Event Objects

Introduction

An **Event** object contains information about an asynchronous event. Typically, events are generated by the controller, but in some special cases it is possible to generate events from the host computer.

The Event object is retrieved through the EventMgr, via the Notify object. The Event object contains data about the type of event, its source, and other information. The user Event fields can be configured to collect data at the time when the event occurs in the controller.

Methods

Configuration and Information Methods

mpiEvent**StatusGet**	Get Event status
mpiEvent**StatusSet**	Set Event status

Data Types

MPIEvent**Message**

MEIEvent**NotifyData**

MPIEvent**Status**

MEIEvent**StatusInfo**

MPIEvent**Type** / MEIEvent**Type**

Constants

MPIEvent**StatusINFO_COUNT_MAX**	defines the size of the MPIEventStatus.info[] array.

mpiEventStatusGet

Declaration

```
long mpiEventStatusGet(MPIEvent       event,
                       MPIEventStatus *status)
```

Required Header

event.h

Description

EventStatusGet gets the status of an Event object (***event***) and writes it into the structure pointed to by ***status***. Event status includes the event type, type-specific codes and the event source.

Return Values

MPIMessageOK	if *EventStatusGet* successfully gets the status of an Event object and writes it into the structure

See Also

mpiEventStatusSet | meiEventStatusInfo

mpiEventStatusSet

Declaration

```
long mpiEventStatusSet(MPIEvent       event,
                       MPIEventStatus *status)
```

Required Header

event.h

Description

EventStatusSet sets (writes) the status of ***event*** using data from the structure pointed to by ***status***. Event status includes the event type, type-specific codes and the event source.

Return Values

MPIMessageOK	if *EventStatusSet* successfully sets (writes) the status of event using data from the structure

See Also

mpiEventStatusGet | meiEventStatusInfo

MPIEventMessage

MPIEventMessage

```
typedef enum {
    MPIEventMessageEVENT_INVALID,
} MPIEventMessage;
```

Description

MPIEventMessageEVENT_INVALID

The event type is not valid. This message code is returned by mpiEventStatusSet(…) if the event type is not a member of the MPIEventType or MEIEventType enumerations.

See Also

MEIEventNotifyData

MPIEventNotifyData

```
typedef struct MEIEventNotifyData {
    void        *address[MEIXmpSignalUserData];
} MEIEventNotifyData;
```

Description

The ***address*** of an **EventNotifyData** structure is passed as the third (void *external) argument to mpi'Object'EventNotify[GS]et(). The address array contains host-based XMP addresses, the contents of which are returned in MEIEventStatusInfo{}.data.

See Also

MEIEventStatusInfo

MPIEventStatus

MPIEventStatus

```
typedef struct MPIEventStatus {
    MPIEventType    type;
    void            *source;

    long            info[MPIEventStatusINFO_COUNT_MAX];
} MPIEventStatus;
```

Description

EventStatus holds information about a particular event that was generated by the XMP.

type	identifies the type of event that was generated.
***source**	identifies what the source of the event was. source will either be a handle to an MPI object or a host pointer. Use mpiObjectModuleId() to identify what source points to.
info	Contains information on what generated the event and the conditions under which it was generated. MEIEventStatusInfo simplifies decoding this array. Sample code is shown on the MEIEventStatusInfo page.

See Also

mpiObjectModuleId | MPIEventType | MPIEventMgr | MPINotify | MEIEventStatusInfo | MPIEventStatusINFO_COUNT_MAX

MEIEventStatusInfo

MEIEventStatusInfo

```
typedef struct MEIEventStatusInfo {
    union {
        MPIHandle  handle;  /* generic */
        MPIAxis    axis;    /* MEIEventTypeAXIS_FIRST ... MEIEventTypeAXIS_LAST - 1
*/
        long       node;    /* MEIEventTypeCAN_FIRST... MEIEventTypeCAN_LAST - 1  */
        long       number;  /* MPIEventTypeMOTION MPIEventTypeMOTOR_FIRST
                               ... MPIEventTypeMOTOR_LAST - 1
                                      MEIEventTypeMOTOR_FIRST ...
MEIEventTypeMOTOR_LAST - 1 */
        long       value;   /* MPIEventTypeEXTERNAL */
            } type;

            MEIXmpSignalID signalID;

            /* Contents of addresses specified by MEIEventNotifyData{} */
            union {
         long sampleCounter;
         struct {
              long sampleCounter;
         } motion;
         struct {
              long sampleCounter;
              long actualPosition;
         } axis;
         struct {
            /* Data associated with the CAN event. */
            long data[4];
        } can;
         struct {
              long sampleCounter;
              long encoderPosition;
         } motor;
         long word[MEIXmpSignalUserData];
     } data;
} MEIEventStatusInfo;
```

