, the recorder stops collecting data when the buffer is full. It can also be configured to continuously collect data, overwriting the previous data until it is commanded to stop. This is useful for trapping a recent history of controller data.

When using data recorders, make sure to enable enough recorder objects and buffer memory with mpiControlConfigSet(.). Then, configure the recorders with mpiRecorderRecordConfig(.) or mpiRecorderConfigSet(.), and start recording with mpiRecorderStart(.). Data can then be collected with mpiRecorderRecordGet(.).

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

| Buffer Size |

Methods

Create, Delete, Validate Methods

mpiRecorder**Create**	Create Recorder object
mpiRecorder**Delete**	Delete Recorder object

mpiRecorder**Validate**	Validate Recorder object

Configuration and Information Methods

mpiRecorder**ConfigGet**	Get Recorder's configuration
mpiRecorder**ConfigSet**	Set Recorder's configuration
mpiRecorder**RecordConfig**	Configure type of data record that Recorder will capture
mpiRecorder**Status**	Get status of Recorder

Event Methods

mpiRecorder**EventNotifyGet**	Get event mask of events for which host notification has been requested
mpiRecorder**EventNotifySet**	Set event mask of events for which host notification will be requested
mpiRecorder**EventReset**	Reset the events specified in event mask that are generated by Recorder

Action Methods

mpiRecorder**RecordGet**	Get data records from Recorder
mpiRecorder**Start**	Start recording data records using Recorder
mpiRecorder**Stop**	Stop recording data records using Recorder

Memory Methods

mpiRecorder**Memory**	Get address to Recorder's memory
mpiRecorder**MemoryGet**	Copy data from Recorder memory to application memory
mpiRecorder**MemorySet**	Copy data from application memory to Recorder memory

Relational Methods

mpiRecorder**Control**	Return handle of Control object associated with Recorder
mpiRecorder**Number**	

Data Types

MPIRecorder**Config** / MEIRecorder**Config**
MPIRecorder**Message**
MPIRecorder**Record** / MEIRecorder**Record**
MEIRecorder**RecordAxis**
MEIRecorder**RecordFilter**
MPIRecorder**RecordPoint**
MPIRecorder**RecordType** / MEIRecorder**RecordType**
MPIRecorder**Status**
MEIRecorder**Trace**
MEIRecorder**Trigger**

MEIRecorder**TriggerCondition**
MEIRecorder**TriggerIndex**
MEIRecorder**TriggerType**
MEIRecorder**TriggerUser**

Constants

MPIRecorder**ADDRESS_COUNT_MAX**
MEIRecorder**MAX_AXIS_RECORDS**
MEIRecorder**MAX_FILTER_RECORDS**

mpiRecorderCreate

Declaration

```
MPIRecorder mpiRecorderCreate(MPIControl control,
                              long       number);
```

Required Header: stdmpi.h

Description

mpiRecorderCreate creates a Recorder object identified by ***number***, which is associated with a control object. *RecorderCreate* is the equivalent of a C++ constructor.

The recorder number specifies which recorder to create. The valid range for the number parameter is -1 to the controller's ***recordCount*** (MPIControlConfig.recorderCount). Use a recorder number of -1 to specify the recorder number as the next available recorder.

See MPIControlConfig{.} for details. If the recorder is not enabled or is already in use (another process has called mpiRecorderCreate(.) with the same number parameter), mpiRecorderCreate(.) will return an invalid handle causing subsequent mpiRecorderValidate(.) calls to fail.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

control	a handle to a Control object.
number	An index to the controller's data recorder. If (-1) is specified, the next available recorder object handle will be returned. The valid range is from -1 (next available recorder) to the controller's recordCount - 1. When using (-1), make sure to delete the recorder object to free it for other applications. If the recorder object is not freed, it will not be accessible to another application until the controller is reset.

