Flypaper

by Steve Berkley
Version 1.0.0
(demo)

Table Of Contents

DISCLAIMER

BIAS makes no warranties, expressed or implied about the usability of the provided software, including the implied warranties of merchantability, fitness for a particular purpose and noninfringement. No oral or written information or advice given by BIAS, its employees, distributors, dealers or agents shall create any new warranties.

CUSTOMER SUPPORT

Owners of registered copies of Flypaper may obtain on-line or phone assistance with the software, Monday through Friday, 9:00AM to 5:00PM PST.

email: sberkley@crl.com
Phone: 415-331-2446

1. Welcome!

Welcome to Flypaper! Flypaper is a flexible MIDI software tool that will allow you to :

- Examine MIDI from hardware devices and other software applications
- Provide a set of flexible MIDI definitions to assist developers in the creation of new MIDI software and hardware.
- Store and send MIDI messages (including SYSEX dumps) limited only by the amount of available RAM.
- Route and modify MIDI messages, map MIDI to different sources
  - ° Transpose incoming signals
  - ° Map controllers to other controllers on different devices
- Allow for sophisticated notification options, which include
  - ° Alerts, Beeps, Graphic Sliders and Keyboards
  - ° Issue Apple Events to other applications upon receipt of a MIDI cue
- Automate complex MIDI tasks through Apple Events communications with Flypaper
  - ° Create an editor for your MIDI device with an application like FileMaker Pro™ or HyperCard™ by sending Apple Events to Flypaper.
- Powerful searching options to locate specific MIDI bytes or definitions.
- Copy logs to the Clipboard, so you may format the MIDI log in a word processor and produce hard copies.

This manual is intended to serve as a reference source and learning aid in using the Flypaper software. It may also be useful as supplemental material for courses and lectures introducing MIDI to music technology students. If you are new to MIDI or the Macintosh, refer to **Appendix A: Resources for Beginners**.

Demo Version Limitations

If you have obtained a demonstration version of Flypaper, please note that it has the following features disabled, which appear in fully registered copies of Flypaper:

- **File** menu lacks Printing option.
- Limited usage (by date)
- **File** menu "Save" and "Save As" features disabled
- Sessions limited to 20 minutes.

You can obtain a fully registered copy of Flypaper by filling out the included Order Form, and sending it with your check or money order to BIAS:

BIAS
202 Donahue Street
Sausalito, CA 94965

1-415-331-2446

System Requirements

To install Flypaper, you will need:

- A Macintosh computer with at least 1,500k of RAM available, and at least 500k available on a hard drive.
- System 7 or greater
- Color monitor (optional)
- MIDI interface and hardware (optional)
- Apple's Midi Manager 2.0.1 or later installed on your machine.

Installation

To install Flypaper, simply double click the icon for Flypaper. You will be asked where you wish to install the software. Select the hard drive you wish to install Flypaper onto, then press the "Extract" button. A folder will be created on your hard drive that contains:

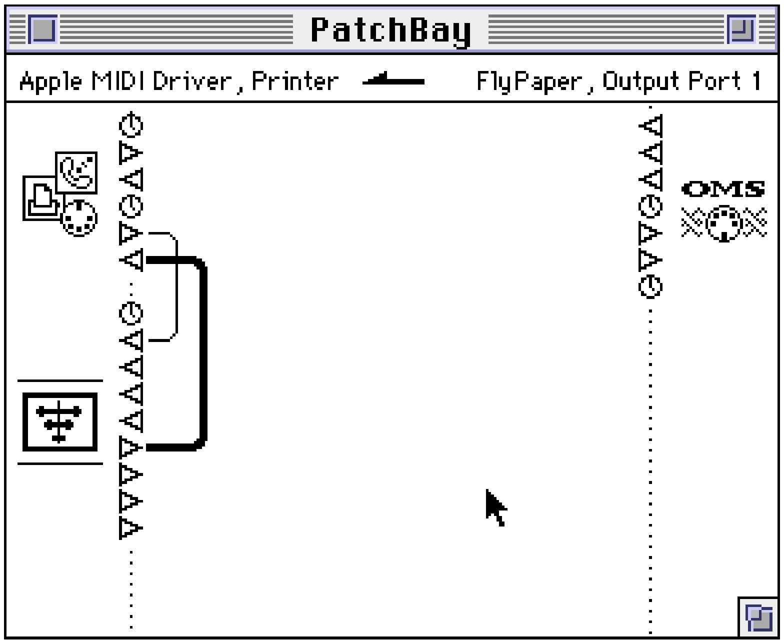
- The Flypaper application
- The Flypaper Transportable Updater application
- Flypaper Tutorial documents
- Any "Read Me" TeachText files that contain up-to-date information regarding installation and operation of Flypaper.

Patching In

After you open up the Flypaper application, you must make sure you connect lines of MIDI information in and/or out of the Flypaper application using the Patchbay desk accessory or application that came with your Apple MIDI Manager installation. Flypaper does not yet support Opcode System's OMS (Open MIDI System).

If you double-clicked the Flypaper application while Appletalk was on, you may have received a message indicating "The Printer Port is in Use...". If this happens and you would like to use your Printer Port for MIDI messages, first Quit the Flypaper application, then open the Chooser and turn Appletalk OFF, and finally double-click on the Flypaper application to start it again. It may be necessary to double-click on your MIDI driver icon in the Patchbay window to reactivate the Printer port.

Open up the Patchbay application and connect incoming MIDI information into Flypaper's "inputs" and outgoing information from Flypaper's "outputs" to some device or application that appears in the Patchbay window:

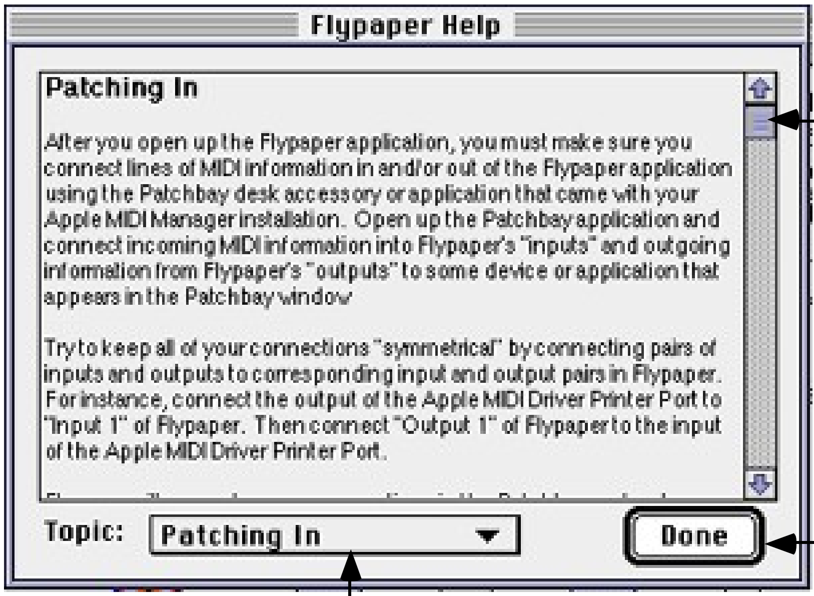


Try to keep all of your connections "symmetrical" by connecting pairs of inputs and outputs to corresponding input and output pairs in Flypaper. For example, connect the output of the Apple MIDI Driver Printer Port to "Input 1" of Flypaper. Then connect "Output 1" of Flypaper to the input of the Apple MIDI Driver Printer Port.