Description

EventStatusInfo is an information structure that tells the XMP what the data in MPIEventStatus.info holds.

type	A union that specifies the object handle, motion number, or external ID value that generated the event
type.handle	A generic object handle. Used by MPIRecorder and MPIMotor events
type.axis	An axis object handle. Used by MPIAxis events
type.node	The CAN Node number of the MEICan object that generated the event.
type.number	The motion number of the MPIMotion object that generated the event
type.value	An ID value used to identify what external source or MPISequence event was generated
signalID	Specifies what type of object actually generated the event

data	A union that contains extra data about the event that was generated
data.sampleCounter	The value of the sampleCounter when the event was generated
data.motion	A union that contains extra data about the motion event that was generated
data.motion.sampleCounter	The value of the sampleCounter when the motion event was generated
data.axis	A union that contains extra data about the axis event that was generated
data.axis.sampleCounter	The value of the sampleCounter when the axis.event was generated
data.axis.actualPosition	The value of the axis' actual position when the event was generated
data.can.data	A union that contains extra data about the CAN event that was generated.
data.motor	A union that contains extra data about the motor event that was generated
data.motor.sampleCounter	The value of the sampleCounter when the motor event was generated
data.motor.encoderPosition	The value of the motor's ecoder position when the event was generated
data.word[]	The extra data about the event that was generated formatted as an array of long values

Sample Code

```
MPINotify      notify
MPIEventStatus eventStatus;

. . .

/* Wait for event */
returnValue =
   mpiNotifyEventWait(notify,
                      &eventStatus,
                       MPIWaitFOREVER);
msgCHECK(returnValue);

if (eventStatus.type == MPIEventTypeMOTION_DONE) {
   MEIEventStatusInfo *info;

   info = (MEIEventStatusInfo *)eventStatus.info;

   . . .
}
```

See Also

MPIEventStatus | MPIAxis

MPIEventType / MEIEventType

MPIEventType

```
typedef enum {
    MPIEventTypeINVALID,

    MPIEventTypeNONE,                 /*  0 */

    /* Motor events */
    MPIEventTypeAMP_FAULT,            /*  1 */
    MPIEventTypeHOME,                 /*  2 */
    MPIEventTypeLIMIT_ERROR,          /*  3 */
    MPIEventTypeLIMIT_HW_NEG,         /*  4 */
    MPIEventTypeLIMIT_HW_POS,         /*  5 */
    MPIEventTypeLIMIT_SW_NEG,         /*  6 */
    MPIEventTypeLIMIT_SW_POS,         /*  7 */
    MPIEventTypeENCODER_FAULT,        /*  8 */
    MPIEventTypeAMP_WARNING,          /*  9 */

    /* Motion events */
    MPIEventTypeMOTION_DONE,          /* 10 */
    MPIEventTypeMOTION_AT_VELOCITY,   /* 11 */

    /* Recorder events */
    MPIEventTypeRECORDER_HIGH,        /* 12 */
    MPIEventTypeRECORDER_FULL,        /* 13 */
    MPIEventTypeRECORDER_DONE,        /* 14 */

    /* External events */
    MPIEventTypeEXTERNAL,             /* 15 */
} MPIEventType;
```

Description

EventType is used by the MPIEventMask macros to help generate event masks.

MPIEventTypeNONE	This event type indicates no event was generated.
MPIEventTypeAMP_FAULT	This event type indicates an Amp Fault event was generated from a Motor object.
MPIEventTypeHOME	This event type indicates a Home event was generated from a Motor object.
MPIEventTypeLIMIT_ERROR	This event type indicates a position Error Limit was generated from a Motor object.
MPIEventTypeLIMIT_HW_NEG	This event type indicates a Negative Hardware Limit event was generated from a Motor object.

