

Command Objects

Introduction

The **Command** object specifies one of a variety of program Sequence commands. These include motion, conditional branch, computational, and time delay commands.

Information about the different types of commands can be found on [MPICommandType](#) and [MPICommandParams](#).

| [Error Messages](#) |

Methods

Create, Delete, Validate Methods

mpiCommandCreate	Create Command object
mpiCommandDelete	Delete Command object
mpiCommandValidate	Validate Command object

Configuration and Informational Methods

mpiCommandLabel	Get pointer to Command label
mpiCommandParams	Get Command parameters
mpiCommandType	Return Command type

Other Methods

meiCommandAxisListGet	Get the axisCount and axisList from a Command object.
---------------------------------------	---

Data Types

[MPICommandAddress](#)
[MPICommandConstant](#)
[MPICommandExpr](#)
[MPICommandMessage](#)
[MPICommandMotion](#)
[MPICommandOperator](#)
[MPICommandParams](#)
[MPICommandType](#)

See Also

[MPISequence](#)

mpiCommandCreate

Declaration

```

MPICommand mpiCommandCreate(MPICommandType type,
                             MPICommandParams *params,
                             const char *label)

```

Required Header: stdmei.h

Description

mpiCommandCreate creates a Command object. The command type is specified by *type*. The type-specific parameters are specified by *params*. If *label* is not Null (i.e., something meaningful), then branch commands can call this Command (by using the *label*).

CommandCreate is the equivalent of a C++ constructor.

Return Values

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

See Also

[mpiCommandDelete](#) | [mpiCommandValidate](#)

mpiCommandType

Declaration

```
long mpiCommandType(MPICommand      command ,
                   MPICommandType *type )
```

Required Header: stdmpi.h

Description

mpiCommandType gets the type from a Command and puts it in the location pointed to by type.

command	a handle to the Command object
*type	a pointer to a MPICommandType returned by the method

Return Values	
MPIMessageOK	
Command (<i>command</i>) parameters	in the location pointed to by <i>type</i>

See Also

[mpiCommandCreate](#) | [MPICommandType](#)

mpiCommandParams

Declaration

```
long mpiCommandParams ( MPICommand      command ,
                        MPICommandParams *params )
```

Required Header: stdmpi.h

Description

mpiCommandParams gets the parameters from a Command and puts it in the location pointed to by params.

command	a handle to the Command object
*params	a pointer to a MPICommandParams structure returned by the method

Return Values	
MPIMessageOK	
Command (<i>command</i>) parameters	in the structure pointed to by <i>params</i>

See Also

[mpiCommandCreate](#) | [MPICommandParams](#)

mpiCommandDelete

Declaration

```
long mpiCommandDelete(MPICommand command)
```

Required Header: stdmpi.h

Description

mpiCommandDelete deletes a Command object and invalidates its handle (*command*).

CommandDelete is the equivalent of a C++ destructor.

Return Values

[MPIMessageOK](#)

See Also

[mpiCommandCreate](#) | [mpiCommandValidate](#)

mpiCommandValidate

Declaration

```
long mpiCommandValidate(MPICommand command)
```

Required Header: stdmpi.h

Description

mpiCommandValidate validates the Command object and its handle (*command*).

command	a handle to the Command object
----------------	--------------------------------

Return Values

MPIMessageOK	
------------------------------	--

See Also

[mpiCommandCreate](#) | [mpiCommandValidate](#)

mpiCommandLabel

Declaration

```
long mpiCommandLabel(MPICommand command,
                    char **label)
```

Required Header: stdmpi.h

Description

mpiCommandLabel gets the string from a Command and puts it in the location pointed to by label.

command	a handle to the Command object
**label	a pointer to a string returned by the method

Return Values	
MPIMessageOK	
pointer	to a Command's (<i>command</i>) label (that is in the location pointed to by <i>label</i>)

See Also

[mpiCommandCreate](#)

meiCommandAxisListGet

Declaration

```
long meiCommandAxisListGet(MPICommand  command,
                           long           *axisCount
                           MPIAxis      *axisList)
```