Return Values	
handle	to a Recorder object
MPIHandleVOID	if the Recorder object could not be created

See Also

mpiRecorderDelete | mpiRecorderValidate | MPIControlConfig | mpiControlConfigGet | mpiControlConfigSet

mpiRecorderDelete

Declaration

```
long mpiRecorderDelete(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderDelete deletes a Recorder object and invalidates its handle (***recorder***). *RecorderDelete* is the equivalent of a C++ destructor.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

control	a handle to a Control object.
number	An index to the controller's data recorder. If (-1) is specified, the next available recorder object handle will be returned. The valid range is from -1 (next available recorder) to the controller's recordCount - 1. When using (-1), make sure to delete the recorder object to free it for other applications. If the recorder object is not freed, it will not be accessible to another application until the controller is reset.

Return Values	
MPIMessageOK	if *RecordeDelete* successfully deletes a Recorder object and invalidates its handle

See Also

mpiRecorderCreate | mpiRecorderValidate

mpiRecorderValidate

Declaration

```
long mpiRecorderValidate(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderValidate validates the Recorder object and its handle. RecorderValidate should be called immediately after an object is created.

It is possible to create a recorder object and not delete it, leaving the resources for the recorder occupied, but forgotten about (abandoned). It is most common to run into this situation when using an index of -1 for the recorder. When developing a program and running it in the debugger, it is common for the developer to exit the program without letting the program clean up its recorder resources. To see how to handle this situation programmatically, please see recorderinuse.c.

recorder	a handle to a Recorder object

Return Values	
MPIMessageOK	if Recorder is a handle to a valid object.
MPIRecorderMessageNOT_ENABLED	if the specified recorder number has not been enabled in the controller.
MPIRecorderMessageNO_RECORDERS_AVAIL	if the specified recorder number is (-1) and there are no more recorders available on the controller.

See Also

mpiRecorderCreate | mpiRecorderDelete

mpiRecorderConfigGet

Declaration

```
long mpiRecorderConfigGet(MPIRecorder       recorder,
                          MPIRecorderConfig *config,
                          void              *external)
```

Required Header: stdmpi.h

Description

mpiRecorderConfigGet gets a Recorder's (***recorder***) 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 Recorder's configuration information in ***external*** is in addition to the Recorder's 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).

Remarks

external either points to a structure of type **MEIRecorderConfig{}** or is NULL.

Return Values	
MPIMessageOK	if *RecorderConfigGet* successfully writes the Recorder's configuration to the structure(s)

See Also

MPIRecorderConfig | mpiRecorderConfigSet

mpiRecorderConfigSet

Declaration

```
long mpiRecorderConfigSet(MPIRecorder       recorder,
                          MPIRecorderConfig *config,
                          void              *external)
```

Required Header: stdmpi.h

Description

mpiRecorderConfigSet sets a Recorder's (***recorder***) 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 Recorder's configuration information in ***external*** is in addition to the Recorder's 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).

Remarks

external either points to a structure of type **MEIRecorderConfig{}** or is NULL.

Return Values	
MPIMessageOK	if *RecorderConfigSet* successfully sets the Recorder's configuration using data from the structure(s)

See Also

MEIRecorderConfig | mpiRecorderConfigGet

mpiRecorderRecordConfig

Declaration

```
long mpiRecorderRecordConfig(MPIRecorder           recorder,
                             MPIRecorderRecordType type,
                             long                  count,
                             void                  *handle)
```

Required Header: stdmpi.h

Description

mpiRecorderRecordConfig configures the type (***type***) of record that a Recorder (***recorder***) will capture.

If "type" is	**Then**
MPIRecorderRecordTypePOINT	***count*** data points will be recorded, and ***handle*** points to an array of ***count*** controller addresses
MEIRecorderRecordTypeAXIS	***count*** records of type MPIRecorderRecordAxis{} will be recorded, and ***handle*** points to an array of ***count*** Axis handles
MEIRecorderRecordTypeFILTER	***count*** records of type MPIRecorderRecordFilter{} will be recorded, and ***handle*** points to an array of ***count*** Filter handles

Return Values	
MPIMessageOK	if *RecorderRecordConfig* successfully configures the type of record that the Recorder will capture