MPIEventTypeLIMIT_HW_POS	This event type indicates a Positive Hardware Limit event was generated from a Motor object.
MPIEventTypeLIMIT_SW_NEG	This event type indicates a Negative Software Limit event was generated from a Motor object.
MPIEventTypeLIMIT_SW_POS	This event type indicates a Positive Software Limit event was generated from a Motor object.
MPIEventTypeENCODER_FAULT	This event type indicates an Encoder Fault event was generated from a Motor object.
MPIEventTypeAMP_WARNING	This event type indicates an Amp Warning event was generated from a Motor object.
MPIEventTypeMOTION_DONE	This event type indicates a Motion Done event was generated from a Motion Supervisor object.
MPIEventTypeMOTION_AT_VELOCITY	This event type indicates an At Velocity event was generated from a Motion Supervisor object.
MPIEventTypeRECORDER_HIGH	This event type indicates that the controller's recorded data exceeded the buffer's high limit.
MPIEventTypeRECORDER_FULL	This event type indicates that the controller's recorded data has filled the buffer.
MPIEventTypeRECORDER_DONE	This event type indicates that the controller has recorded the number of requested data records.
MPIEventTypeEXTERNAL	This event type indicates an External event was generated from an external source.

MEIEventType

```
typedef enum {
    /* Motor events */
    MEIEventTypeLIMIT_USER0,
    MEIEventTypeLIMIT_USER1,
    MEIEventTypeLIMIT_USER2,
    MEIEventTypeLIMIT_USER3,
    MEIEventTypeLIMIT_USER4,
    MEIEventTypeLIMIT_USER5,
    MEIEventTypeLIMIT_USER6,
    MEIEventTypeLIMIT_USER7,
    MEIEventTypeLIMIT_USER8,
    MEIEventTypeLIMIT_USER9,
    MEIEventTypeLIMIT_USER10,
    MEIEventTypeLIMIT_USER11,
    MEIEventTypeLIMIT_USER12,
    MEIEventTypeLIMIT_USER13,
    MEIEventTypeLIMIT_USER14,
    MEIEventTypeLIMIT_USER15,
```

```
    /* Motion events */
    MEIEventTypeMOTION_OUT_OF_FRAMES,
    MEIEventTypeMOTION_RESERVED0,

    /* Axis events */
    MEIEventTypeIN_POSITION_COARSE,
    MEIEventTypeIN_POSITION_FINE,
    MEIEventTypeSETTLED
    MEIEventTypeAT_TARGET,
    MEIEventTypeFRAME,
    MEIEventTypeAXIS_RESERVED0,
    MEIEventTypeAXIS_RESERVED1,

    /* SynqNet events */
    MEIEventTypeSYNQNET_DEAD,
    MEIEventTypeSYNQNET_RX_FAILURE,
    MEIEventTypeSYNQNET_TX_FAILURE,
    MEIEventTypeSYNQNET_NODE_FAILURE,
    MEIEventTypeSYNQNET_RECOVERY,

    /* SqNode events */
    MEIEventTypeSQNODE_IO_ABORT,
    MEIEventTypeSQNODE_NODE_DISABLE,
    MEIEventTypeSQNODE_NODE_ALARM,
    MEIEventTypeSQNODE_ANALOG_POWER_FAULT,
    MEIEventTypeSQNODE_USER_FAULT,
    MEIEventTypeSQNODE_NODE_FAILURE,

    /* CAN events */
    MEIEventTypeCAN_BUS_STATE,
    MEIEventTypeCAN_RECEIVE_OVERRUN,
    MEIEventTypeCAN_EMERGENGY,
    MEIEventTypeCAN_NODE_BOOT,
    MEIEventTypeCAN_HEALTH,
    MEIEventTypeCAN_DIGITAL_INPUT,
    MEIEventTypeCAN_ANALOG_INPUT,

} MEIEventType;
```

Description

EventType is used by the MPIEventMask macros to help generate event masks.

MEIEventTypeLIMIT_USER0

This event type indicates a User Limit event was generated from a Motor object. User Limit number 0.

MEIEventTypeLIMIT_USER1

This event type indicates a User Limit event was generated from a Motor object. User Limit number 1.

MEIEventTypeLIMIT_USER2

This event type indicates a User Limit event was generated from a Motor object. User Limit number 2.

MEIEventTypeLIMIT_USER3

This event type indicates a User Limit event was generated from a Motor object. User Limit number 3.

MEIEventTypeLIMIT_USER4

This event type indicates a User Limit event was generated from a Motor object. User Limit number 4.

MEIEventTypeLIMIT_USER5

This event type indicates a User Limit event was generated from a Motor object. User Limit number 5.

MEIEventTypeLIMIT_USER6

This event type indicates a User Limit event was generated from a Motor object. User Limit number 6.

MEIEventTypeLIMIT_USER7

This event type indicates a User Limit event was generated from a Motor object. User Limit number 7.

MEIEventTypeLIMIT_USER8

This event type indicates a User Limit event was generated from a Motor object. User Limit number 8.

MEIEventTypeLIMIT_USER9

This event type indicates a User Limit event was generated from a Motor object. User Limit number 9.

