Recorder Objects

Introduction

The **Recorder** object provides a mechanism to collect and buffer data from the controller. Only one Recorder object per controller is supported. After the Recorder is configured and started, the controller copies the data from the specified fields to a local buffer every "*N*" samples. The host can collect the data from the Recorder object using polling or interrupt-based techniques.

| 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**FlashConfigGet**	Get Recorder's flash configuration
mpiRecorder**FlashConfigSet**	Set Recorder's flash 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

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**

Constants

MPIRecorder**ADDRESS_COUNT_MAX**

mpiRecorderCreate

Declaration `const MPIRecorder mpiRecorderCreate(MPIControl control)`

Required Header stdmpi.h

Description **RecorderCreate** creates a Recorder object associated with a Control object (***control***). *RecorderCreate* is the equivalent of a C++ constructor.

Return Values

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

See Also mpiRecorderDelete | mpiRecorderValidate

mpiRecorderDelete

Declaration `long mpiRecorderDelete(MPIRecorder recorder)`

Required Header stdmpi.h

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

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 **RecorderValidate** validates the Recorder object and its handle (***recorder***).

Return Values	
MPIMessageOK	if Recorder is a handle to a valid object.

See Also mpiRecorderCreate | mpiRecorderDelete

mpiRecorderConfigGet

Declaration

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

Required Header

stdmpi.h

Description

RecorderConfigGet 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).

XMP Only ***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

MEIRecorderConfig | mpiRecorderConfigSet

mpiRecorderConfigSet

Declaration

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

Required Header

stdmpi.h

Description

RecorderConfigSet 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).

XMP Only	***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

mpiRecorderFlashConfigGet

Declaration

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

Required Header

stdmpi.h

Description

RecorderFlashConfigGet gets the flash configuration (***flash***) for a Recorder (***recorder***) and writes it into the structure pointed to by ***config***, and also writes it in the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Recorder's flash configuration information in ***external*** is *in addition* to the Recorder's flash configuration information in ***config***, i.e, the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

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

Return Values

MPIMessageOK	if *RecorderFlashConfigGet* successfully writes the Recorder's flash configuration to the structure(s) ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

MEIRecorderConfig | MEIFlash | mpiRecorderFlashConfigSet

mpiRecorderFlashConfigSet

Declaration

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

Required Header

stdmpi.h

Description

RecorderFlashConfigSet sets the flash configuration (***flash***) for a Recorder (***recorder***) 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 flash configuration information in ***external*** is in addition to the Recorder's flash configuration information in config, i.e., the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

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

Return Values

MPIMessageOK	if *RecorderFlashConfigSet* successfully sets the Recorder's flash configuration using data from the structure(s) ***flash*** is either an MEIFlash handle or MPIHandleVOID. If ***flash*** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

MEIRecorderConfig | MEIFlash | mpiRecorderFlashConfigGet

mpiRecorderRecordConfig

Declaration

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

Required Header

stdmpi.h

Description

RecorderRecordConfig 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

RecorderStatus 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).

The Recorder's status information in ***external*** is in addition to the Recorder's status information in ***status***, i.e., the status information in ***status*** and in ***external*** is not the same information. Note that ***status*** or ***external*** can be NULL (but not both NULL).

XMP Only	***external*** either points to a structure of type MPIRecorderStatus{} or is NULL.

Return Values

MPIMessageOK	if *RecorderStatus* successfully gets the Recorder's status and writes it into the structure(s)

See Also

MPIRecorderStatus

mpiRecorderEventNotifyGet

Declaration

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

Required Header

stdmpi.h

Description

RecorderEventNotifyGet 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).