See Also

MPIRecorderRecordAxis | MPIRecorderRecordFilter

mpiRecorderStatus

Declaration

```
long mpiRecorderStatus(MPIRecorder       recorder,
                       MPIRecorderStatus *status,
                       void              *external)
```

Required Header: stdmpi.h

Description

mpiRecorderStatus gets the status of the Recorder (***recorder***) and writes it into the structure pointed to by ***status***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

Remarks

external should always be set to NULL.

recorder	a handle to a Recorder object
***status**	a pointer to Recorder's status structure
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	if *RecorderStatus* successfully gets the Recorder's status and writes it into the structure(s)
MPIMessageARG_INVALID	if the *status* pointer is NULL.

See Also

MPIRecorderStatus

mpiRecorderEventNotifyGet

Declaration

```
long mpiRecorderEventNotifyGet(MPIRecorder  recorder,
                               MPIEventMask *eventMask,
                               void         *external)
```

Required Header: stdmpi.h

Description

mpiRecorderEventNotifyGet writes the event mask into the structure pointed to by **eventMask**, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL). (The event mask specifies the event type(s) generated by a Recorder (***recorder***), for which host notification has been requested.)

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

Remarks

external either points to a structure of type MEIEventNotifyData{} or is NULL. An MEIEventNotifyData{} structure is an array of firmware addresses. The contents of these firmware addresses are placed into the MEIEventStatusInfo{} structure (which contains all events generated by this Recorder object).

Return Values	
MPIMessageOK	if *RecorderEventNotifyGet* successfully writes the event mask to the structure(s)

See Also

MEIEventNotifyData | MEIEventStatusInfo | mpiRecorderEventNotifySet

mpiRecorderEventNotifySet

Declaration

```
long mpiRecorderEventNotifySet(MPIRecorder   recorder,
                               MPIEventMask  eventMask,
                               void          *external)
```

Required Header: stdmpi.h

Description

mpiRecorderEventNotifySet requests host notification of the event(s) specified by ***eventMask*** and generated by a Recorder (***recorder***), and also generated by the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The events in ***external*** are in addition to the events in ***recorder***, i.e, the events in ***recorder*** and in ***external*** are not necessarily the same events. Note that ***recorder*** or ***external*** can be NULL (but not both NULL).

Event notification is enabled for the event types specified in ***eventMask***. ***eventMask*** is a bit mask generated by the logical OR of the MPIEventMask bits that are associated with the desired MPIEventType values. Event notification is disabled for event types not specified in eventMask.

The mask of event types (generated by a Recorder object) consists of MEIEventMaskRECORDER_FULL and MEIEventMaskRECORDER_DONE.

To	Use "eventMask"
Enable host notification of all Recorder events	MPIEventMaskALL
Disable host notification of all Recorder events	MPIEventTypeNONE

Remarks

external either points to a structure of type MEIEventNotifyData{} or is NULL. An MEIEventNotifyData{} structure is an array of firmware addresses. The contents of these firmware addresses are placed into the MEIEventStatusInfo{} structure (which contains all events generated by this Recorder object).

Return Values	
MPIMessageOK	if *RecorderEventNotifySet* successfully requests host notification of the event(s) as specified by the structure(s)

See Also

MEIEventMaskRECORDER | MEIEventNotifyData | MEIEventStatusInfo
mpiRecorderEventNotifyGet

mpiRecorderEventReset

Declaration

```
long mpiRecorderEventReset(MPIRecorder  recorder,
                           MPIEventMask eventMask)
```

Required Header: stdmpi.h

Description

mpiRecorderEventReset resets the event(s) specified in ***eventMask*** and generated by a Recorder (***recorder***). Your application should call *RecorderEventReset* only after one or more latchable events have occurred.

Return Values	
MPIMessageOK	if *RecorderEventReset* successfully resets the event(s) that are specified in ***eventMask*** and generated by a Recorder

See Also

mpiRecorderRecordGet

Declaration

```
long mpiRecorderRecordGet(MPIRecorder       recorder,
                          long              countMax,
                          MPIRecorderRecord *record,
                          long              *count)
```

Required Header: stdmpi.h

Description

mpiRecorderRecordGet obtains a Recorder's (***recorder***) data records. The record type must have been configured previously, by a prior call to mpiRecorderRecordConfig(...).