MEIEventTypeLIMIT_USER10

This event type indicates a User Limit event was generated from a Motor object. User Limit number 10.

MEIEventTypeLIMIT_USER11

This event type indicates a User Limit event was generated from a Motor object. User Limit number 11.

MEIEventTypeLIMIT_USER12

This event type indicates a User Limit event was generated from a Motor object. User Limit number 12.

MEIEventTypeLIMIT_USER13

This event type indicates a User Limit event was generated from a Motor object. User Limit number 13.

MEIEventTypeLIMIT_USER14

This event type indicates a User Limit event was generated from a Motor object. User Limit number 14.

MEIEventTypeLIMIT_USER15

This event type indicates a User Limit event was generated from a Motor object. User Limit number 15.

MEIEventTypeMOTION_OUT_OF_FRAMES

This event type indicates a Motion Done event was generated from a Motion Supervisor object.

MEIEventTypeMOTION_RESERVED0

This event type indicates a Reserved Motion event was generated from a Motion Supervisor object. This event type is reserved for future use or custom motion events.

MEIEventTypeIN_POSITION_COARSE

This event type indicates an In Coarse Position event was generated from an Axis object.

MEIEventTypeIN_POSITION_FINE

This event type indicates that an In Fine Position event was generated from an Axis object.

MEISynqNetMessageREADY_TIMEOUT

The node failed to be ready for a service command within the timeout. This message code is returned by MPI methods that fail a service command transaction because the node is not ready to accept service commands. To correct this problem, check your node hardware. There are 32 possible message codes for this error. Each message code specifies a different node, from node number 0 to 31.

MEIEventTypeSETTLED

Equivalent to MEIEventTypeIN_POSITION_FINE.

MEIEventTypeAT_TARGET

Reserved Frame Event.

MEIEventTypeFRAME

This event type is currently not supported and is reserved for future use.

MEIEventTypeAXIS_RESERVED0

This event type indicates a Reserved Axis event was generated from an Axis object.
This event type is currently not supported and is reserved for future use or custom axis events.

MEIEventTypeAXIS_RESERVED1

This event type indicates that a Reserved Axis event was generated from an Axis object.
This event type is currently not supported and is reserved for future use or custom axis events.

MEIEventTypeSYNQNET_DEAD

The SynqNet network was shutdown due to a communication failure. This status/event occurs when the controller fails to read/write data to the SynqNet network interface from an RX_FAILURE or a TX_FAILURE. To recover from a DEAD event, the network must be shutdown and reinitialized. SYNQNET_DEAD is latched by the controller, use meiSynqNetEventReset(…) to clear the status/event bit.

MEIEventTypeSYNQNET_RX_FAILURE

SynqNet network data receive failure. Generated when the controller fails to receive the packet data buffer (Rincon DMA to internal memory) in two successive controller samples. A SYNQNET_RX_FAILURE is most likely caused by an incorrect RX_COPY_TIMER value (internal) or a timing problem. To recover from an RX_FAILURE event, the network must be shutdown and reinitialized. SYNQNET_RX_FAILURE is latched by the controller, use meiSynqNetEventReset(…) to clear the status/event bit.

MEIEventTypeSYNQNET_TX_FAILURE

SynqNet network data transmission failure. Generated when the controller fails to transmit the packet data buffer in two successive controller samples. This occurs when the maximum foreground time exceeds the Tx time percentage of the controller's sample period. The default Tx time value is 75% of the controller's sample period. To correct Tx failures, either increase the Tx time or decrease the controller's sample rate. To recover from a TX_FAILURE event, the network must be shutdown and reinitialized. SYNQNET_TX_FAILURE is latched by the controller, use meiSynqNetEventReset(…) to clear the status/event bit.

MEIEventTypeSYNQNET_NODE_FAILURE

SynqNet node failure. Generated when any node's upstream or downstream packet error rate counters exceed the failure limit. The failure limits are configured with meiSqNodeConfigSet(…). Use meiSynqNetStatus(…) to read the nodeFailedMask to identify the failed nodes. Also, a SQNODE_NODE_FAILURE will be generated for each node that fails. SYNQNET_NODE_FAILURE is latched by the controller, use meiSynqNetEventReset(…) to clear the status/event bit. To recover from a node failure, the network must be shutdown and reinitialized.

MEIEventTypeSYNQNET_RECOVERY

SynqNet fault recovery. Generated when any node's upstream or downstream packet error rate counters exceed the fault limit and the data traffic is redirected around the fault. The fault limits are configurable via meiSqNodeConfigSet(…). SYNQNET_RECOVERY is latched by the controller. Use meiSynqNetEventReset(…) to clear the status/event bit.