Required Header: stdmei.h

Description

meiCommandAxisListGet reads number of axes and the list of axes associated with a motion type Command object (***command***) and writes them into the long pointed to by ***axisCount*** and the array of axis objects pointed to by ***axisList***.

command	a handle to the Command object
*axisCount	a pointer to a long, representing the number of axes returned by the method
*axisList	a pointer to an array of axis objects returned by the method

Return Values

[MPIMessageOK](#)

See Also

[MPICommand](#) | [MPIAxis](#) | [MPIMotion](#)

MPICommandAddress

Definition

```
typedef union {  
    long    *l;  
    float   *f;  
} MPICommandAddress;
```

Description

MPICommandAddress defines a generic pointer that can specify either a long or a float pointer.

*l	is used to access the long pointer of MPICommandAddress.
*f	is used to access the float pointer of MPICommandAddress.

See Also

[MPICommandConstant](#)

MPICommandConstant

Definition

```
typedef union {  
    long    l;  
    float   f;  
} MPICommandConstant;
```

Description

MPICommandConstant defines a generic variable that can specify either a *long* or *float* value.

l	is used to access the long value of MPICommandConstant.
f	is used to access the float value of MPICommandConstant.

See Also

[MPICommandAddress](#)

MPICommandExpr

Definition

```
typedef struct MPICommandExpr {
    MPICommandOperator  oper;
    MPICommandAddress  address;
    union {
        MPICommandConstant  value; /* ['address'] 'oper' ['value'] */
        MPICommandAddress  ref;    /* ['address'] 'oper' ['ref'] */
    } by;
} MPICommandExpr;
```

Description

MPICommandExpr is a structure that represents an expression for an MPICommand object.

The expression is evaluated as either:

	*address oper value
	*address oper *ref

depending on the command type.

See Also

[MPICommand](#) | [MPICommandParams](#) | [MPICommandType](#)

MPICommandMessage

Definition

```
typedef enum {  
    MPICommandMessageCOMMAND_INVALID,  
    MPICommandMessageTYPE_INVALID,  
    MPICommandMessagePARAM_INVALID,  
} MPICommandMessage;
```

Description

MPICommandMessageCOMMAND_INVALID

Currently not supported and is reserved for future use.

MPICommandMessageTYPE_INVALID

The command type is not valid. This message code is returned by [mpiCommandCreate\(...\)](#) if the command type is not a member of the [MPICommandType](#) enumeration.

MPICommandMessagePARAM_INVALID

Currently not supported and is reserved for future use.

See Also

[MPICommandType](#)

MPICommandMotion

Definition

```
typedef enum {
    MPICommandMotionABORT,
    MPICommandMotionE_STOP,
    MPICommandMotionE_STOP_MODIFY,
    MPICommandMotionE_STOP_ABORT,
    MPICommandMotionE_STOP_CMD_EQ_ACT,
    MPICommandMotionMODIFY,
    MPICommandMotionRESET,
    MPICommandMotionRESUME,
    MPICommandMotionSTART,
    MPICommandMotionSTOP,
} MPICommandMotion;
```

Change History: Modified in the 03.03.00

Description

MPICommandMotion is an enumeration of motion specific controller commands that can be used in a program sequence. It specifies a single motion action for the controller to execute. The CommandMotion also defines the command parameters that must be passed to [mpiCommandCreate](#). For MPICommandMotion, there is a corresponding motion{...} structure in the [MPICommandParams](#) structure.

MPICommandMotionABORT	Commands an Abort action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionABORT for details.
MPICommandMotionE_STOP	Commands an E-Stop action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionE_STOP for details.
MPICommandMotionE_STOP_MODIFY	Commands an E-Stop Modify action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionE_STOP_MODIFY for details.

