Copyright ©1995 by NeXT Computer, Inc. All Rights Reserved.

MIDIAwaitReply(), MIDIHandleReply()

SUMMARY Handle replies from the MIDI driver to an application

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDIAwaitReply(**port_t *reply_port*, MIDIReplyFunctions **handlers*, int *timeout***)**
kern_return_t **MIDIHandleReply(**msg_header_t **msg*, MIDIReplyFunctions **handlers***)**

DESCRIPTION **MIDIAwaitReply()** receives and handles a message from the MIDI driver. *reply_port* is the port set used to receive messages. *handlers* is a MIDIReplyFunctions structure containing pointers to functions for handling replies (see ªTypes and Constantsº for a description of the MIDIReplyFunctions structure). *timeout* represents the amount of time, in milliseconds, the **MIDIAwaitReply()** function will wait before returning if no message is in the MIDI driver's queue. After receiving the message from the MIDI driver as specified, **MIDIAwaitReply()** calls **MIDIHandleReply()**.

MIDIHandleReply() accepts a message received from the MIDI driver and passes it to the appropriate handling function. *msg* is the Mach message received from the MIDI driver on the application's port set. *handlers* is a **MIDIReplyFunctions** structure containing pointers to functions for handling replies.

Before calling one of these functions, you register requests with the MIDI driver by calling one or more of the functions **MIDIRequestData()**, **MIDIRequestAlarm()**, **MIDIRequestExceptions()**, and **MIDIRequestQueueNotification()**. The *handlers* passed in the reply handling functions should include a function for handling each of the responses requested; the *reply_port* set passed to **MIDIAwaitReply()** should include a port for handling each of the request types.

One common use of these functions is to receive MIDI data. The application calls **MIDIRequestData()**, then repeatedly calls one of these reply handling functions in a loop. To do so in an Application Kit application, you must run **MIDIAwaitReply()** in a separate Mach thread. Alternatively, you may register the port set with the **DPSAddPort()** function, use the Mach function **msg_receive()** to receive the response from the MIDI driver, then handle the message with **MIDIHandleReply()**.

RETURN Both functions return KERN_SUCCESS if they successfully handle the reply. If unsuccessful, they return an exception code indicating the reason they couldn't handle the reply.

MIDIBecomeOwner(), MIDIReleaseOwnership()

SUMMARY Acquire and release ownership of the MIDI driver

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDIBecomeOwner(**port_t *driverPort*, port_t *ownerPort***)**
kern_return_t **MIDIReleaseOwnership(**port_t *driverPort*, port_t *ownerPort***)**

DESCRIPTION **MIDIBecomeOwner()** makes the sending process the owner of the MIDI driver. Before becoming owner of the MIDI driver, an application must look up *driverPort* with a call to the Mach **netname_look_up()** function. It must also allocate, using the Mach **port_allocate()** function, an *ownerPort* to identify it to the MIDI driver in other function calls.

MIDIReleaseOwnership() releases the MIDI driver port from the control of the sending application.

RETURN Both functions return KERN_SUCCESS if they complete successfully, and MIDI_ERROR_BUSY if another process is using the driver.

MIDIClaimUnit(), MIDIReleaseUnit()

SUMMARY Claim and release ownership of serial ports for MIDI driver clients

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDIClaimUnit(**port_t *driverPort*, port_t *ownerPort*, short *unit***)**
kern_return_t **MIDIReleaseUnit(**port_t *driverPort*, port_t *ownerPort*, short *unit***)**

DESCRIPTION These functions control the access of a MIDI driver client application to the host computer's serial ports.

MIDIClaimUnit() is used to acquire a serial port for MIDI communication. It is called after the MIDI driver has been acquired by the application with the **MIDIBecomeOwner()** function. *driverPort* is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**. *unit* may be one of the symbolic constants MIDI_PORT_A_UNIT and MIDI_PORT_B_UNIT, defined in the header file **mididriver/midi_driver.h**.

MIDIReleaseUnit() is used to release the serial port used in MIDI communication.

RETURN **MIDIClaimUnit()** returns KERN_SUCCESS if it successfully acquires the serial port as requested. **MIDIReleaseUnit()** returns KERN_SUCCESS if it successfully releases the serial port as requested. Both return MIDI_ERROR_NOT_OWNER if the sending process hasn't acquired the MIDI driver and MIDI_ERROR_UNIT_UNAVAILABLE if the specified serial port is busy.