MEIEventTypeSQNODE_IO_ABORT

SynqNet node I/O abort. Generated when the node I/O Abort is activated. When the I/O Abort is triggered, the node's outputs are disabled (set to the power-on condition). The node I/O Abort can be configured to trigger when either a Synq Lost occurs, Node Disable is active, a Power Fault occurs, or a User Fault is triggered. See MEISqNodeConfigIoAbort{…} for more details.

MEIEventTypeSQNODE_NODE_DISABLE

SynqNet node's Node Disable input is activated. Generated when the Node Disable input signal transitions from inactive to active. This signal is latched in hardware. Use meiSqNodeEventReset(…) to clear the status/event and the hardware latch.

MEIEventTypeSQNODE_NODE_ALARM

SynqNet node alarm digital output. Generated when the node alarm output signal transitions from inactive to active. This signal is not latched.

MEIEventTypeSQNODE_ANALOG_POWER_FAULT

SynqNet node analog power failure. Generated when the node's power failure input bit transitions from inactive to active. The power fault circuit is node specific, but is typically connected to an analog power monitor. This signal is latched in hardware. Use meiSqNodeEventReset(…) to clear the status/event and the hardware latch.

MEIEventTypeSQNODE_USER_FAULT

SynqNet node user fault. Generated when the node's user configurable fault is triggered. The user fault can be configured to monitor any controller memory address and compare the masked value to a specified pattern. This signal is latched by the controller, use meiSqNodeEventReset(…) to clear the status/event bit.

MEIEventTypeSQNODE_NODE_FAILURE

SynqNet node failure. Generated when a node's upstream or downstream packet error rate counters exceed the failure limit. The failure limits are configured with meiSqNodeConfigSet(…). SQNODE_NODE_FAILURE is latched by the controller, use meiSqNodeEventReset(…) to clear the status/event bit. To recover from a node failure, the network must be shutdown and reinitialized.

MEIEventTypeCAN_BUS_STATE

The BusState has changed. Data[0] contains the new bus state.

MEIEventTypeCAN_RECEIVE_OVERRUN

The CAN hardware detected a receive overrun.

MEIEventTypeCAN_EMERGENGY

An emergency message was received from a node. Data[0] contains the node number. Data[1 to 4] contains the contents of the emergency message.

MEIEventTypeCAN_NODE_BOOT

A node boot message was received from a node. Data[0] contains the node number.

MEIEventTypeCAN_HEALTH

The health of a node has changed. Data[0] contains the node number. Data[1] contains the new node health.

MEIEventTypeCAN_DIGITAL_INPUT

A digital input event was received from a node. Data[0] contains the node number. Data[1 to 4] contains the new input state.

MEIEventTypeCAN_ANALOG_INPUT

An analog input event was received from a node. Data[0] contains the node number. Data[1 to 4] contains the new input state.

See Also

MPIEventMask | MPIEventMgr | MPINotify | MPIEventStatus | meiSynqNetEventReset | Special Note on the use of MPIEventTypeENCODER_FAULT

Special Note: *Use of MPIEventTypeENCODER_FAULT*

This event type is used to detect three types of encoder faults:

- **Broken wire errors**
- **Illegal state errors**
- **Absolute encoder initialization errors**
  - Timeout errors
  - Protocol errors

Broken wire errors are detected for either incremental or absolute encoders whenever both differential inputs of any encoder receiver (A, B, or Index) are at the same voltage level (i. e., whenever one or both inputs is disconnected from the encoders differential transmitter). The EncoderTermination configuration of the encoder input must be TRUE for correct detection of broken wires.

Illegal state errors occur whenever transitions are seen on both A and B phases of an encoder input at the same time (e.g. noise spikes).

There are two types of **absolute encoder initialization errors**: Timeout errors and Protocol errors. **Timeout errors** occur when an absolute encoder does not transmit absolute encoder data within the timeout period starting at the transition of the interrogation line (SEN line). **Protocol errors** are detected when serial absolute data is sent during the timeout, but the data cannot be interpreted by the XMP. Both error types result in an ENCODER_FAULT event.

Return to MPIEventType

MPIEventStatusINFO_COUNT_MAX

MPIEventStatusINFO_COUNT_MAX

```
#define MPIEventStatusINFO_COUNT_MAX    (16)
```

Description **EventStatusINFO_COUNT_MAX** defines the size of the MPIEventStatus.info[] array.

See Also MPIEventStatus | MPIEventMgr | MPINotify