XMP Only

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

RecorderEventNotifySet 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

XMP Only	***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

RecorderEventReset 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

RecorderRecordGet 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,
                      long         period,
                      long         fullCount)
```

Required Header

stdmpi.h

Description

RecorderStart instructs a Recorder (***recorder***) to begin recording data records. The record type must have been configured previously, by a prior call to mpiRecorderRecordConfig(...).

An event of type MPIEventRECORDER_DONE will be generated when the Recorder has finished collecting data records.

count (for ***count*** > 0)	specifies the total number of data records to record
if ***count*** = -1	then recording is continuous, and uncollected records may be overwritten
period if ***period*** = 0	frequency of recording in milliseconds then data will be recorded as quickly as possible (on every sample)
if ***fullCount*** > 0	then an event of type MPIEventRECORDER_COUNT will be generated when the Recorder's data buffer contains ***fullCount*** data records

Return Values

MPIMessageOK	if *RecorderStart* successfully instructs a Recorder to begin recording data records

See Also

mpiRecorderRecordConfig | mpiRecorderStop

mpiRecorderStop

Declaration `long mpiRecorderStop(MPIRecorder recorder)`

Required Header stdmpi.h

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

Return Values	
MPIMessageOK	if *RecorderStop* successfully stops recording data records

See Also mpiRecorderStart

mpiRecorderMemory

Declaration

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

Required Header stdmpi.h

Description **RecorderMemory** 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

RecorderMemoryGet 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 **RecorderMemorySet** 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 `const MPIControl mpiRecorderControl(MPIRecorder recorder)`

Required Header stdmpi.h

Description **RecorderControl** 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

MPIRecorderConfig / MEIRecorderConfig

MPIRecorderConfig

```
typedef struct MPIRecorderConfig {
    long        period;         /* collect 1 record every `period` milliseconds */
    long        fullCount;      /* >0 => record count to trigger full buffer */

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

Description

period	The number of milliseconds between successive recordings. A value of zero means the recorder will record data every sample.
fullCount	If fullCount>0, then the recorder will generate a "recorder full" event when this fullCount number or records are in the recorder buffer.
addressCount	The number of XMP memory addresses in address[].
***address[]**	An array of XMP memory addresses the recorder will record.

MEIRecorderConfig

```
typedef MPIEmpty        MEIRecorderConfig;
```

Description

RecorderConfig is currently not supported and is reserved for future use.

See Also

mpiRecorderConfigGet | mpiRecorderConfigSet

MPIRecorderMessage

MPIRecorderMessage

```
typedef enum {

    MPIRecorderMessageRECORDER_INVALID,
    MPIRecorderMessageSTARTED,
    MPIRecorderMessageSTOPPED,
    MPIRecorderMessageNOT_CONFIGURED,
} MPIRecorderMessage;
```

Description

MPIRecorderMessageRECORDER_INVALID	
Meaning	The MPIRecorder handle passed to an MPIRecorder method is invalid.
Possible Causes	Either the handle was never initialized or the MPIRecorderCreate method failed.
Recommendations	Use MPIRecorderValidate after MPIRecorderCreate to see if the returned handle is valid.

MPIRecorderMessageSTARTED	
Meaning	
Possible Causes	An MPIRecorder method that assumes that recorder has not been started has been called while the recorder was recording data.
Recommendations	

MPIRecorderMessageSTOPPED	
Meaning	
Possible Causes	An MPIRecorder method that assumes that recorder has been started has been called while the recorder was not recording data.
Recommendations	

MPIRecorderMessageNOT_CONFIGURED	
Meaning	
Possible Causes	An MPIRecorder method has been called that assumes that the recorder has been configured. mpiRecorderStart is the most common method to return this error code.
Recommendations	

Sample Code

```
MPIControl     control;
MPIRecorder    recorder;
long           returnValue;
…

recorder =
      mpRecorderCreate(control);
returnValue =
      mpiRecorderValidate(recorder);
```

See Also

mpiRecorderCreate | mpiRecorderValidate

MPIRecorderRecord / MEIRecorderRecord

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[].

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

MEIRecorderRecordAxis

MEIRecorderRecordAxis

```
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

MEIRecorderRecordFilter

```
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

MPIRecorderRecordPoint

```
typedef long MPIRecorderRecordPoint;
```

Description

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

See Also

MPIRecorderRecordType / MEIRecorderRecordType

MPIRecorderRecordType

```
typedef enum {
    MPIRecorderRecordTypeINVALID,
    MPIRecorderRecordTypePOINT,

} MPIRecorderRecordType;
```

Description

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

MEIRecorderRecordType

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

Description

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

Remarks

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

See Also

MPIRecorder | MEIRecorderRecordAxis | MEIRecorderRecordFilter | mpiRecorderRecordConfig

MPIRecorderStatus

MPIRecorderStatus

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

Description

enabled	If the recorder is enabled (recording) then enabled will equal TRUE, otherwise enabled will equal FALSE.
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

MEIRecorderRecordTrace

MEIRecorderRecordTrace

```
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

MPIRecorderADDRESS_COUNT_MAX

MPIRecorderADDRESS_COUNT_MAX

```
#define MPIRecorderADDRESS_COUNT_MAX    (128)
```

Description

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

See Also

MPIRecorderConfig

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