Capture Object

Introduction

A **Capture** object manages a single position capture logic block. It represents the physical hardware capture logic and data. When configured and armed, the capture logic block can latch a motor's position based on one or more source input triggers.

The Capture object's number, motor input trigger sources, edge, type, feedback source, and capture index are all configurable. There are two capture types: Position and Time based. For the Position type, the position counters are latched in the FPGA and are read directly by the controller. This methodology works well for incremental quadrature encoders. For the Time type, the FPGA latches the clock and the controller reads the clock value and position value for that sample period. The controller interpolates the position value from the previous sample's position, the present sample's position, and the clock data. This methodology works very well for cyclic feedback data that is digitally transmitted from the drive to the FPGA. Many drives have a proprietary serial encoder that decodes the encoder position and sends the position information to the FPGA once per sample. In these cases, time-based capture is more accurate than position-based capture.

For the **Position** type, the motor number for the input sources and the feedback motor number must be the same.

For the **Time** type, the motor number and feedback motor number can be different. This makes is possible to use inputs from one node to capture positions on another node.

When using captures, the controller must have enough enabled captures to process the specified capture number. The controller will process the enabled captures (captureCount) every sample period. Since each capture object is configurable, use the minimum number of captures possible for best controller performance. For example, if you want to use 2 captures for motor 0 and motor 3, set the capture count to 2 and use capture number 0 and 1.

Methods

Create, Delete, Validate Methods

mpiCapture**Create**	Create Capture object
mpiCapture**Delete**	Delete Capture object
mpiCapture**Validate**	Validate Capture object

Configuration and Information Methods

mpiCapture**ConfigGet**	Get Capture configuration
mpiCapture**ConfigSet**	Set Capture configuration
mpiCapture**Status**	Get status of Capture
mpiCapture**ConfigReset**	

Action Methods

mpiCapture**Arm**	Arm capture object

Memory Methods

mpiCapture**Memory**	Set address to Capture memory
mpiCapture**MemoryGet**	Copy Capture memory to application memory
mpiCapture**MemorySet**	Copy application memory to Capture memory

Relational Methods

mpiCapture**Number**	Get index of Capture (for Control list)

Data Types

MPICapture**Config**

MPICapture**Edge**

MPICapture**Message** / MEICapture**Message**

MPICapture**Source**

MPICapture**State**

MPICapture**Status**

MPICapture**Trigger**

MPICapture**TriggerGlobal**

MPICapture**Type**

Constants

MPICapture**NOT_MAPPED**

mpiCaptureCreate

Declaration

```
MPICapture mpiCaptureCreate(MPIControl     control,
                            long           number);
```

Required Header

stdmpi.h

Description

CaptureCreate creates a Capture object. The Capture object is identified by its association with a motor object, the motor's encoder and the encoder's capture number. The maximum number of enabled captures is 16.

CaptureCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	An index to the encoder's capture block.

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

See Also

mpiCaptureNumber

mpiCaptureDelete

Declaration

```
long mpiCaptureDelete(MPICapture  capture)
```

Required Header

stdmpi.h

Description

CaptureDelete deletes a Capture object and invalidates its handle (***capture***). *CaptureDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	if *CaptureDelete* successfully deletes the Capture object and invalidates its handle

See Also

mpiCaptureCreate | mpiCaptureValidate

mpiCaptureValidate

Declaration

```
long mpiCaptureValidate(MPICapture  capture)
```

Required Header stdmpi.h

Description

CaptureValidate validates the Capture object and its handle. CaptureValidate should be called immediately after an object is created.

capture	a handle to a capture object

Return Values

MPIMessageOK	if Capture is a handle to a valid object.

See Also mpiCaptureCreate | mpiCaptureDelete

mpiCaptureConfigGet

Declaration

```
long mpiCaptureConfigGet(MPICapture        capture,
                         MPICaptureConfig *config,
                         void             *external)
```

Required Header

stdmpi.h

Description

CaptureConfigGet gets a Capture object's (***capture***) 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 a Capture object's configuration information in ***external*** is *in addition* to the Capture object's configuration information in ***config***, i.e, the Capture object's configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

If a capture is in an unknown configuration (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet(...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

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

Return Values

MPIMessageOK	if CaptureConfigGet successfully writes the Capture object's configuration to the structure(s)

See Also

mpiCaptureConfigSet | mpiCaptureConfigReset

mpiCaptureConfigSet

Declaration

```
long mpiCaptureConfigSet(MPICapture        capture,
                         MPICaptureConfig  *config,
                         void              *external)
```