Flypaper will remember your connections in the Patchbay and restore them the next time you launch the Flypaper application.

Online Help

While you are using Flypaper, you can use two types of online help. First Balloon Help may be activated by pulling down the **Balloon Help** menu and selecting "Show Balloons." Balloon help will show you the functions of each menu item as you move the mouse across different menu items. The second online help available in Flypaper is under the **Apple** menu's "Help" item. This help system gives you detailed information about how to use the Flypaper software.



Patching In

After you open up the Flypaper application, you must make sure you connect lines of MIDI information in and/or out of the Flypaper application using the Patchbay desk accessory or application that came with your Apple MIDI Manager installation. Open up the Patchbay application and connect incoming MIDI information into Flypaper's "inputs" and outgoing information from Flypaper's "outputs" to some device or application that appears in the Patchbay window

Try to keep all of your connections "symmetrical" by connecting pairs of inputs and outputs to corresponding input and output pairs in Flypaper. For instance, connect the output of the Apple MIDI Driver Printer Port to "Input 1" of Flypaper. Then connect "Output 1" of Flypaper to the input of the Apple MIDI Driver Printer Port.

click on the scroll bar to scroll through the Help Topic's text

Click on "Done" when you are finished receiving Help.

Choose a Help Topic category from this pop-up menu

2. Flypaper Concepts

Sessions

Flypaper records two types of information, MIDI *cues* defined by the user and the actual MIDI messages received by Flypaper via Apple's MIDI Manager. Flypaper stores its dictionary of MIDI cues and the actual MIDI messages into a *Session Document.* The MIDI messages (including timing and port information) is held in the Session Document's *log*. You'll recognize a Flypaper Session Document on the desktop by it's icon:

Tutorial 1

To open up Flypaper with a Session Document, simply double click the icon of the Session Document from the document, or use the "Open" menu item from the **File** menu in Flypaper. The list of received MIDI bytes are shown in the *log*, a scrollable window displaying MIDI messages. The MIDI cues are shown in another window, the *Session Window*.

Cues and Responses

MIDI messages are difficult to read and locate information in, so Flypaper allows you to "define" a series of MIDI messages into a collection called a *cue.* For example, MIDI Note-On messages are three bytes. An example of a Note-On message would be:

90 40 50 (hexadecimal)

The "9" of the 90 indicates that this is a Note-On message, while the "0" of the 90 tells us that the message is on Channel 1. The next two bytes indicate the key number and the key velocity. In this case, the key number is 64 (4 * 16 = 64...), which is an "E" for musicians, and the key velocity is 90. Once we know all of this information, it is easy to define a "Note On" cue in Flypaper that becomes part of the Session Document's vocabulary of MIDI messages. A cue has a name and an icon associated with it, both of which you may specify when creating or editing a Flypaper cue (see Figure 1). This makes reading your log and identifying MIDI messages much simpler, because you are no longer searching for symbolic data.

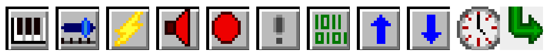


Figure 1. Flypaper icons you can use for your cues and responses

Once a MIDI cue is defined in Flypaper, you can search the log for instances of that cue, or you can set up triggered *responses* to that cue. Responses are just like cues except that they issue some action rather than identify a defined set of MIDI bytes. *The simplest way to understand cues and responses is to understand cues as "input" and responses as "output" from Flypaper.* For instance, you may wish an alert dialog to appear (a response) whenever a Sysex start message (a cue) is identified in the incoming MIDI messages.

Cues may also set or take *user variables.* Flypaper has sixteen user variables that may hold values during your Flypaper session. For instance, you may want to store MIDI controller values in a user variable so that you can pass those controller values to the slider reaction, which shows a graphic slider with some value.

Flypaper has eight cue types for you to work with. Two of the eight are cues, the other six are reactions. Below is a brief description of what each cue does. For detailed information on the format and specifics for each cue, please reference **5. Cues** later in this manual.

MIDI Cue and Reaction Cue

Holds a description of the MIDI information to search for, or the MIDI information to send out.

And Also... Cue

Causes an *additional* reaction when the preceding cue is recognized.

This cue takes no description.

Alert Reaction Cue

Brings up a dialog with some user-specified text in it when executed

Enter the text you wish to appear in the dialog into the description field of this reaction cue.

Sound Reaction Cue

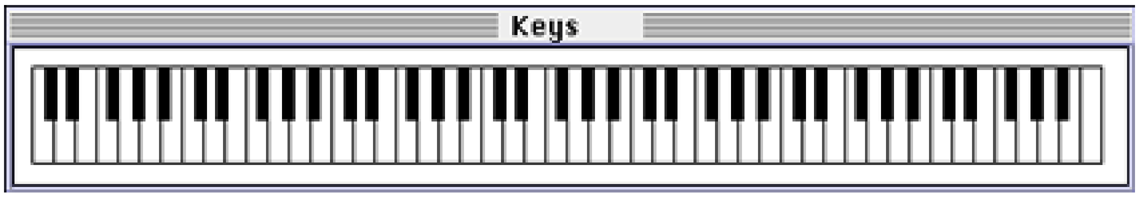
Plays the "System Beep" sound when executed

Value Display Reaction Cue

This cue displays the value of a user variable.

Keyboard Reaction Cue

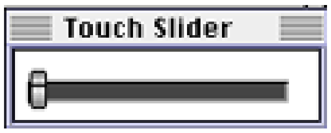
Shows information on a graphic keyboard, by taking two values: which note to highlight and a which note to unhighlight.



Slider Reaction Cue

Shows information on a graphic slider, by taking one value: the value of the slider.

Shows information on a graphic slider.