MIDIClearQueue(), MIDIFlushQueue(), MIDIGetAvailableQueueSize()

SUMMARY Manage the MIDI driver queue

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDIClearQueue(**port_t *driverPort*, port_t *ownerPort*, short *unit***)**
kern_return_t **MIDIFlushQueue(**port_t *device_port*, port_name_t *ownerPort_port*, short *unit***)**
kern_return_t **MIDIGetAvailableQueueSize(**port_t *driverPort*, port_t *ownerPort*, short *unit*, int **theSize***)**

DESCRIPTION These functions allow an application to manage the queue in the MIDI driver. *driverPort* is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**.

MIDIClearQueue() empties the specified queue without sending any remaining data.

MIDIFlushQueue() returns after sending the data remaining in the queue immediately, bypassing the normal time scheduling mechanism.

MIDIGetAvailableQueueSize() returns, by reference in *theSize*, the amount of space currently

available in the queue.

RETURN Each of these functions returns KERN_SUCCESS if the specified operation is performed successfully. Otherwise, they return an error code indicating why the operation wasn't completed.

MIDIFlushQueue() → See MIDIClearQueue()

MIDIGetAvailableQueueSize() → See MIDIClearQueue()

MIDIGetClockTime() → See MIDISetClockMode()

MIDIGetMTCTime() → See MIDISetClockMode()

MIDIHandleReply() → See MIDIAwaitReply()

MIDIReleaseOwnership() → See MIDIBecomeOwner()

MIDIReleaseUnit() → See MIDIClaimUnit()

MIDIRequestAlarm(), MIDIRequestData(), MIDIRequestExceptions(), MIDIRequestQueueNotification()

SUMMARY Request notification from the MIDI driver

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDIRequestData(**port_t *driverPort*, port_t *ownerPort*, short *unit*, port_t *replyPort***)**
kern_return_t **MIDIRequestAlarm(**port_t *driverPort*, port_t *ownerPort*, port_t *replyPort*, int *time***)**
kern_return_t **MIDIRequestExceptions(**port_t *driverPort*, port_t *ownerPort*, port_t *replyPort***)**
kern_return_t **MIDIRequestQueueNotification(**port_t *driverPort*, port_t *ownerPort*, short *unit*, port_t *replyPort*, int *size***)**

DESCRIPTION These functions allow an application to request notification by the MIDI driver in case of specific events.

The reply returned in response to these requests should be handled by an application's corresponding MIDIReplyFunction. For example, the MIDI driver's response to **MIDIRequestExceptions()** should be handled by an application's MIDIExceptionReplyFunction. MIDIReplyFunction types are declared in the header **mididriver/midi_driver.h** and described in the section ªTypes and Constants.º After calling one of these functions, call **MIDIAwaitReply()** or **MIDIHandleReply()** to handle the response returned by the MIDI driver.

driverPort is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**. *unit* is the serial port associated with the request. *replyPort* is the port on which the response to the request is expected to be sent. This port should be included in the port set passed to **MIDIAwaitReply()** or in the message header passed to **MIDIHandleReply()**.

In **MIDIRequestQueueNotification()**, *size* is the queue size below which notification will be sent.

RETURN These functions return KERN_SUCCESS if the specified request is registered with the MIDI driver. Otherwise, they return an error code indicating why the operation wasn't completed.

SEE ALSO **MIDIAwaitReply()**

MIDIRequestData() → See MIDIRequestAlarm()

MIDIRequestExceptions() → See MIDIRequestAlarm()

MIDIRequestQueueNotification() → See MIDIRequestAlarm()

MIDISendData()

SUMMARY Send data using the MIDI driver

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDISendData(**port_t *driverPort*, port_t *ownerPort*, short *unit***,** MIDIRawEvent ***data*, unsigned int *count***)**

DESCRIPTION This function sends data, using the MIDI driver and specified serial port to other MIDI devices. *driverPort* is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**. *data* is an array of **MIDIRawEvent** data. *count* is the number of **MIDIRawEvent** structures in the array

RETURN This function returns KERN_SUCCESS if the data is successfully written to the serial port. Otherwise, returns an error code indicating why the operation wasn't completed.