RecorderRecordGet gets a maximum of ***countMax*** records and writes them into the location pointed to by ***record*** (the location must be large enough to hold them). RecorderRecordGet also writes the actual number of records that were obtained to the location pointed to by ***count***.

If the recorder data buffer is full and recording is enabled, recording will be temporarily disabled while either all or ***countMax*** records are obtained, whichever is less. Any records not obtained will be lost.

Return Values	
MPIMessageOK	if *RecorderRecordGet* successfully gets the data records

See Also

mpiRecorderRecordConfig

mpiRecorderStart

Declaration

```
long mpiRecorderStart(MPIRecorder  recorder,
                      long         count); /* -1 => continuous,
                                              >0 => # of records */
```

Required Header: stdmpi.h

Description

mpiRecorderStart commands the controller to begin recording data records. Before starting a recorder, it must be configured with mpiRecorderRecordConfig(.) or mpiRecordConfigGet/Set(.).

recorder	a handle to a Recorder object
count	The number of data records to record. If (-1) is specified, the data recorder will continuously record until the buffer is full. If the host is retrieving data from the buffer faster than the controller can fill the buffer, the controller will continuously copy data to the buffer. The valid range is from -1 (continuous recording) to the maximum number of records available in the data recorder buffer.

Return Values	
MPIMessageOK	if the data recorder successfully begins to record data.
MPIRecorderMessageSTARTED	if the data recorder is already running.

See Also

mpiRecorderRecordConfig | mpiRecorderStop | mpiRecorderConfigGet | mpiRecorderConfigSet | mpiControlConfigGet | mpiControlConfigSet

mpiRecorderStop

Declaration

```
long mpiRecorderStop(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderStop instructs a Recorder (***recorder***) to stop recording data records.

recorder	a handle to a Recorder object

Return Values	
MPIMessageOK	if *RecorderStop* successfully stops recording data records
MPIRecorderMessageSTOPPED	This means the the recorder was already stopped when mpiRecorderStop was called. This is a warning, not an error. This can be ignored if the user does not have some reason for why the recorder must be running at this point.

Sample Code

```
/*
    Look for the warning code when the recorder is already stopped.
    This is usually not considered a bad thing (error).
*/
returnValue = mpiRecorderStop(recorder);
if(returnValue == MPIRecorderMessageSTOPPED)
{
returnValue = MPIMessageOK;
}
msgCHECK(returnValue);
```

See Also

mpiRecorderStart

mpiRecorderMemory

Declaration

```
long mpiRecorderMemory(MPIRecorder recorder,
                       void        **memory)
```

Required Header: stdmpi.h

Description

mpiRecorderMemory writes an address to the contents of ***memory***. An address can be used to access a Recorder's (***recorder***) memory. An address calculated from it can be passed as the ***src*** argument to mpiRecorderMemoryGet(...) and as the ***dst*** argument to mpiRecorderMemorySet(...).

Return Values	
MPIMessageOK	if *RecorderMemory* successfully writes the Recorder's memory address to the contents of ***memory***

See Also

mpiRecorderMemoryGet | mpiRecorderMemorySet

mpiRecorderMemoryGet

Declaration

```
long mpiRecorderMemoryGet(MPIRecorder recorder,
                          void        *dst,
                          void        *src,
                          long        count)
```

Required Header: stdmpi.h

Description

mpiRecorderMemoryGet copies ***count*** bytes of a Recorder's (***recorder***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *RecorderMemoryGet* successfully copies data from Recorder memory to application memory

See Also

mpiRecorderMemory | mpiRecorderMemorySet

mpiRecorderMemorySet

Declaration