Delay Reaction Cue

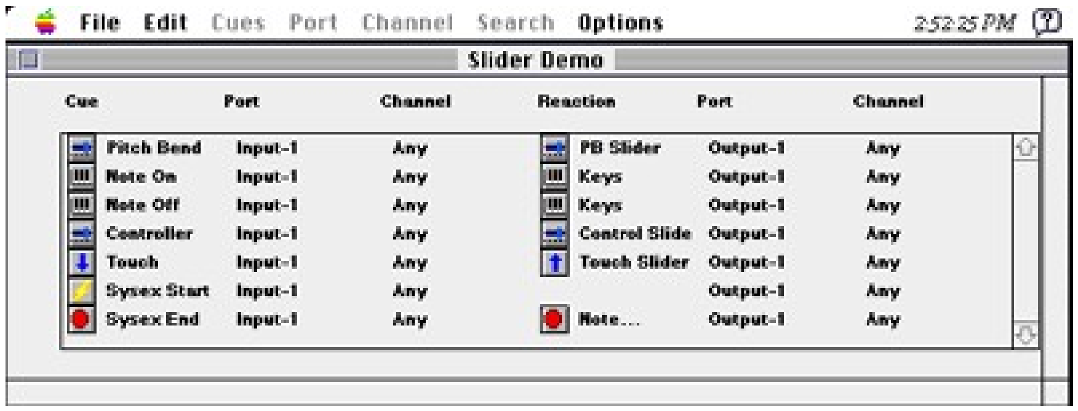
Delays for a specified count of milliseconds when executed.

Apple Event Cue

Triggers a reaction when an Apple Event is received that matches the cue's definition (an Apple Event type such as 'plno'.. you may define as many as you wish). This is a very powerful automation feature that allows you to control your MIDI hardware and Flypaper from another Macintosh application such as FileMaker Pro™ or HyperCard™. You can even control your Macintosh software with your MIDI hardware!

Session Definitions

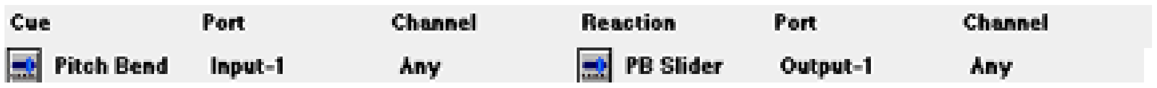
Flypaper allows you to organize a cue and a response together into a *definition.* A definition is one cue, one response, and port and channel information to identify input and output. All definitions are shown in the Flypaper document's Session Window (see Figure 2).



Cue	Port	Channel	Reaction	Port	Channel
Pitch Bend	Input-1	Any	PB Slider	Output-1	Any
Note On	Input-1	Any	Keys	Output-1	Any
Note Off	Input-1	Any	Keys	Output-1	Any
Controller	Input-1	Any	Control Slide	Output-1	Any
Touch	Input-1	Any	Touch Slider	Output-1	Any
Sysex Start	Input-1	Any		Output-1	Any
Sysex End	Input-1	Any	Note...	Output-1	Any

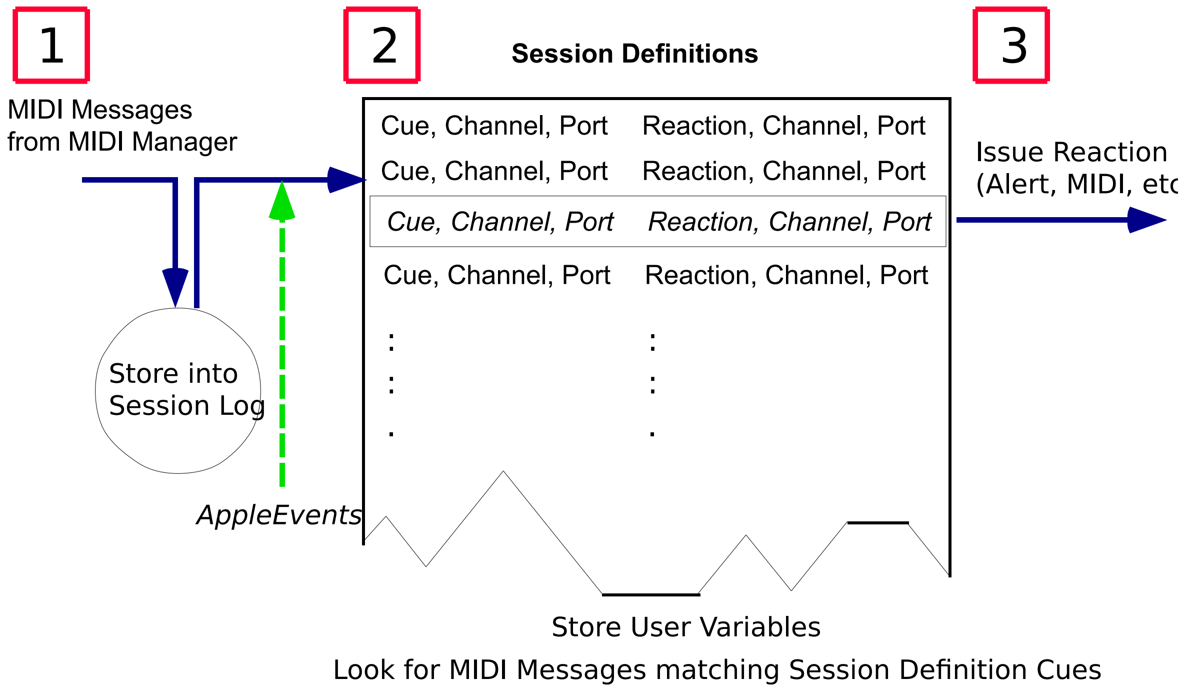
Figure 2. A Session Window, showing seven definitions.

Below is an example of a **Session Definition**. In this example, Pitch bend messages are identified by the Pitch Bend Cue. Only those messages coming in through MIDI manager input port 1, on any channel, will be identified in this definition. The reaction is to show the Pitch Bend value with a *reaction cue* called "PB Slider." Note the output port and channel are not applicable to this particular reaction cue, because no MIDI is being produced in response to receiving Pitch Bend MIDI messages.



Cue	Port	Channel	Reaction	Port	Channel
Pitch Bend	Input-1	Any	PB Slider	Output-1	Any

Here's an overview of how Flypaper works:


1
MIDI Messages from MIDI Manager
2
Session Definitions
3
Cue, Channel, Port Reaction, Channel, Port
Cue, Channel, Port Reaction, Channel, Port
Cue, Channel, Port Reaction, Channel, Port
Cue, Channel, Port Reaction, Channel, Port
Issue Reaction (Alert, MIDI, etc
Store into Session Log
AppleEvents
Store User Variables
Look for MIDI Messages matching Session Definition Cues

1 The Apple MIDI Manager passes MIDI messages into Flypaper's MIDI Manager Input Ports (which have been connected in Patchbay). All MIDI Messages are recorded into the Session's log, in RAM (except in record mute mode).

2 MIDI Messages stored into the Session Log in RAM are then checked as quickly as possible against cues in the Session Definitions. Any matches may also store values from MIDI Messages into user variables. Incoming high-level Apple Events are also checked against the cues in the Session Definitions.