MPICCommandMotionE_STOP_ABORT	Commands an E-Stop, then Abort action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionE_STOP_ABORT for details.
MPICCommandMotionE_STOP_CMD_EQ_ACT	Commands an E-Stop (command position = actual position) action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionE_STOP_CMD_EQ_ACT for details.
MPICCommandMotionMODIFY	Commands a Motion Modify on the motion supervisor associated with the motion object. Make sure to specify the MPIMotionType and MPIMotionParams in the MPICCommandParams{...} structure. See mpiMotionModify(...) for details.
MPICCommandMotionRESET	Commands a Reset action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionRESET for details.
MPICCommandMotionRESUME	Commands a Resume action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionRESUME for details.
MPICCommandMotionSTART	Commands a Motion Start on the motion supervisor associated with the motion object. Make sure to specify the MPIMotionType and MPIMotionParams in the MPICCommandParams{...} structure. See mpiMotionStart(...) for details.
MPICCommandMotionSTOP	Commands a Stop action on the motion supervisor associated with the motion object. See mpiMotionAction(...) , MPIActionSTOP for details.

See Also

[MPIAction](#) | [MPICCommand](#) | [MPICCommandParams](#)

MPICommandOperator

Definition

```
typedef enum {
    /* Arithmetic operators */
    MPICommandOperatorADD,
    MPICommandOperatorSUBTRACT,
    MPICommandOperatorMULTIPLY,
    MPICommandOperatorDIVIDE,

    MPICommandOperatorAND,
    MPICommandOperatorOR,
    MPICommandOperatorXOR,

    /* Logical operators */
    MPICommandOperatorALWAYS,

    MPICommandOperatorEQUAL,
    MPICommandOperatorNOT_EQUAL,

    MPICommandOperatorGREATER_OR_EQUAL,
    MPICommandOperatorGREATER,

    MPICommandOperatorLESS_OR_EQUAL,
    MPICommandOperatorLESS,

    MPICommandOperatorBIT_CLEAR,
    MPICommandOperatorBIT_SET,
} MPICommandOperator;
```

Description

The following are operators used by the MPICommand and MPICompare objects.

Arithmetic Operators	
MPICommandOperatorADD	Performs an addition. Equivalent to the C operator (+).
MPICommandOperatorSUBTRACT	Performs a subtraction. Equivalent to the C operator (-).
MPICommandOperatorMULTIPLY	Performs a multiplication. Equivalent to the C operator (*).

MPICommandOperatorDIVIDE	Performs a division. Equivalent to the C operator (/).
MPICommandOperatorAND	Performs a logical AND. Equivalent to the C operator (&).
MPICommandOperatorOR	Performs a logical OR. Equivalent to the C operator ().
MPICommandOperatorXOR	Performs a logical XOR. Equivalent to the C operator (^).

Logical Operators	
MPICommandOperatorALWAYS	Always evaluates TRUE. Equivalent in C to (1) or TRUE.
MPICommandOperatorEQUAL	Performs an equality comparison. Equivalent to the C operator (==)
MPICommandOperatorGREATER_OR_EQUAL	Performs an inequality comparison. Equivalent to the C operator (!=)
MPICommandOperatorGREATER_OR_EQUAL	Performs a greater than or equal to comparison. Equivalent to the C operator (>=)
MPICommandOperatorGREATER	Performs a greater than comparison. Equivalent to the C operator (>)
MPICommandOperatorLESS_OR_EQUAL	Performs a less than or equal to comparison. Equivalent to the C operator (<=)
MPICommandOperatorLESS	Performs a less than comparison. Equivalent to the C operator (<)
MPICommandOperatorBIT_CLEAR	Clears specified bits. Equivalent in C to the statement: variable &= ~(bits)
MPICommandOperatorBIT_SET	Sets specified bits. Equivalent in C to the statement: variable = (bits)

See Also

[MPICommand](#) | [MPICommandExpr](#) | [MPICommandParams](#)