Required Header

stdmpi.h

Description

CaptureConfigSet sets a Capture object's (***capture***) 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 Capture object's configuration information in ***external*** is *in addition* to the Capture object's configuration information in ***config***, i.e, the Capture object's configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

If a capture is in an unknown configuration (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet(...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

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

Return Values

MPIMessageOK	if *CaptureConfigSet* successfully sets the Capture object's configuration using data from the structure(s)

See Also

mpiCaptureConfigGet | mpiCaptureConfigReset

mpiCaptureStatus

Declaration

```
long mpiCaptureStatus(MPICapture        capture,
                      MPICaptureStatus  *status,
                      void              *external)
```

Required Header

stdmpi.h

Description

CaptureStatus writes a Capture object's (***capture***) status into the structure pointed to by ***status***, and also into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

external is reserved for future functionality and should be set to NULL.

capture	a handle to a Capture object
***status**	a pointer to MPIStatus structure
***external**	a pointer to an implementation-specific structure

XMP Only	***external*** should always be set to NULL.

Return Values

MPIMessageOK	if *CaptureStatus* successfully writes the status of a Capture object to the structure(s)
MPIMessageARG_INVALID	if the *status* pointer is NULL.

See Also

mpiCaptureConfigReset

Declaration

```
long mpiCaptureConfigReset(MPICapture   capture);
```

Required Header

stdmpi.h

Description

CaptureConfigReset return the capture object to its unmapped state.

A capture object has no assumed resources, and is unmapped under default conditions. When a capture is first created, its captureMotorNumber and feedbackMotorNumber are unmapped. Once a capture has been configured, the next time that the capture object is created, it will retain the captureMotorNumber and feedbackMotorNumber that was previously assigned. mpiCaptureConfigReset(...) will return the capture object to its unmapped state.

If a capture is in an unknown configuration (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet(...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

capture	a handle to a Capture object

See Also

mpiCaptureConfigGet | mpiCaptureConfigSet | MPICaptureConfig

mpiCaptureArm

Declaration

```
long mpiCaptureArm(MPICapture  capture,
                   long        arm)   /* TRUE/FALSE */
```

Required Header

stdmpi.h

Description

CaptureArm arms or disarms ***capture***.

Value of "arm"	***Action of mpiCaptureArm***
FALSE	Disarms ***capture*** and sets the state of ***capture*** to MPICaptureStateIDLE
TRUE	Arms ***capture*** and sets the state of ***capture*** to MPICaptureStateARMED

Return Values

MPIMessageOK	if the Capture object is successfully armed or disarmed

See Also

MPICaptureState

mpiCaptureMemory

Declaration

```
long mpiCaptureMemory(MPICapture  capture,
                      void        **memory)
```

Required Header

stdmpi.h

Description

CaptureMemory writes an address [which is used to access a Capture object's (***capture***) memory] to the contents of ***memory***. This address, or an address calculated from it, can be passed as the src parameter to mpiCaptureMemoryGet(...) and as the ***dst*** parameter to mpiCaptureMemorySet(...).

Return Values

MPIMessageOK	if *CaptureMemory* successfully writes the Capture object's memory address to the contents of ***memory***

See Also

mpiCaptureMemoryGet | mpiCaptureMemorySet

mpiCaptureMemoryGet

Declaration

```
long mpiCaptureMemoryGet(MPICapture   capture,
                         void         *dst,
                         void         *src,
                         long         count)
```

Required Header

stdmpi.h

Description

CaptureMemoryGet copies ***count*** bytes of a Capture object’s (***capture***) memory (starting at address ***src***) and writes them into application memory (starting at address ***dst***).

Return Values

MPIMessageOK	if *CaptureMemoryGet* successfully copies data from Capture memory to application memory

See Also

mpiCaptureMemory | mpiCaptureMemorySet

mpiCaptureMemorySet

Declaration

```
long mpiCaptureMemorySet(MPICapture capture,
                         void       *dst,
                         void       *src,
                         long       count)
```

Required Header

stdmpi.h

Description

CaptureMemorySet copies count bytes of application memory (starting at address ***src***) and writes them into a Capture object's (***capture***) memory (starting at address ***dst***).

Return Values

MPIMessageOK	if *CaptureMemorySet* successfully copies count bytes of application memory to Capture memory

See Also

mpiCaptureMemory | mpiCaptureMemoryGet

mpiCaptureNumber

Declaration

```
long mpiCaptureNumber(MPICapture    capture,
                      long          *number)
```

Required Header

stdmpi.h

Description

CaptureNumber reads the index of the capture block associated with the capture object and writes it into the contents of a long pointed to by encoder.

capture	a handle to a capture object
***number**	pointer to the capture number.