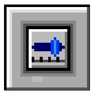
3 If a MIDI Manager message matched a Session Definition's cue, the Session Definition's reaction is invoked. This may cause an alert dialog, a sound, an output MIDI message, etc.

Flypaper's most important task is to store the incoming MIDI information into RAM (the Session Log). Then, time permitting, it looks through the MIDI Messages stored on disk to look for Session Definition cue matches. You may notice a significant lag in controller sliders and keyboards while using Flypaper, depending upon the speed of your Macintosh. There are several techniques you can use to minimize the lag between a MIDI Message and a Flypaper reaction.

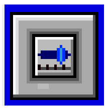
- Responses take time, including the Blinking of Session Cues option, the Autofront option, and the Autoscroll option. Turn them off for optimal cue recognition and response speed.
- More Session Definitions take longer to search through on each incoming MIDI Message

3. The Session Log Window

The Session Log Window allows you to control the recording, playback, and display formatting of MIDI information. The Session Log Window has six settings buttons on the left side of the window, and a tabular listing of the MIDI messages divided up into four columns covering the majority of the Session Log Window. The icons associated with different log options and functions will are framed with a dark colored square if they are "on."



Cue View Option **OFF**

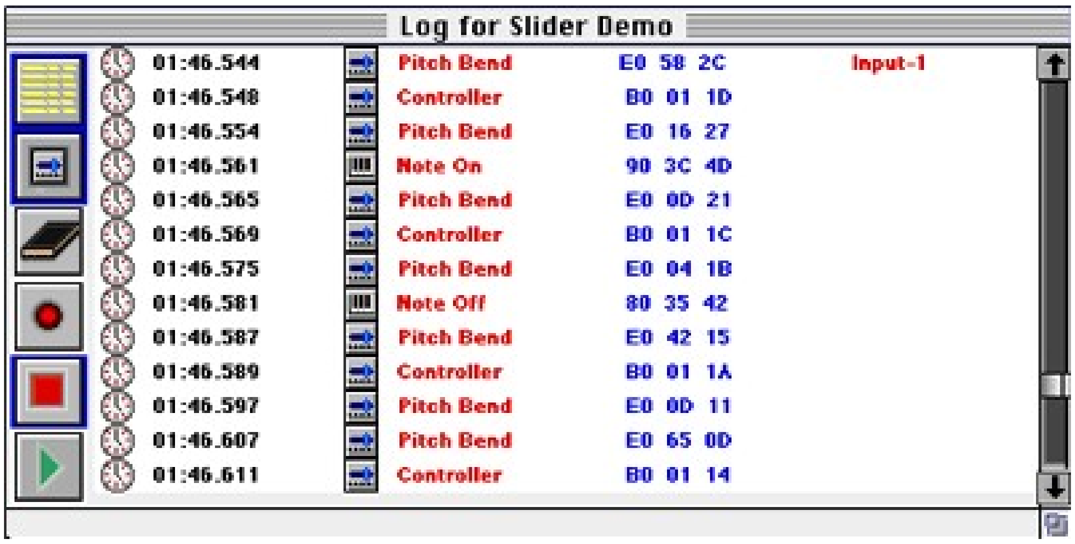


Cue View Option **ON**

The first column indicates the time at which a MIDI event was received by Flypaper. Times are shown in

hours **:** seconds **:** milliseconds

format. The second column shows any recognized cues from the Session's list of cues. If more than one cue is recognized in the MIDI data, all identified cues will appear in succession in the second column, listed in the order they were identified. The third column shows the actual MIDI messages. The fourth column shows which Flypaper MIDI Manager port the incoming messages were received from.



```
01:46.544   Pitch Bend   E0 58 2C   Input-1
01:46.548   Controller   B0 01 1D
01:46.554   Pitch Bend   E0 16 27
01:46.561   Note On      90 3C 4D
01:46.565   Pitch Bend   E0 0D 21
01:46.569   Controller   B0 01 1C
01:46.575   Pitch Bend   E0 04 1B
01:46.581   Note Off     80 35 42
01:46.587   Pitch Bend   E0 42 15
01:46.589   Controller   B0 01 1A
01:46.597   Pitch Bend   E0 0D 11
01:46.607   Pitch Bend   E0 65 0D
01:46.611   Controller   B0 01 14
```

Figure 3-1. A Log Window

The scroll bar on the right side of the Session Document's log window is used to scroll through the MIDI messages recorded in the log. The scroll bar arrows located at the upper and lower extremities of the bar can be used to increment through the log one MIDI byte at a time.

View Modes: MIDI View and Cue View

Clicking on the **MIDI View** button will toggle display of MIDI messages in the log on or off. If you don't want to look at all of the incoming MIDI messages, turn the Cue View option off.

Clicking on the **Cue View** button will toggle display of recognized MIDI cues from your Session's cue definitions on or off. If you don't want to see which cues are recognized in the list of MIDI messages, turn the Cue View option off.

Play

Clicking on the **Play** button will cause the entire Session Log of MIDI to be played out from Flypaper's MIDI Manager outputs. Input and output connections are assumed to be connected symmetrically in the MIDI Manager Patchbay. For instance, if the MIDI message was originally recorded from Flypaper's Input 1 port, it will be played back through Flypaper's Output 1 port. If the **Autoscroll Log** option is on (See Chapter 6- *The Options Menu*), Flypaper will scroll the log and highlight the current playback entry as it plays the log back.

Stop

Clicking on the **Stop** button will cause playback of the MIDI log to cease. The **Stop** button only works when a Flypaper log is being played back.

Record

Clicking on the **Record** button will trigger one of three recording modes for MIDI messages. The first mode is indicated by a flashing red or black border around the **Record** button, called *trigger record.* In trigger record, Flypaper waits until MIDI messages come in, and then begins recording the MIDI messages into the log. The first MIDI message received is considered to be at time 0:00:00. The second recording mode is called *record start* and is indicated by a solid red or black border around the **Record** button. In record start mode, Flypaper immediately begins recording any MIDI messages. The first piece of MIDI information is considered to be at time relative to when record start mode was initiated. The final recording mode, *record mute* is indicated by two black slash lines through the **Record** button in addition to a black border around the **Record** button. In record mute mode, no MIDI messages will be recorded to the log.

Clear Log

To clear the log of any MIDI message that have been stored in your Session Document, click on the **Clear Log** button. You will be warned that the log will be entirely cleared. The information stored in the Session Document on disk is not actually cleared until you issue the "Save" command from the **File** menu.