```
long mpiRecorderMemorySet(MPIRecorder recorder,
                          void        *dst,
                          void        *src,
                          long        count)
```

Required Header: stdmpi.h

Description

mpiRecorderMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Recorder's (***recorder***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *RecorderMemorySet* successfully copies data from application memory to Recorder memory

See Also

mpiRecorderMemory | mpiRecorderMemoryGet

mpiRecorderControl

Declaration

```
MPIControl mpiRecorderControl(MPIRecorder recorder)
```

Required Header: stdmpi.h

Description

mpiRecorderControl returns a handle to the motion controller (Control object) that a Recorder (***recorder***) is associated with.

Return Values	
handle	to a Control object that a Recorder is associated with
MPIHandleVOID	if the Recorder object is invalid

See Also

mpiRecorderNumber

Declaration

```
long mpiRecorderNumber(MPIRecorder        recorder,
                       long               *number);
```

Required Header: stdmpi.h

Description

mpiRecorderNumber reads the index of a Recorder object and writes it into the contents of a long pointed to by ***number***. Each data recorder associated with a controller is indexed by a number (0, 1, 2, etc.).

recorder	a handle to a Recorder object.
***number**	a pointer to the index of a Recorder object.

Return Values	
MPIMessageOK	if *RecorderNumber* successfully gets the index of a data recorder and writes it into the contents of a long pointed to by *number*.
MPIMessageARG_INVALID	See description.
MPIMessageHANDLE_INVALID	See description.

See Also

mpiRecorderCreate

MPIRecorderConfig / MEIRecorderConfig

Definition: MPIRecorderConfig

```
typedef struct MPIRecorderConfig {
    long  period;     /* collect 1 record every `period` milliseconds */
    long  highCount;  /* >0 => record count to trigger high buffer */
    long  bufferWrap; /* TRUE/FALSE */

    long  addressCount; /* number of data point addresses in address[] */
    void  *address[MPIRecorderADDRESS_COUNT_MAX];
} MPIRecorderConfig;
```

Description

MPIRecorderConfig structure specifies the configurations for a data recorder. It configures the sampling period, the buffer high event level, whether the buffering should wrap around, and a list of controller addresses to record.

period	The number of controller samples between successive data recorder acquisitions. A value of zero or one means the data recorder will acquire data every sample. A value of 2 means every other sample, 3 means every 3rd sample, etc. The valid range is 0 to 32767.
highCount	The number of buffered records until a MPIEventTypeRECORDER_HIGH status/event is generated. The valid range is 1 to the recorder buffer size configured by mpiControlConfigSet(.).
bufferWrap	Data recorder buffer rollover. A value of TRUE enables the buffer rollover, FALSE (default) disables the buffer rollover. When the bufferWrap is disabled, the controller will stop collecting data when the buffer is full. When bufferWrap is enabled, the controller will continuously collect data after the buffer is full, overwriting any previously collected data. The bufferWrap should be enabled if your application only wants to retrieve the last buffer of data after the data recorder is stopped. Most applications should set the bufferWrap to FALSE.
addressCount	The number of controller addresses in the address array.
***address**	An array of controller memory addresses to be recorded.

Definition: MEIRecorderConfig

```
typedef struct  MEIRecorderConfig {
    MEIRecorderTrigger  trigger[MEIRecorderTriggerIndexLAST];
} MEIRecorderConfig;
```

Description

MEIRecorderConfig specifies the configurations for the controller's data recorder triggers.

A data recorder can be started or stopped from the host application with mpiRecorderStart/Stop(.) or from the controller by configuring a data recorder trigger. When the trigger conditions are met, the controller will automatically start or stop a data recorder.

trigger	An array of data recorder trigger configuration structures.

See Also

mpiRecorderConfigGet | mpiRecorderConfigSet | mpiRecorderStart | mpiRecorderStop

MPIRecorderMessage

Definition