Return Values

MPIMessageOK	if *CaptureNumber* successfully writes the index of a Capture object to the contents of **number**

See Also

mpiCaptureCreate

MPICaptureConfig

MPICaptureConfig

```
typedef struct MPICaptureConfig {
    MPICaptureTrigger         source[MPICaptureSourceCOUNT];
                                   /* use MPICaptureSource to index */
    MPICaptureEdge            edge;
    MPICaptureTriggerGlobal   global;
    MPICaptureType            type;
    long                      captureMotorNumber;
    long                      feedbackMotorNumber; /* the same as
                                  captureMotorNumber for POSITION capture */
    MPIMotorEncoder           encoder;
    long                      captureIndex;    /* 0,1,... */
} MPICaptureConfig;
```

Description

source[MPICaptureSourceCOUNT]	An array of capture trigger source inputs. The capture can be configured to trigger from one or more sources. See MPICaptureTrigger and MPICaptureSourceCOUNT.
edge	An enumerated index to the trigger edge type. The capture can be configured to trigger from a variety of logic. See MPICaptureEdge.
global	A structure to configure the global capture, to chain capture block triggering. See MPICaptureTriggerGlobal.
type	Specifies either postion-based or time-based capture. Use MPICaptureTypePOSITION for position-based capture and MPICaptureTypeTIME for time-based capture.
captureMotorNumber	The number of the motor whose "source" (MPICaptureTrigger) is used to capture position.
feedbackMotorNumber	The number of the motor whose position is being returned from the capture event. (It must be the same as captureMotorNumber for position capture).
encoder	Specifies the encoder feedback being captured.

captureIndex	A zero-based index that specifies which capture resource on an axis is to be associated with the capture object. Each axis on a node has a given number of captures associated with it. An axis may have up to 4 capture resources on it. At present, no vendor provides a node with more than one capture resource, therefore, **captureIndex must be set to zero.**

See Also

MPICaptureType

MPICaptureEdge

MPICaptureEdge

```
typedef enum MPICaptureEdge {
    MPICaptureEdgeNONE,
    MPICaptureEdgeRISING,
    MPICaptureEdgeFALLING,
    MPICaptureEdgeEITHER,
} MPICaptureEdge;
```

Description

CaptureEdge is an enumeration of input trigger edge logic for a capture.

MPICaptureEdgeRISING	Triggers on a 0 to 1 transition.
MPICaptureEdgeFALLING	Triggers on a 1 to 0 transition.
MPICaptureEdgeEITHER	Triggers on either 0 to 1 or 1 to 0 transitions.

See Also

MPICaptureTrigger

MPICaptureMessage / MEICaptureMessage

MPICaptureMessage

```
typedef enum {
    MPICaptureMessageMOTOR_INVALID,
    MPICaptureMessageCAPTURE_TYPE_INVALID,
    MPICaptureMessageCAPTURE_INVALID,
    MPICaptureMessageENCODER_INVALID,

} MPICaptureMessage;
```

Description

CaptureMessage is an enumeration of Capture error messages that can be returned by the MPI library.

MEICaptureMessageMOTOR_INVALID

mpiCaptureConfigSet(...) --> config.captureMotorNumber is not valid. It's either greater than maxMotors or = = MPICaptureNOT.MAPPED.

MEICaptureMessageCAPTURE_TYPE_INVALID

mpiCaptureConfigSet(...) --> config.Type = = MPICaptureNOT.MAPPED.

MPICaptureMessageCAPTURE_INVALID

The capture number is out of range. This message code is returned by mpiCaptureCreate(…) if the capture number is less than zero or greater than or equal to MEIXmpMaxCapturesPerMotor.

MPICaptureMessageENCODER_INVALID

The encoder index is out of range. This message code is returned by mpiCaptureCreate(…) if the encoder index is less than MPIMotorEncoderFIRST or greater than or equal to MPIMotorEncoderLAST.

See Also

mpiCaptureCreate | mpiControlConfigSet

MEICaptureMessage

```
typedef enum {
    MEICaptureMessageINVALID_EDGE,
    MEICaptureMessageGLOBAL_CONFIG_ERR,
    MEICaptureMessageGLOBAL_ALREADY_ENABLED,
    MEICaptureMessageCAPTURE_NOT_ENABLED,
    MEICaptureMessageCAPTURE_STATE_INVALID,
    MEICaptureMessageNOT_MAPPED,
    MEICaptureMessageUNSUPPORTED_PRIMARY,
    MEICaptureMessageUNSUPPORTED_SECONDARY,
    MEICaptureMessageSECONDARY_INDEX_INVALID,
} MEICaptureMessage;
```

Description

MEICaptureMessageINVALID_EDGE

The encoder edge trigger type is not valid. This message code is returned by mpiCaptureConfigSet(…) if the encoder capture edge type is not a member of the MPICaptureEdge enumeration.

