10

MIDI Driver API

Library:	libsys_s.a
Header File Directory:	/NextDeveloper/Headers/mididriver

Introduction

This chapter describes NEXTSTEP's MIDI driver C functions and supporting header files for MIDI applications.   This introduction contains conceptual discussions of the MIDI interface and its implementation on NeXT computers.

The sections ªWhat Is MIDI?º and ªConnecting MIDI Devicesº provide general information on MIDI on the NeXT.   The section ªMIDI Driver Overviewº outlines how to structure the MIDI support section of an application that uses the MIDI driver functions.

What Is MIDI?

MIDI, the Musical Instrument Digital Interface, defines a software format and a hardware standard for exchanging information among electronic musical instruments (such as synthesizers, samplers, digital pianos, and guitar or wind controllers) as well as other devices (such as computers, sequencers, mixers, signal processors, and even stage lighting).   Originally designed to capture the performance gestures of a keyboard player, MIDI normally transmits keyboard-oriented information, such as which key the performer depressed and with what velocity, or which button or slider was adjusted on a synthesizer's control panel.   This sort of data is much more compact and more easily edited than the data in a digital audio recording of the same performance.   Unlike audio data, MIDI data can easily be used to control other instruments or to create a printed score (using a music notation application).

Connecting MIDI Devices

You can connect MIDI instruments to either of a NeXT computer's serial ports, using an external device known as a MIDI interface.   The instruments connect to the MIDI interface (or to each other) with standard MIDI cables, available at most music stores.   The MIDI interface adapts these cables' unidirectional 5-pin DIN connectors to the serial port's bidirectional mini-DIN connector. Any number of instruments can be connected to a serial port through the interface, and the two ports can be used simultaneously by a single application.   A single serial port can receive and transmit MIDI data at the same time.

The musical instrument must be set up correctly for MIDI communication to work as expected. Because MIDI is a unidirectional protocol, there's no means for an application to verify that the external device is receiving the MIDI data that the application sends. Thus the user is responsible for ensuring that the configuration is correct. For instructions on setting up the MIDI device, see the owner's manual for that device.

In particular, note that most MIDI commands are sent on specific ªchannels.º Unlike the left and right channels of analog audio signals, MIDI channels don't use separate cables, but instead are encoded in the MIDI data itself. The sixteen MIDI channels are used for sending separate streams of commands to different synthesizers on a single MIDI network, or to the distinct sound-generating units within a single multi-timbral synthesizer. There's no MIDI command that asks a device to start using a certain MIDI channel. Instead, the user must manually set the MIDI device to transmit and receive on the channels expected by the software. A typical default is to transmit and receive on channel 1.

The NeXT MIDI Device Driver

The MIDI driver is a loadable Mach device driver that controls the flow of MIDI data to and from the serial ports. The MIDI device driver API contains C functions for direct control of the MIDI driver, giving you control over the buffering and timing of MIDI data. The functions also provide other featuresÐfor example, you can manage the size of the MIDI data queue, manipulate the driver's timer, and filter out a few more kinds of MIDI commandsÐbut you'll rarely need these features.

The rest of this document contains information that's useful for programming with the MIDI driver C functions. The sample C programs in **/NextDeveloper/Examples/SoundAndMusic/Drivers/MidiDriver** illustrate some of the functions documented here. Information can also be gleaned from the header files in **NextDeveloper/Headers/mididriver**.

The MIDI Data Format

If you use the MIDI driver functions, you'll be examining MIDI data as hexadecimal values, so you'll need to understand the MIDI data format. Read this section for a synopsis of the data format, if you're not already familiar with the MIDI specification.

MIDI data consists of commands sent in an asynchronous serial stream at 31.25 kBaud. The data is transmitted in ten-bit bytes, but the first and last bits of each byte are start and stop bits, added by the transmitting device and stripped off by the receiving device. Thus, MIDI commands are considered to consist of eight-bit bytes. A typical MIDI command contains:

· One Status byte (whose most significant bit is set to 1). The Status byte defines a type of command, such as Note On or Pitch Bend.

· Zero, one, or two Data bytes (each having its most significant bit set to 0). Data bytes contain values applied by the command, such as ªkey numberº and ªvelocity,º or ªamount of pitch bend.º The type of command, specified by the preceding Status byte, determines how many Data bytes are expected.

There are two exceptions to the above pattern:

· The Status byte may be omitted, in which case the type of command is given by the most recent Status byte. This condition is called Running Status.

· The Status byte F0 (hexadecimal) is the special System Exclusive command, which is followed by a Data byte identifying a particular manufacturer, and any number of subsequent Data bytes whose meaning the manufacturer is free to determine. Only that manufacturer's instruments are expected to respond to the System Exclusive command.

Status bytes with hexadecimal values from 80 to EF are ªchannel commands.º These MIDI commands are sent on specific MIDI channels, as determined by the rightmost four bits of the Status byte. Most MIDI devices can be configured to respond only to certain channels, making it possible for a single MIDI data stream to deliver different musical information to different devices simultaneously.

Note that although MIDI bytes are classified as Status bytes or Data bytes, the term ªMIDI dataº refers generically to everything in a stream of MIDI commands, both Status bytes and Data bytes.

The file **mididriver/midi_spec.h** includes a list of Status bytes and other standard MIDI definitions. You can obtain the complete MIDI specification from the International MIDI Association at 11857 Hartsook St., North Hollywood, CA 91607, U.S.A. For an introduction to the MIDI specification, including a summary of commands, see Gareth Loy's article ªMusicians Make a Standard: The MIDI Phenomenonº in *Computer Music Journal* Vol. 9, No. 4 (Winter 1985).

MIDI Driver Overview

The MIDI driver is a loadable server residing within the Mach kernel. (For more on loadable servers, see *NEXTSTEP Operating System Software*.) For each serial port, the MIDI driver maintains an input queue (containing MIDI data received from external instruments) and an output queue (for data received from an application). The MIDI driver C functions let you retrieve data from the input queue, place data in the output queue, and perform numerous other operations.

Instead of using a direct message-passing mechanism for forwarding received MIDI data, the driver uses a request/reply interface. This means that data received from a serial port is queued within the driver until the application requests the data. Then the driver asynchronously sends Mach messages containing all the MIDI data that it's received since the last time the application requested data. The application must supply functions that perform the actual work of manipulating the incoming MIDI data in whatever manner is desired. The reply handler acts as a dispatcher by examining each incoming Mach message and routing it in a suitable format to the appropriate one of these application-supplied functions. When the application is ready for the next set of MIDI data, it must make another request for data from the driver.

Output is managed similarly. In addition to the asynchronous messages that contain incoming MIDI data, the driver sends the application a message whenever the output queue has space available for more outgoing data. The reply handler passes these notifications to another application-supplied function, which typically responds by sending more data to the driver.

A stream of MIDI bytes coming in real time from an external instrument doesn't necessarily contain any information about *when* each MIDI command was received. However, to make musical sense of recorded MIDI data, timing information is essential. Thus the driver always timestamps MIDI commands on input. A timer service, included with the driver, serves this purpose. It also schedules each outgoing MIDI command. Additionally, an application can ask this timer service to notify it at a certain time, and the application can stop and restart the timerÐor even make it run backwards. The MIDI library has a separate reply handler for messages from the timer service, analogous to the reply handler that manages MIDI input and output.