MPICommandParams

Definition

```

typedef union {
    struct { /* *'dst' = 'value' */
        MPICommandAddress    dst;
        MPICommandConstant value;
        MPIControl           control; /* Ignored by Sequence */
    } assign;

    struct { /* branch to 'label' on 'expr' */
        char                *label; /* NULL => stop sequence */
        MPICommandExpr      expr; /* expr.oper => MPICommandOperatorLogical */
        MPIControl         control; /* Ignored by Sequence */
    } branch;

    struct { /* branch to 'label' on MPIEventMask('handle') 'oper' 'mask' */
        char                *label; /* NULL => stop sequence */
        MPIHandle           handle; /* [MPIMotor|MPIMotion|...] */
        MPICommandOperator oper; /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
        MPIEventMask      mask; /* MPIEventMask('handle') 'oper' 'mask' */
    } branchEvent;

    struct { /* branch to 'label' on Io.input 'oper' 'mask' */
        char                *label; /* NULL => stop sequence */
        MPIIoType           type; /* MOTOR, USER */
        MPIIoSource        source; /* MPIMotor index */
        MPICommandOperator oper; /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
        long                mask; /* [motor|user]Io.input 'oper' 'mask' */
    } branchIO;

    struct { /* *'dst' = 'expr' */
        MPICommandAddress    dst;
        MPICommandExpr      expr; /* expr.oper => MPICommandOperatorArithmetic */
        MPIControl         control; /* Ignored by Sequence */
    } compute;

    struct { /* Io.output = Io.output 'oper' 'mask' */
        MPIIoType           type; /* MOTOR, USER */
        MPIIoSource        source; /* MPIMotor index */
        MPICommandOperator oper; /* AND/OR/XOR */
        long                mask;
    } computeIO;

    struct { /* memcpy(dst, src, count) */
        void                *dst;
        void                *src;
        long                count;
        MPIControl         control; /* Ignored by Sequence */
    } copy;

    float delay; /* seconds */

```

```

struct {
    long          value;      /* MPIEventStatus.type    = MPIEventTypeEXTERNAL */
                                /*                          .source = MPISequence/MPIProgram */
                                /*                          .info[0] = value */
    MPIEventMgr  eventMgr; /* Ignored by Sequence */
} event;

struct { /* mpiMotion[Abort|EStop|Reset|Resume|Start|Stop](motion[, type,
params]) */
    MPICommandMotion  motionCommand;
    MPIMotion          motion;
    MPIMotionType     type;      /* MPICommandMotionSTART */
    MPIMotionParams  params;    /* MPICommandMotionSTART */
} motion;

struct { /* wait until 'expr' */
    MPICommandExpr    expr;      /* expr.oper => MPICommandOperatorLogical */
    MPIControl        control; /* Ignored by Sequence */
} wait;

struct { /* wait until MPIEventMask('handle') 'oper' 'mask' */
    MPIHandle         handle; /* [MPIMotor|MPIMotion|...] */
    MPICommandOperator oper; /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
    MPIEventMask      mask; /* MPIEventMask('handle') 'oper' 'mask' */
} waitEvent;

struct { /* wait until Io.input 'oper' 'mask' */
    MPIIoType         type; /* MOTOR, USER */
    MPIIoSource      source; /* MPIMotor index */
    MPICommandOperator oper; /* EQUAL/NOT_EQUAL/BIT_CLEAR/BIT_SET */
    long              mask; /* [motor|user]Io.input 'oper' 'mask' */
} waitIO;
} MPICommandParams;

```

Description

MPICommandParams holds the parameters used by an MPICommand. Each element in the MPICommandParams union corresponds to different types of commands (specified by the MPICommandType enumeration).

Element	Description	Supported by
assign	Assign a value to a particular controller address: *dst = value assign.control is currently not supported and is reserved for future use.	MPICommandTypeASSIGN MPICommandTypeASSIGN_FLOAT