MEICaptureMessageGLOBAL_CONFIG_ERR

The global trigger configuration is not valid. This message code is returned by mpiCaptureConfigSet(…) if the capture's trigger source is set to global and the capture's global trigger is enabled simultaneously. To correct this problem, either set the capture's trigger source to global or enable the capture's global trigger (not both).

MEICaptureMessage_GLOBAL_ALREADY_ENABLED

The global trigger is already enabled. This message code is returned by mpiCaptureConfigSet(…) if a global trigger is already enabled on another capture on the same node. Only one global trigger enable is allowed per node. To prevent this problem, do not enable a second global trigger on a single node.

MEICaptureMessageCAPTURE_NOT_ENABLED

This value is returned by mpiCatureCreate(...) when the capture number specified is greater than the number of captures enabled in firmware. See MPIControlConfig.

MEICaptureMessageCAPTURE_STATE_INVALID

This value is returned by mpiCaptureStatus(...) when the communication between the controller and the capture logic on the node fails resulting in an invalid capture state. See MPICaptureState.

MEICaptureMessageNOT_MAPPED

The capture object's hardware resource is not available. This message code is returned by mpiCaptureCreate(…) if the node hardware for the specified motor and encoder is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor and capture objects. A capture object cannot be created if there is no mapped hardware to support it. To correct this problem, verify that all expected nodes were found. Use meiSynqNetInfo(…) and meiSqNodeInfo(…) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEICaptureMessageUNSUPPORTED_PRIMARY

The capture hardware does not support the primary encoder. This message code is returned by mpiCaptureCreate(…) if the node hardware's primary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageUNSUPPORTED_SECONDARY

The capture hardware does not support the secondary encoder. This message code is returned by mpiCaptureCreate(…) if the node hardware's secondary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageSECONDARY_INDEX_INVALID

This message is returned from MPICaptureConfigSet(...) when the secondary encoder's index is specified as a trigger source in conjunction with other capture sources.

See Also

mpiCaptureCreate

MPICaptureSource

MPICaptureSource

```
typedef enum MPICaptureSource {
    MPICaptureSourceMOTOR_IO_0,
    MPICaptureSourceMOTOR_IO_1,
    MPICaptureSourceMOTOR_IO_2,
    MPICaptureSourceMOTOR_IO_3,
    MPICaptureSourceMOTOR_IO_4,
    MPICaptureSourceMOTOR_IO_5,
    MPICaptureSourceMOTOR_IO_6,
    MPICaptureSourceMOTOR_IO_7,
    MPICaptureSourceHOME,
    MPICaptureSourceINDEX,
    MPICaptureSourceLIMIT_HW_NEG,
    MPICaptureSourceLIMIT_HW_POS,
    MPICaptureSourceGLOBAL,
    MPICaptureSourceINDEX_SECONDARY,
    MPICaptureSourceCOUNT,
} MPICaptureSource;
```

Description

CaptureSource is an enumeration of input trigger sources for a capture.

When using one of the MPICaptureSourceMOTOR_IO values in MPICaptureSource, you can determine which MPICaptureSourceMOTOR_IO to use by referencing the appropriate node module. Look in *Node*MotorIoConfig (replacing *Node* with your node name) in the appropriate node module. Add the appropriate *Node*MotorIoConfig value to MPICaptureSourceMOTOR_IO_0.

Example: RMB-10V
Let's say you are using an MEI RMB-10V and want to find the trigger for XCVR_C.

Look in RMBMotorIoConfig in *mei_rmb.h*. You will find that the appropriate value for XCVR_C is RMBMotorIoConfigXCVR_C. RMBMotorIoConfigXCVR_C is the third value in RMBMotorIoConfig. This means that the value to use in MPICaptureSource is MPICaptureSourceMOTOR_IO_2 (the third MPICaptureSourceMOTOR_IO value).

A better way of making this conversion in your program is to add the MPICaptureSourceMOTOR_IO_0 to the nodeMotorIoConfig value you want to use. In the above example, it would be (MPICaptureSourceMOTOR_IO_0 + RMBMotorIoConfigXCVR_C).