```
typedef enum {

    MPIRecorderMessageRECORDER_INVALID,
    MPIRecorderMessageSTARTED,
    MPIRecorderMessageSTOPPED,
    MPIRecorderMessageNOT_CONFIGURED,
    MPIRecorderMessageNO_RECORDERS_AVAIL,
    MPIRecorderMessageNOT_ENABLED,
    MPIRecorderMessageRUNNING,
} MPIRecorderMessage;
```

Description

MPIRecorderMessage lists the error messages returned by the Recorder module.

MPIRecorderMessageRECORDER_INVALID

The recorder object is not valid. This message code is returned by a recorder method if the recorder object handle is not valid. This problem can be caused by a failed mpiRecorderCreate(.). To prevent this problem, check your recorder objects after creation by using mpiRecorderValidate(.).

MPIRecorderMessageSTARTED

The data recorder is already running. This message code is returned by mpiRecorderStart(.) if the data recorder has already been started. If this is a problem, call mpiRecorderStop(.) to stop the data recorder or wait for the recorder to collect the number of specified records and stop.

MPIRecorderMessageSTOPPED

The data recorder is not running. This message code is returned by mpiRecorderStop(.) if the data recorder has already been stopped. If this is a problem, call mpiRecorderStart(.) to start the data recorder.

MPIRecorderMessageNOT_CONFIGURED

The data recorder has not been configured. This message code is returned by mpiRecorderRecordGet(.) if the data address count has not been configured. To correct this problem, configure the data recorder with mpiRecorderConfigSet(.).

MPIRecorderMessageNO_RECORDERS_AVAIL

Returned when a recorder number of -1 is specified and all enabled recorders have been previously reserved by mpiRecorderCreate(...) method calls. Reserved recorders are released by calling mpiRecorderDelete(...), however, it is possible for a fatal error to occur in your application in which case mpiRecorderDelete(...) may not be called. To override a reserved recorder number, explicitly specify the recorder number (i.e. a number other than -1) when calling mpiRecorderCreate(...).

MPIRecorderMessageNOT_ENABLED

An attempt was made to create a recorder that is not enabled on the controller. Recorder objects can be enabled on the controller by calling mpiControlConfigSet(...).

MPIRecorderMessageRUNNING

An attempt was made to call mpiRecorderConfigSet(...) while the recorder was running.

See Also

mpiRecorderCreate | mpiRecorderValidate

MPIRecorderRecord / MEIRecorderRecord

Definition: MPIRecorderRecord

```
typedef union {
   MPIRecorderRecordPoint  point[MPIRecorderADDRESS_COUNT_MAX];
} MPIRecorderRecord;
```

Description

point	An array of recorded values corresponding to the XMP addresses stored in MPIRecorderConfig.address[].

Definition: MEIRecorderRecord

```
typedef union {
    MEIRecorderRecordAxis     axis[MEIXmpMAX_Axes];
    MEIRecorderRecordFilter   filter[MEIXmpMAX_Filters];
    MPIRecorderRecord         dummy;  /* ensure proper sizing */
} MEIRecorderRecord;
```

Description

axis	An array of MEIRecorderRecordAxis records.
filter	An array of MEIRecorderRecordFilter records.
dummy	A dummy structure that ensures that MEIRecorderRecord has the proper size.

See Also

MPIRecorderConfig

MEIRecorderRecordAxis

Definition

```
typedef struct MEIRecorderRecordAxis {
    long    sample;     /* sample number */
    long    command;    /* command position */
    long    actual;        /* actual position */
    float   dac;               /* voltage */
} MEIRecorderRecordAxis;
```

Description

sample	The XMP sample number in which the following values were recorded.
command	The command position of the axis.
actual	The actual position of the axis.
dac	The output of the primary DAC of the motor associated with the axis.

See Also

MEIRecorderRecordFilter

Definition

```
typedef struct MEIRecorderRecordFilter {
    long   sample;   /* sample number */
    long   command;  /* command position */
    long   actual;   /* actual position */
    float  dac;      /* voltage */
} MEIRecorderRecordFilter;
```

Description

sample	The XMP sample number in which the following values were recorded
command	The command position the filter uses to calculate the filter output.
actual	The actual position (of an axis) the filter uses to calculate the filter output.
dac	The output of the filter that gets sent to a motor's primary DAC.

See Also

MPIRecorderRecordPoint

Definition