Stopping Hung Notes, All Notes Off

If you have notes that are stuck on your synthesizer, you can issue an "All Notes Off" command to all MIDI ports and channels attached to Flypaper by selecting the "Reset/All Notes Off" item from the **Options** menu. This feature is also useful to bring all Session Definitions up to date with MIDI that has been stored on disk. For instance, if Flypaper falls far behind in identifying cues, you can "catch up" with the "Reset/All Notes Off" menu selection.

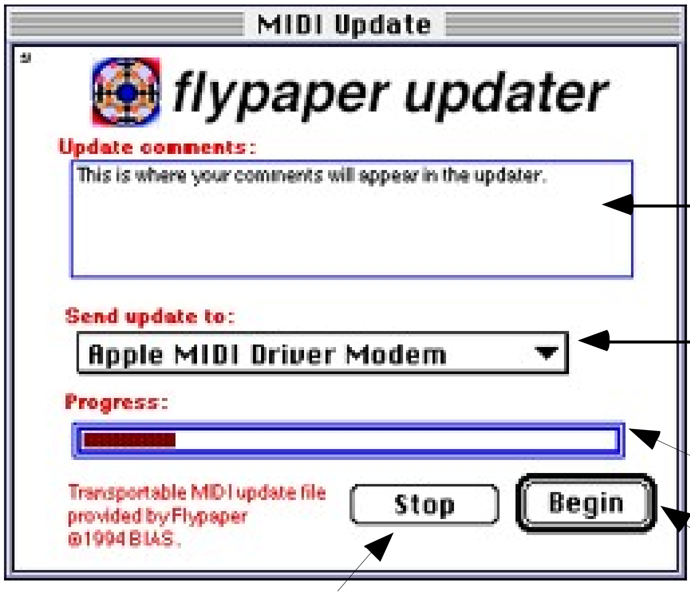
If your synthesizer does not support the All Notes Off MIDI controller message, then choose the "Exhaustive All Notes Off" option from the **Options** menu. This will cause every note on every channel of your keyboard to receive a note-on message with a velocity of zero, followed by a note-off message with a velocity of zero. If you issue the "Reset/All Notes Off" command from the **Options** menu, and you still hear hung notes, turn on the "Exhaustive All Notes Off" option in the **Options** menu. If you still hear hung notes, check your connections in the Patchbay application/desk accessory.

Creating a Transportable Updater Application

Flypaper can place the MIDI data from your Session Log into a transportable application that other users can use to receive sequences, system exclusive patches, or any other MIDI data that you have recorded. You are free to distribute Transportable Update applications created by Flypaper with no licensing fees or other costs as long as you do not modify the original Transportable Updater application.

In order to use the Transportable Updater option, you must first *create a copy of the "Transportable Updater.Original" file* that came with your Flypaper application. You can do this from the Finder by finding and selecting the "Transportable Updater.Original" application and then selecting "Duplicate" from the **File** menu in the Finder. Never create an updater from your original "Transportable Updater.Original" file, as you will not have a "blank" to install the MIDI messages into any longer. If this happens, reinstall the "Transportable Updater.Original" application from your Flypaper installation disks.

The Transportable Updater will store and playback any messages *from Port 1* stored in your Flypaper Session Document's log. The Transportable Updater that you create will contain a copy of your Session Log's MIDI messages, allowing another user to playback the messages through their MIDI devices through a simple interface:


MIDI Update
flypaper updater
Update comments:
This is where your comments will appear in the updater.
Send update to:
Apple MIDI Driver Modem
Progress:
Transportable MIDI update file provided by Flypaper ©1994 BIAS.
Stop
Begin

"Comments" on the content of your updater appears here.

All available MIDI Manager outputs are placed in a pop-up menu. Users do not need to use the Patchbay application as the patches are made automatically for them when they choose a menu item.

Playback progress is shown here

User clicks **Begin** to play your MIDI log

User clicks **Stop** to halt playback of the MIDI Update.

After recording MIDI messages into a Session Document's log, use the "Create Updater" command from the **File** menu. Updaters contain a "comments" field that you may enter with up to 255 characters describing the contents of the Transportable Updater. You will be asked to enter the comment after you choose the "Create Updater" item from the **File** menu.

After entering the comments for your Transportable Updater, the Standard File Picker will appear, asking you to locate a **COPY** of the Transportable Updater application that came with your Flypaper installation. Find the duplicate of the Transportable Updater application and click "Open". Flypaper will automatically install the contents of your current Session Document's log into the Transportable Updater.

When users click on the **Stop** button during playback from a Transportable Updater, the "Exhaustive All Notes Off" mode is used to send note-on messages of velocity zero to every note on every channel of their selected MIDI device.

4. The Session Window

The Session Window holds a scrollable list of Session Definitions. The Session Definitions are shown on each line of the list, with six entries per line. The six columns are labeled Cue, Port, Channel, Reaction, Port, Channel (see Figure 4-1).

Cue	Which cue to look for in the incoming MIDI messages
Port	The MIDI Manager port to look for the cue in
Channel	The Channel to look for the MIDI message on.
Reaction	Which reaction cue to initiate if the cue is recognized
Port	Which port to send the reaction cue (for MIDI reactions only)
Channel	Which channel to send the reaction cue on (for MIDI reactions only)

Creating New Session Definitions

To create a new Session Definition, select the "New Definition" menu item from the **File** menu. An empty Session Definition will appear in your Session Window.

If you cannot use the "New Definition" menu item from the **File** menu, you must first open an existing Session Document or create a new one with the "New Session" item from the **File** menu.

Deleting Session Definitions

To delete a Session Definition, select the definition you wish to delete my clicking on any item of the Session Definition that you wish to delete, then select the "Delete Session Definition" menu item from the **File** menu.