branch	<p>Branch to a particular command (similar to a <i>goto</i> statement) if a particular comparison evaluates to TRUE: branch to label on expr</p> <p>If <i>label</i> = NULL, then no more commands will be executed if the comparison evaluates to TRUE.</p> <p>branch.control is currently not supported and is reserved for future use.</p>	<p>MPICommandTypeBRANCH MPICommandTypeBRANCH_REF MPICommandTypeBRANCH_FLOAT MPICommandTypeBRANCH_FLOAT_REF</p>
branchEvent	<p>Branch to a particular command (similar to a <i>goto</i> statement) if a particular event occurs or has occurred: branch to label on</p> <p>MPIEventMask(handle) oper mask</p> <p>If <i>label</i> = NULL, then no more commands will be executed if a particular event occurs or has occurred.</p>	<p>MPICommandTypeBRANCH_EVENT</p>
branchIO	<p>Branch to a particular command (similar to a <i>goto</i> statement) if a particular i/o state matches a specified condition: branch to label on Io.input oper mask</p> <p>If <i>label</i> = NULL, then no more commands will be executed if a particular i/o state matches a specified condition.</p>	<p>MPICommandTypeBRANCH_IO</p>
compute	<p>perform some computation and place the result at some controller address: *dst = expr</p> <p>compute.control is currently not supported and is reserved for future use.</p>	<p>MPICommandTypeCOMPUTE MPICommandTypeCOMPUTE_REF MPICommandTypeCOMPUTE_FLOAT MPICommandTypeCOMPUTE_FLOAT_REF</p>
computeIO	<p>Performs a computation on a set of i/o bits: <i>Io.output = Io.output oper mask</i></p>	<p>MPICommandType_IO</p>
copy	<p>Copies controller memory from one place to another: memcpy(dst, src, count);</p> <p>Remember: count represents the number of bytes copied, NOT the number of controller words.</p> <p>event.control is currently not supported and is reserved for future use.</p>	<p>MPICommandTypeCOPY</p>
delay	<p>Delays execution of the next command <i>delay</i> seconds.</p>	<p>MPICommandTypeDELAY</p>
event	<p>Generates an event: MPIEventStatus.type = MPIEventTypeEXTERNAL MPIEventStatus.source = MPISequence MPIEventStatus.info[0] = value</p> <p>event.eventMgr is currently not supported and is reserved for future use.</p>	<p>MPICommandTypeEVENT</p>
motion	<p>Commands a motion action (See MPICommandMotion):</p> <p>mpiMotionStart (motion, type, params)]; or mpiMotionAction(motion, MPIAction[ABORT E_STOP E_STOP_ABORT RESET RESUME STOP]);</p>	<p>MPICommandTypeMOTION</p>

wait	Delays execution of the next command until a particular comparison evaluates to TRUE: wait until <i>expr</i> wait.control is currently not supported and is reserved for future use.	MPICommandTypeWAIT MPICommandTypeWAIT_REF MPICommandTypeWAIT_FLOAT MPICommandTypeWAIT_FLOAT_REF
waitEvent	Delays execution of the next command until a particular event occurs: wait until MPIEventMask (<i>handle</i>) oper <i>mask</i>	MPICommandTypeWAIT_EVENT
waitIO	Delays execution of the next command until a particular i/o state matches a specified condition: wait until <i>Io.input</i> oper <i>mask</i>	MPICommandTypeWAIT_IO

See Also

[MPICommand](#) | [MPICommandType](#) | [mpiCommandCreate](#) | [mpiCommandParams](#)

MPICommandType

Definition

```
typedef enum {
    MPICommandTypeASSIGN,
    MPICommandTypeASSIGN_FLOAT,

    MPICommandTypeBRANCH,
    MPICommandTypeBRANCH_REF,
    MPICommandTypeBRANCH_FLOAT,
    MPICommandTypeBRANCH_FLOAT_REF,
    MPICommandTypeBRANCH_EVENT,
    MPICommandTypeBRANCH_IO,

    MPICommandTypeCOMPUTE,
    MPICommandTypeCOMPUTE_REF,
    MPICommandTypeCOMPUTE_FLOAT,
    MPICommandTypeCOMPUTE_FLOAT_REF,
    MPICommandTypeCOMPUTE_IO,

    MPICommandTypeCOPY,
    MPICommandTypeDELAY,
    MPICommandTypeEVENT,
    MPICommandTypeMOTION,

    MPICommandTypeWAIT,
    MPICommandTypeWAIT_REF,
    MPICommandTypeWAIT_FLOAT,
    MPICommandTypeWAIT_FLOAT_REF,
    MPICommandTypeWAIT_EVENT,
    MPICommandTypeWAIT_IO,
} MPICommandType;
```

Description

MPICommandType is an enumeration of controller commands that can be used in a program sequence. It specifies a single instruction for the controller to execute. The **CommandType** also defines the command parameters that must be passed to `mpiCommandCreate(...)`. For each **MPICommandType** there is a corresponding structure in the `MPICommandParams{...}` union. For example, when the `MPICommandTypeASSIGN` is specified, the `assign{...}` structure in `MPICommandParams{...}` must be filled in to specify the address and value.