SEE ALSO **MIDIRequestData()**

MIDISetClockMode(), MIDISetClockQuantum(), MIDISetClockTime(), MIDIGetClockTime(), MIDIGetMTCTime()

SUMMARY Control the MIDI driver clock

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDISetClockMode(**port_t *driverPort*, port_t *ownerPort*, short *synchUnit*, int *mode***)**
kern_return_t **MIDISetClockQuantum(**port_t *driverPort*, port_t *ownerPort*, int *interval***)**
kern_return_t **MIDISetClockTime(**port_t *driverPort*, port_t *ownerPort*, int *time***)**
kern_return_t **MIDIGetClockTime(**port_t *driverPort*, port_t *ownerPort*, int ***time***)**
kern_return_t **MIDIGetMTCTime(**port_t *driverPort*, port_t *ownerPort*, short ****format***, short ****hours***, short ****minutes***, short ****seconds***, short ****frames***)**

DESCRIPTION These functions let you set and test parameters for the MIDI driver clock. *driverPort* is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**.

MIDISetClockMode() sets the synchronization mode of the MIDI driver clock. *synchUnit* represents the serial port on which the driver will listen for MIDI time code signals. *mode* is one of the symbolic constants MIDI_CLOCK_MODE_INTERNAL or MIDI_CLOCK_MODE_MTC_SYNC. MIDI_CLOCK_MODE_INTERNAL causes the clock to run on its own internal time, while MIDI_CLOCK_MODE_MTC_SYNC causes the clock to synchronize with the MIDI time code present on *synchUnit*.

MIDISetClockQuantum() sets the interval between clock signals. *interval* represents this quantum in microseconds. The default setting is 1000 (1 millisecond).

MIDISetClockTime() sets the current clock time. *time* is an integer representing the time to which you want to set the MIDI driver clock.

MIDIGetClockTime() returns by reference in *time* the current clock time.

MIDIGetMTCTime() returns the MIDI time code time. *format* represents the MIDI time code format of the current time. *hours, minutes*, and *seconds* represent the chronological value of the current time. *frames* represents the frame number of the current time.

RETURN These functions return KERN_SUCCESS if the operation is performed successfully. Otherwise, they return an error code indicating why the operation wasn't completed.

MIDIGetClockTime() returns, by reference in *time*, the current time.

SEE ALSO **MIDIRequestAlarm()**, **MIDIStartClock()**

MIDISetClockQuantum() → See MIDISetClockMode()

MIDISetClockTime() → See MIDISetClockMode()

MIDISetSystemIgnores()

SUMMARY Sets MIDI system codes the MIDI driver ignores

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDISetSystemIgnores(**port_t *driverPort*, port_t *ownerPort*, short *unit*, unsigned int *ignoreBits***)**

DESCRIPTION **MIDISetSystemIgnores()** sets MIDI system codes the MIDI driver ignores. *driverPort* is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**. *unit* may be one of the symbolic constants MIDI_PORT_A_UNIT or MIDI_PORT_B_UNIT (defined in the header file **mididriver/midi_driver.h**), representing the port on which MIDI system codes should be ignored. *ignoreBits* may be one of the symbolic constants MIDI_IGNORE_CLOCK, MIDI_IGNORE_START, or MIDI_IGNORE_CONTINUE (defined in **mididriver/midi_driver.h**); you may logically OR these constants for multiple settings.

RETURN This function returns KERN_SUCCESS if the operation is performed successfully. Otherwise, it returns an error code indicating why the operation wasn't completed.

MIDIStartClock(), MIDIStopClock()

SUMMARY Start and stop the MIDI clock

DECLARED IN mididriver/midi_driver.h

SYNOPSIS kern_return_t **MIDIStartClock(**port_t *driverPort*, port_t *ownerPort***)**
kern_return_t **MIDIStopClock(**port_t *driverPort*, port_t *ownerPort***)**

DESCRIPTION **MIDIStartClock()** starts the clock using the current settings. **MIDIStopClock()** stops the clock. *driverPort* is the MIDI driver port. *ownerPort* is the port allocated by the process to identify it to the MIDI driver in MIDI function calls, and first registered with the MIDI driver in **MIDIBecomeOwner()**.

RETURN These functions return KERN_SUCCESS if the operation is performed successfully. Otherwise, they return an error code indicating why the operation wasn't completed.

SEE ALSO **MIDIRequestAlarm()**, **MIDISetClockMode**

MIDIStopClock() → **See MIDIStartClock()**