```
typedef long MPIRecorderRecordPoint;
```

Description

MPIRecorderRecordPoint	represents one recorder record. This will correspond to the value of one XMP address.

See Also

MPIRecorderType / MEIRecorderType

Definition: MPIRecorderType

```
typedef enum {
    MPIRecorderRecordTypeINVALID,
    MPIRecorderRecordTypePOINT,
} MPIRecorderRecordType;
```

Description

MPIRecorderRecordTypeINVALID	an invalid record type.
MPIRecorderRecordTypePOINT	specifies to the data recorder that MPIRecorderRecordPoint records (copies of controller memory locations) are being recorded.

Definition: MEIRecorderType

```
typedef enum {
    MEIRecorderRecordTypeAXIS,
    MEIRecorderRecordTypeFILTER,
} MEIRecorderRecordType;
```

Description

Predefined types for setting up the type of data an MPIRecorder object will record. This is used by the mpiRecorderRecordConfig() method.

MEIRecorderRecordTypeAXIS	specifies to the data recorder that MEIRecorderRecordAxis records are being recorded.
MEIRecorderRecordTypeFILTER	specifies to the data recorder that MEIRecorderRecordFilter records are being recorded.

See Also

MPIRecorder | MEIRecorderRecordAxis | MEIRecorderRecordFilter | mpiRecorderRecordConfig

MPIRecorderStatus

Definition

```
typedef struct MPIRecorderStatus {
    long    enabled;
    long    full;
    long    recordCount;
    long    recordCountMax;
} MPIRecorderStatus;
```

Description

enabled	If the recorder is enabled (recording) then enabled will equal a non-zero value (-1), otherwise enabled will equal 0.
full	If the recorder is full (the number of stored records >= MPIRecorderConfig.fullCount) then full will equal TRUE, otherwise full will equal FALSE.
recordCount	The number of stored records in the recorder.
recordCountMax	The maximum number of records the recorder can store.

See Also

mpiRecorderStatus

MEIRecorderTrace

Definition

```
typedef enum {

    MEIRecorderTraceRECORD_GET,
    MEIRecorderTraceSTATUS,
    MEIRecorderTraceOVERFLOW,
} MEIRecorderTrace;
```

Description

MEIRecorderTraceRECORD_GET	will display trace information when the data recorder retrieves records.
MEIRecorderTraceSTATUS	will display trace information when the MPI retrieves the data recorder status.
MEIRecorderTraceOVERFLOW	will display trace information when the data recorder overflows.

See Also

MEIRecorderTrigger

Definition

```
typedef struct MEIRecorderTrigger {
    MEIRecorderTriggerType     type;
    union {
        MEIRecorderTriggerUser user;
        } attributes;
} MEIRecorderTrigger;
```

Description

MEIRecorderTrigger specifies the configurations for a data recorder trigger.

type	The data recorder trigger type. See the MEIRecorderTriggerType enumeration.
user	The configurations for a user specified trigger type. See MEIRecorderTriggerUser.

See Also

MEIRecorderTrigger | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerCondtion

Definition

```
typedef enum MEIRecorderTriggerCondition {
    MEIRecorderTriggerConditionMATCH,
    MEIRecorderTriggerConditionCHANGE,
} MEIRecorderTriggerCondition;
```

Description

MEIRecorderTriggerCondtion is an enumeration of a data recorder's trigger conditions. The mask and pattern fields referred to are from the MEIRecorderTriggerUser structure.

MEIRecorderTriggerTriggerMATCH	Triggers when the value at the specified address ANDed with the **mask** is equal to the specified **pattern**.
MEIRecorderTriggerTriggerCHANGE	Triggers when the value at the specified address ANDed with the **mask** changes. The **pattern** field is only used to set the initial bit pattern used to determine if a change occurs.

See Also

MEIRecorderTriggerUser | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerIndex

Definition