Moving Session Definitions

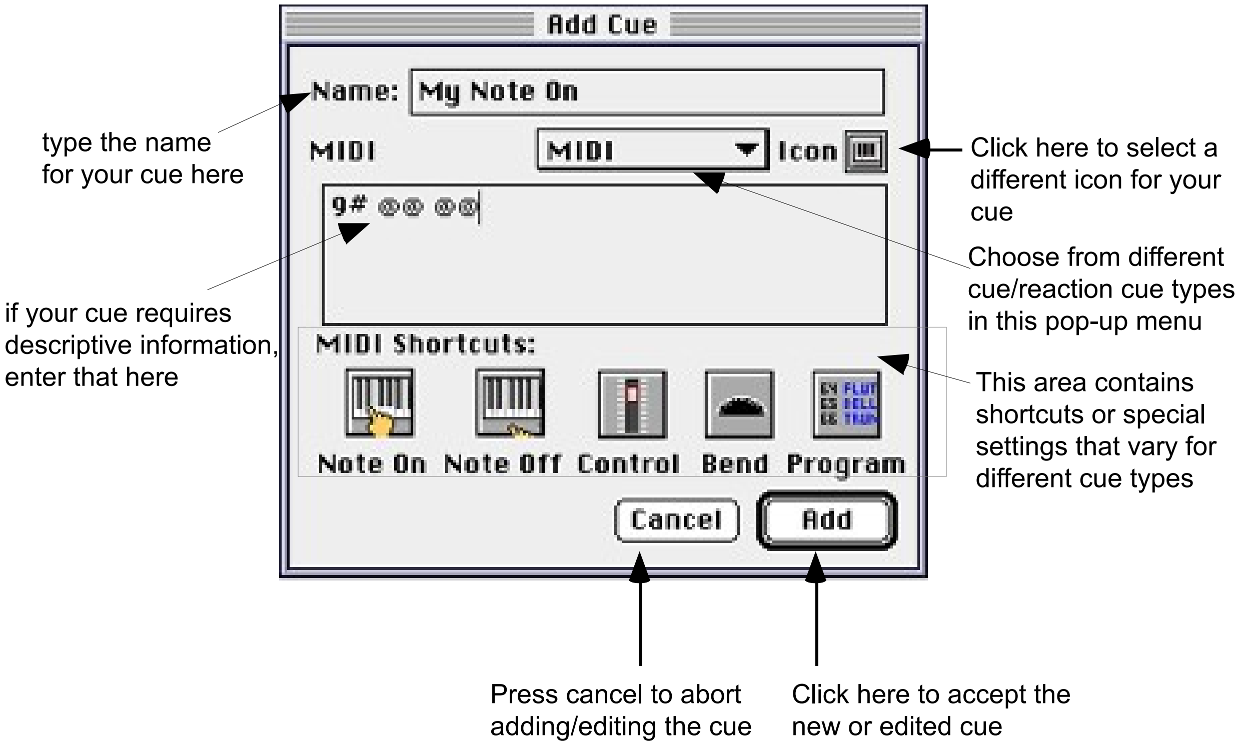
There may be times that you wish to move a session definition up or down in the list of Session Definitions. For instance, you may want to position an "And Also..." cue below another definition. To move a session definition up or down in the list of Session Definitions, click on a field of the Session Definition you wish to move to select it, then press the arrow keys

↑ ↓

on your keyboard to slide the Session Definition up or down.

Creating and Modifying Cues

To create a new cue, click on the cue column of any Session Definition and then select "Add Cue" or "Edit Cue" from the **Cue** menu. If a the cue column for a session definition is empty, then you may simply double-click on the cue column to add a new cue. If the cue column for the definition currently contains a cue, then double-clicking on it will cause the "Edit Cue" window for come up for that cue.


Add Cue
Name: My Note On
MIDI
MIDI
Icon
9# @@ @@
MIDI Shortcuts:
Note On
Note Off
Control
Bend
Program
Cancel
Add
type the name for your cue here
if your cue requires descriptive information, enter that here
Click here to select a different icon for your cue
Choose from different cue/reaction cue types in this pop-up menu
This area contains shortcuts or special settings that vary for different cue types
Press cancel to abort adding/editing the cue
Click here to accept the new or edited cue

Changing the Channel or Port of a Cue

If your cue has the '#' symbol in it's definition, then the '#' symbol will be replaced with the current setting of the Session Definition's channel field. To change the channel field, highlight the channel field of the Session Definition by clicking once on it. The **Channel** menu will become available on the menu bar, and you can select a new channel by selecting a value from the menu.

You may also change the port that the cue is recognized on by selecting the port field of a Session Definition, and then choosing an item from the **Port** menu.

Find

To find an occurrence of a cue in the log, highlight the Session Definition cue you wish to locate by clicking once on it and then use the "Find" item from the **Search** menu. Flypaper will beep if it cannot find an occurrence of the cue in the log.

Find Again

To find another occurrence of a cue in the log, select the "Find Again" item from the **Search** menu after using the "Find" command. Flypaper will beep if it cannot find any additional occurrences of the cue in the log.

Count Occurrences

To count the number of occurrences of a cue in the log, select the "Count Occurrences" item from the **Search** menu after using the "Find" command. Flypaper will beep if it cannot find occurrences of the cue in the log. Otherwise, you will be presented with the number of occurrences Flypaper was able to find in the log.

Invoking Reaction Cues Manually

At times you may wish to test a reaction cue manually from Flypaper. For instance, you may wish to try out an output MIDI cue to make sure it is working properly. In order to do this, hold down the [option] key. The cursor will change from the arrow to a lightning bolt ⚡. While holding down the [option] key, click on the Session Definition that you wish to invoke the reaction cue for.

MIDI Cue Syntax

To identify simple MIDI cue descriptors, type in the direct hexadecimal bytes you wish to search for, *separated by spaces.* Spaces act as delimiters for MIDI bytes. For example, if you wanted to identify a Sysex Start message, you would enter

```
F0
```

as the MIDI cue description.

When typing in the MIDI information that you wish to identify as a cue, Flypaper allows for some special symbols that make your MIDI cue definitions more flexible.

@ Symbol
Use the @ symbol as a "wild card" search character that can be anything. For instance, if you wish to identify note-on messages on any channel, not just channel 1 (e.g. "90") you would enter:

```
9@
```

as the MIDI cue descriptor.

$ Symbol
Use the $ symbol to store a 4-bit value into a Flypaper *user variable.* Sixteen user variables are available, $0 through $F. Don't confuse the $ symbol with "hexadecimal." Flypaper treats all numbers entered in the Add/Edit Cue dialog as hexadecimal values.

For example, to store the hi-byte of the key number in a note-on message and the low-byte of the key number, enter

```
90 $0$1
```

as the MIDI cue descriptor. If you also wanted to capture key velocity information to user variables $2 and $3, type

```
90 $0$1 $2$3
```

# Symbol
Use the # symbol to refer to the current Session Definition's channel.

Operators

The following symbols create an arithmetic operation :

+ Symbol
Adds two values together

- Symbol
Subtracts two values

***** Symbol
Multiplies two values

/ Symbol
Divides two values

>
Bitwise SHIFT RIGHT on two values

<
Bitwise SHIFT LEFT on two values

&
Bitwise AND on two values

|
Bitwise OR on two values

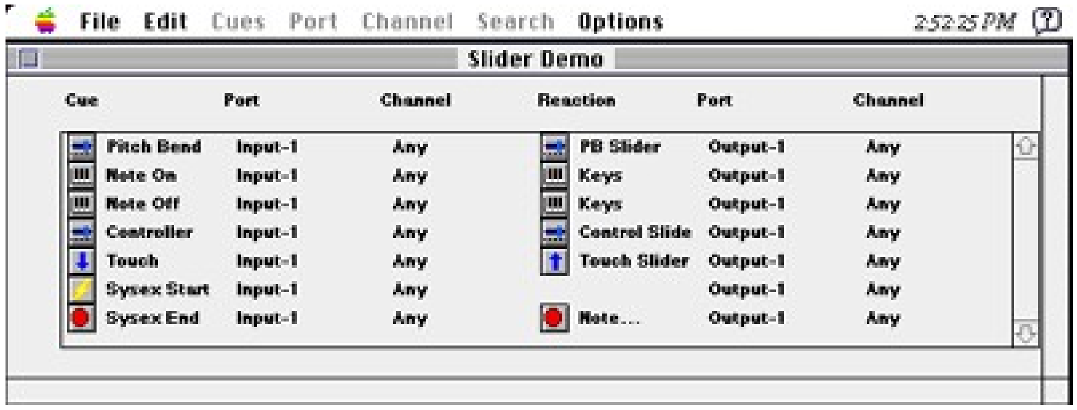
For example, perhaps you want to transpose an incoming note on message. The Session Definition *cue* descriptor would be:

90 $0$1 $2$3

and the Session Definition *reaction* cue descriptor could be:

90 $0$1+7 $2$3

using the arithmetic operator "+" to add seven (a perfect fifth) to every note on message's note number.



Cue	Port	Channel	Reaction	Port	Channel
Pitch Bend	Input-1	Any	PB Slider	Output-1	Any
Note On	Input-1	Any	Keys	Output-1	Any
Note Off	Input-1	Any	Keys	Output-1	Any
Controller	Input-1	Any	Control Slide	Output-1	Any
Touch	Input-1	Any	Touch Slider	Output-1	Any
Sysex Start	Input-1	Any		Output-1	Any
Sysex End	Input-1	Any	Note...	Output-1	Any

Figure 4-1. A Session Window, showing seven definitions.

5. Cues

Flypaper has a wealth of cues to identify incoming data, notify the user, transform messages, and output actions. In short, cues are at the heart of Flypaper. Below is a description of each cue, the format that the descriptive information on the cue must appear in, and an example usage of each cue type. Use the following formats for editing or adding new cues to your Session Document with the "Add Cue" or "Edit Cue" dialog accessible from the **Edit** menu.

MIDI Cue and Reaction Cue

The MIDI cue holds a description of the MIDI information to search for, or the MIDI information to send out.

Format

Use the MIDI cue as a reaction cue by entering the MIDI bytes you wish to send out in the description field. Output MIDI descriptions may also contain special characters @+/><*- described in Chapter 4, "Midi Cue Syntax."

And Also... Cue

The "And Also..." cue causes an *additional* reaction when the preceding cue is recognized.

This cue takes no description.

Alert Reaction Cue

Brings up a dialog with some user-specified text in it when executed.

Enter the text you wish to appear in the dialog into the description field of this reaction cue.

Sound Reaction Cue

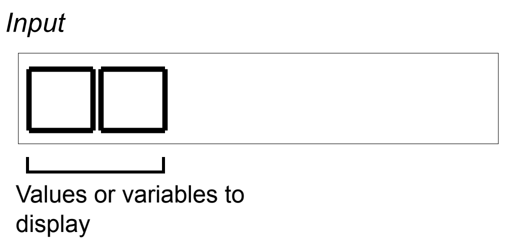
The Sound Reaction cue plays the "System Beep" sound when executed. The System Beep is set in the Sound Control Panel.

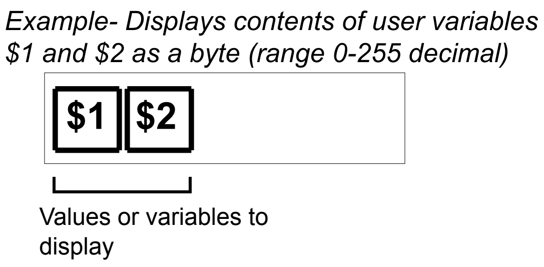
This cue takes no description.

Value Display Reaction Cue

This cue displays the value of a user variable.

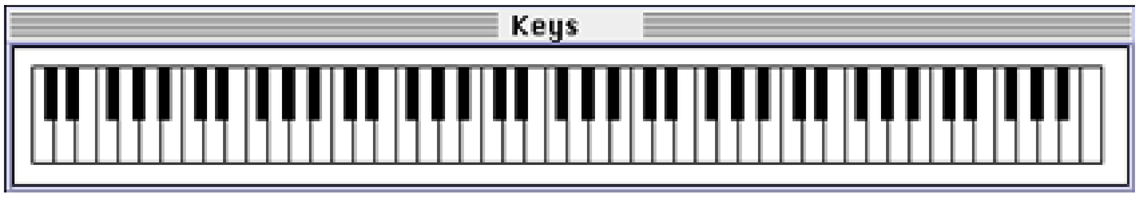
Format




Example- Displays contents of user variables $1 and $2 as a byte (range 0-255 decimal)
$1
$2
Values or variables to display

Keyboard Reaction Cue

Shows information on a graphic keyboard by taking two values: which note to highlight and a which note to unhighlight.

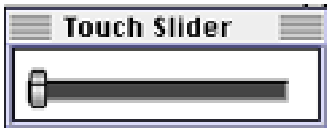


Format

Slider Reaction Cue

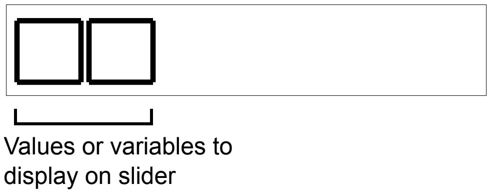
Shows information on a graphic slider, by taking one value: the value of the slider.

Shows information on a graphic slider. The name of the cue becomes the name of the window containing the slider.

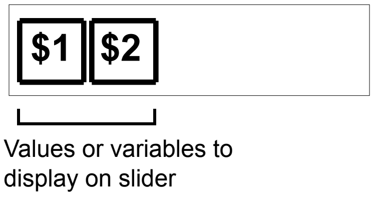


Format

Input



Example- Displays contents of user variables $1 and $2 as a byte (range 0-255 decimal)

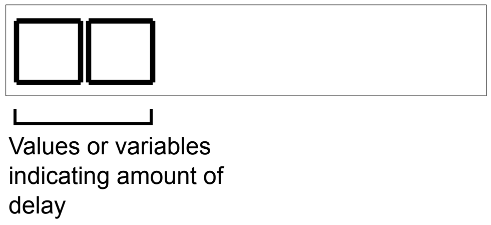


Delay Reaction Cue

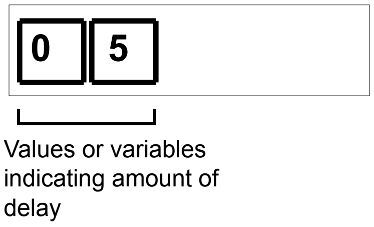
Delays for a specified count of 16.6 millisecond increments when executed.