Example: Trust TA800
To trigger off of hall A on a Trust TA800 node, you would use (MPICaptureSourceMOTOR_IO_0 + TA800MotorIoConfigHALL_A). Remember that you will need to look in *trust_ta800.h* (the node module) to find TA800MotorIoConfigHALL_A.

MPICaptureSourceMOTOR_IO_0	a capture trigger source is the 0 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_1	a capture trigger source is the 1 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_2	a capture trigger source is the 2 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_3	a capture trigger source is the 3 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_4	a capture trigger source is the 4 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_5	a capture trigger source is the 5 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_6	a capture trigger source is the 6 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_7	a capture trigger source is the 7 bit in the motor's configurable I/O.
MPICaptureSourceHOME	a capture trigger source is the HOME input in the dedicated I/O input.
MPICaptureSourceINDEX	a capture trigger source is the encoder INDEX input in the dedicated I/O input.
MPICaptureSourceLIMIT_HW_NEG	a capture trigger source is the Hardware Negative Limit input in the dedicated I/O input.
MPICaptureSourceLIMIT_HW_POS	a capture trigger source is the Hardware Positive Limit input in the dedicated IO word. Please see MPIMotorDedicatedIn.
MPICaptureSourceGLOBAL	a capture trigger source is the Global capture signal found on the node. Please see MPICaptureTriggerGlobal.
MPICaptureSourceINDEX_SECONDARY	A a capture trigger source is the index on the secondary encoder. If position based capture is selected with the feedback source being the secondary encoder, this is the only valid capture source.
MPICaptureSourceCOUNT	Total number of possible input sources for a capture.

See Also

MPICaptureTrigger | MEIMotorIoMask

MPICaptureState

MPICaptureState

```
typedef enum {
    MPICaptureStateIDLE,
    MPICaptureStateARMED,
    MPICaptureStateCAPTURED,
    MPICaptureStateCLEAR,
} MPICaptureState;
```

Description

MPICaptureStateIDLE	Capture is not armed. This is the default state.
MPICaptureStateARMED	Capture is armed, but has not triggered yet.
MPICaptureStateCAPTURED	Capture triggered and position data is valid.
MPICaptureStateCLEAR	Capture is not armed, but has not transitioned to the IDLE state yet. This is an internal transitional state between CAPTURED and IDLE. It occurs when a capture is disarmed.

See Also

MPICaptureStatus

MPICaptureStatus

MPICaptureStatus

```
typedef struct MPICaptureStatus {
    MPICaptureState     state;
    double             latchedValue;
} MPICaptureStatus;
```

Description

state	An enumerated value representing the present state of the capture logic
latchedValue	The captured position value. This value is only valid when the state is CAPTURED.

See Also

MPICaptureState

MPICaptureTrigger

MPICaptureTrigger

```
typedef struct MPICaptureTrigger {
    long enabled;   /* TRUE/FALSE */
    long invert;    /* TRUE = invert, FALSE = normal */
} MPICaptureTrigger;
```

Description

The **CaptureTrigger** structure specifies the trigger configurations for a capture.

enabled	Enables or disables the trigger. A value of TRUE enables the trigger, FALSE disables the trigger.
invert	Normal or inverted trigger polarity. A value of FALSE indicates normal polarity, TRUE indicates inverted polarity.

See Also

MPICaptureSource

MPICaptureTriggerGlobal

MPICaptureTriggerGlobal

```
typedef struct MPICaptureTriggerGlobal {
    long    enabled;    /* TRUE/FALSE */
} MPICaptureTriggerGlobal;
```

Description

The **CaptureTriggerGlobal** structure specifies the global input trigger configuration for a capture.

enabled	Enables or disables the global input trigger. A value of TRUE enables the trigger, FALSE disables the trigger.

See Also

MPICaptureConfig

MPICaptureType

MPICaptureType

```
typedef enum  {
    MPICaptureTypePOSITION,
    MPICaptureTypeTIME,
} MPICaptureType;
```

Description

MPICaptureTypePOSITION	An actual position is captured by the Node from its feedback source.
MPICaptureTypeTIME	An internal timer is captured by the node and then a captured position is interpolated by the XMP firmware.

See Also

MPICaptureNOT_MAPPED

Declaration

```
#define MPICaptureNOT_MAPPED (-1)
```

Required Header

stdmpi.h

Description

Capture objects are associated with the controller and are not mapped to any hardware resources under default conditions. MPICaptureNOT_MAPPED will be assigned to:

```
long    captureMotorNumber;
long    feedbackMotorNumber;
```

when mpiCaptureConfigGet() is called for the first time on a capture object. After a capture object has been used once, the resource mapping will remain in place until it is reassigned.

See Also