```
typedef enum MEIRecorderTriggerIndex {
    MEIRecorderTriggerIndexSTART,
    MEIRecorderTriggerIndexSTOP,
} MEIRecorderTriggerIndex;
```

Description

MEIRecorderTriggerIndex is an enumeration of indices to a data recorder's trigger logic.

MEIRecorderTriggerIndexSTART	Index to a data recorder's start trigger.
MEIRecorderTriggerIndexSTOP	Index to a data recorder's stop trigger.

See Also

MEIRecorderConfig | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerType

Definition

```
typedef enum MEIRecorderTriggerType {
    MEIRecorderTriggerTypeDISABLED,
    MEIRecorderTriggerTypeUSER,
} MEIRecorderTriggerType;
```

Description

MEIRecorderTriggerType is an enumeration of a data recorder's trigger logic types.

MEIRecorderTriggerTypeDISABLED	The data recorder trigger is not enabled.
MEIRecorderTriggerTypeUSER	The data recorder trigger is user configurable. See the MEIRecorderTriggerUser{.} structure for details.

See Also

MEIRecorderTrigger | MEIRecorderTriggerUser | mpiRecorderConfigGet | mpiRecorderConfigSet

MEIRecorderTriggerUser

Definition

```
typedef struct  MEIRecorderTriggerUser {
    MEIRecorderTriggerCondition    condition;
    long                           *addr;
    unsigned long                  mask;
    unsigned long                  pattern;
    unsigned long                  count;
} MEIRecorderTriggerUser;
```

Description

MEIRecorderTriggerUser specifies the configurations for a user specified data recorder trigger.

condition	The logic that determines how to evaluate the addr, mask, and pattern. See the MEIRecorderTriggerCondition enumeration.
***addr**	A pointer to a controller address.
mask	A bit mask ANDed with the value at the controller address.
pattern	A bit pattern compared to the masked value at the controller address.
count	The number of records to collect when the recorder is triggered. This is valid for both start and stop triggers. The valid range is 0 to the recorder buffer size configured by mpiControlConfigSet(.). When used for the start trigger, the valid values range from -1 (continuous recording) to the maximum number of records available in the data recorder buffer. When used for the stop trigger, *count* records will be collected after the trigger has triggered.

See Also

MEIRecorderTrigger | mpiRecorderConfigGet | mpiRecorderConfigSet

MPIRecorderADDRESS_COUNT_MAX

Definition

```
#define  MPIRecorderADDRESS_COUNT_MAX  (32)
```

Description

MPIRecorderADDRESS_COUNT_MAX defines the maximum number of addresses the Recorder object supports.

See Also

MPIRecorderConfig

MEIRecorderMAX_AXIS_RECORDS

Definition

```
#define  MEIRecorderMAX_AXIS_RECORDS  (8)
```

Description

MEIRecorderMAX_AXIS_RECORDS defines the maximum number of MEIRecorderRecordAxis records that can be recorded by a single recorder at any one time.

See Also

MEIRecorderRecordAxis | mpiRecorderRecordConfig

MEIRecorderMAX_FILTER_RECORDS

Definition

```
#define  MEIRecorderMAX_FILTER_RECORDS   (8)
```

Description

MEIRecorderMAX_FILTER_RECORDS defines the maximum number of MEIRecorderRecordFilter records that can be recorded by a single recorder at any one time.

See Also

MEIRecorderRecordFilter | mpiRecorderRecordConfig

Recorder Buffer Size

The Data Recorder buffer size can be dynamically allocated. The MPIControlConfig{...} structure has a new element, called recordCount. This element allows the application to change the size of the recorder object's data buffer using the mpiControlConfigGet/Set(...) methods. The Record buffer size (the default is 3064 records) is defined within the MEIXmpDefaultEnabled_Records structure (*xmp.h*). Each record is the size of one memory word. Using a larger data buffer size can improve the performance of MotionScope running on a slow host or running in Client/Server mode over a congested network.

A new method, meiControlExtMemAvail(...), has been added which will return the size of external memory available for allocation. This value can be added to the current recordCount to expand the record buffer to the maximum possible size.

For more information, see the Special Note on *Dynamic Allocation of External Memory Buffer*s.

Return to Recorder Object's page