Commands must be created with `mpiCommandCreate(...)` and then added to a sequence using `mpiSequenceCommandAppend(...)`, `mpiSequenceCommandInsert(...)`, or `mpiSequenceCommandListSet(...)`. Then the command sequence can be loaded into the controller with `mpiSequenceLoad(...)` and started with `mpiSequenceStart(...)`.

Element	Description	Associated MPICommandParams structure
MPICommandTypeASSIGN	Writes a constant value (long or float) into the controller's memory at the specified address.	assign
MPICommandTypeASSIGN_FLOAT	These commands assign a value to a particular controller address. MPICommandTypeASSIGN assigns a long value while MPICommandTypeASSIGN_FLOAT assigns a float value.	
MPICommandTypeBRANCH	These commands branch to a particular command (similar to a goto statement) if a particular comparison evaluates to TRUE. MPICommandTypeBRANCH compares a controller address to a specified constant long value. MPICommandTypeBRANCH_REF compares a controller address to a long value at a specified controller address.	branch
MPICommandTypeBRANCH_REF	Branch to a particular command if the comparison evaluates to TRUE. Compares a controller address to a long value at a specified controller address.	
MPICommandTypeBRANCH_FLOAT	Compares a controller address to a specified constant float value.	
MPICommandTypeBRANCH_FLOAT_REF	Compares a controller address to a float value at a specified controller address.	
MPICommandTypeBRANCH_EVENT	Branch to a particular command (similar to a goto statement) if a particular event occurs or has occurred.	branchEvent
MPICommandTypeBRANCH_IO	Branch to a particular command (similar to a goto statement) if a particular I/O state matches a specified condition.	branchIO

MPICommandTypeCOMPUTE	These commands perform some computation and place the result at some controller address. MPICommandTypeCOMPUTE performs a computation of some controller address and a constant long value.	compute
MPICommandTypeCOMPUTE_REF		
MPICommandTypeCOMPUTE_FLOAT	Performs a computation of some controller address and a constant float value.	
MPICommandTypeCOMPUTE_FLOAT_REF	Performs a computation of some controller address and a float value at a specified controller address.	
MPICommandTypeCOMPUTE_IO	Performs a computation on a set of I/O bits.	computeIO
MPICommandTypeCOPY	Copies controller memory from one place to another.	copy
MPICommandTypeDELAY	Delays execution of the next command.	delay
MPICommandTypeEVENT	Generate an event.	event
MPICommandTypeMOTION	Commands a motion action. See MPICommandMotion .	motion
MPICommandTypeWAIT	These delays execution of the next command until a particular comparison evaluates to TRUE. MPICommandTypeWAIT compares a controller address to a specified constant long value. MPICommandTypeWAIT_REF Compares a controller address to a long value at a specified controller address.	wait
MPICommandTypeWAIT_REF	Compares a controller address to a long value at a specified controller address.	
MPICommandTypeWAIT_FLOAT	Compares a controller address to a specified constant float value.	
MPICommandTypeWAIT_FLOAT_REF	Compares a controller address to a float value at a specified controller address.	

MPICommandTypeWAIT_EVENT	Delays execution of the next command until a particular event occurs.	waitEvent
MPICommandTypeWAIT_IO	Delays execution of the next command until a particular I/O state matches a specified condition.	waitIO

See Also

[MPICommand](#) | [MPICommandMotion](#) | [MPICommandParams](#) | [mpiCommandCreate](#) | [mpiCommandType](#) | [mpiSequenceCommandAppend](#) | [mpiSequenceCommandInsert](#) | [mpiSequenceCommandListSet](#) | [mpiSequenceLoad](#) | [mpiSequenceStart](#)