Format

Input



Example- Delays for 5 increments of 16.6 milliseconds (approximately 93 ms)



Apple Event Cue

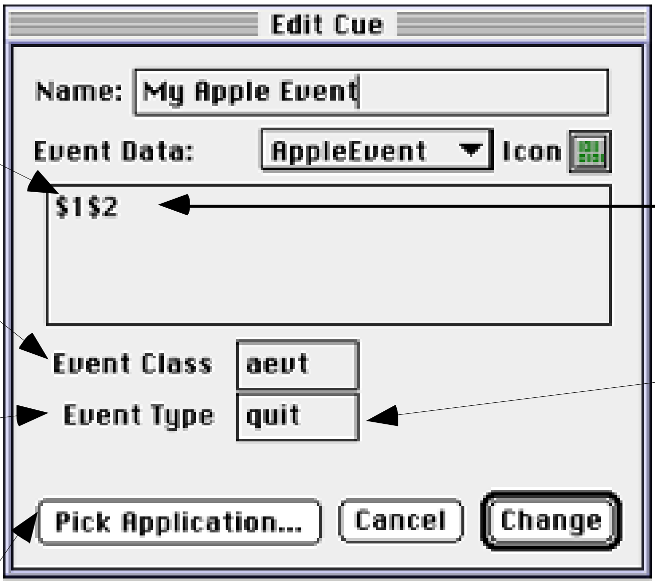
The Apple Event cue can trigger a cue when an Apple Event is received that matches the cue's apple event type that you specify (an Apple Event type such as 'plno'.. you may define as many as you wish). You may also issue Apple Events as reaction cues. This is a very powerful automation feature that allows you to control your MIDI hardware and Flypaper from another Macintosh application such as FileMaker Pro™ or HyperCard™. You can even control your other Macintosh applications with your MIDI hardware!

If you are sending an event, enter any data (including variables) you wish to send to an application here

If you are sending an event, enter the class of the event to send here

If you are sending an event, enter the event type to send here

If you are sending an event, click here to select which application to send the event to. Note the application must be running to send it.



Name: My Apple Event
Event Data: AppleEvent Icon
$1$2
Event Class aevt
Event Type quit

if you are receiving an event, type in any variables you wish to store data passed along with the event here.

if you are receiving an event, type the event type you wish to identify as a cue here

6. Tutorials

Flypaper includes ten tutorial Session Documents for you to investigate and try out in order to understand how Flypaper Session Documents can be created for different needs. You'll find the Tutorial documents in the folder "Tutorials."

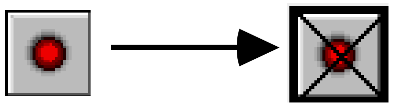
Tutorials

1. Playing back a Flypaper Log

This tutorial demonstrates how Flypaper can:

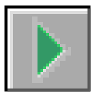
• Playback MIDI information from a Session Document's log.

1. Make sure Appletalk is off if you use your Printer port for MIDI activity.
2. Open the Flypaper Session Document "Tutorial 1-Playing back a Log" which is in the Tutorials folder.
3. Make sure Flypaper's MIDI Manager "Output 1" is connected to a MIDI output device (a synth, sampler, or internal synthesizer) by using the Patchbay application. After starting up Flypaper, switch to the Patchbay application to make your patches (don't Quit Flypaper as you will not see the Flypaper application in the Patchbay window). Assign Channel 1 of your output MIDI device to a piano sound if one is available.
4. Press the **Record** button in the Flypaper Log Window until it is in *record mute* mode:

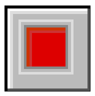


This keeps MIDI messages from being recorded into the log while you are playing back the log.

5. Press the **Play** button in the Flypaper Log Window. You should hear a counterpoint example.



6. If you grow tired of this masterpiece, press the **Stop** button in the Flypaper Log Window.



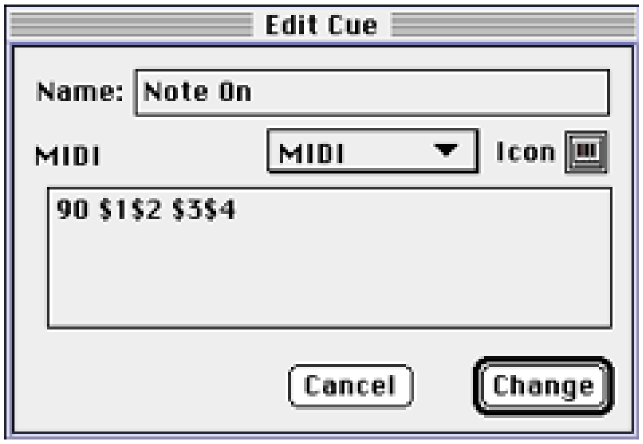
2. Note on, Note off

This tutorial demonstrates how Flypaper can :

- Use cues to identify MIDI messages as "Note On" or "Note Off"
- Hold note on and note off information in "user variables."

1. Make sure Appletalk is off if you use your Printer port for MIDI activity.
2. Open the Flypaper Session Document "Tutorial 2-Note On, Note Off" which is in the Tutorials folder.
3. Make sure Flypaper's MIDI Manager "Input 1" is connected to a MIDI input device (a synth, sampler, or internal synthesizer) by using the Patchbay application. After starting up Flypaper, switch to the Patchbay application to make your patches (don't Quit Flypaper as you will not see the Flypaper application in the Patchbay window)
4. Play some notes on your MIDI input device. Make sure it is transmitting on Channel 1. You should see the "Note On" cue become selected when you press a key, and the "Note Off" cue become selected when you release a key.

This Flypaper Session contains two Session Definitions. Both definitions have no reactions, just cues. The cue for "Note On" looks like:


Edit Cue

Name: Note On

MIDI: MIDI

```
90 $1$2 $3$4
```

MIDI Note-On messages consist of a Note-On Message in the upper 4 bytes (the "9"), followed by the channel (here, the "0" means channel 1), then the note number (ranges hexadecimal 0 - 7F), then the note velocity (ranges hexadecimal 0-7F)

The "Note On" cue recognizes MIDI messages with a hexadecimal 90 as a Note-On cue. In addition, this cue definition stores the note number into user variables 1 and 2 (indicated by "$1$2"), and then stores the key velocity into user variables 3 and 4 (indicated by "$3$4"). Any variable numbers could have been used to store the note numbers and the key velocities.

If you study the Note-off cue's definition, you will find that it is very similar to the Note-On cue's description.

You will see in **Tutorial 3** how you can utilize user variables to activate a graphic keyboard that displays which keys are on and which keys are off.

3. Adding a Keyboard

This tutorial demonstrates how Flypaper can

- Show information with the graphic keyboard
- Find occurrences of cues in the MIDI log.

1. Make sure Appletalk is off if you use your Printer port for MIDI activity.

2. Open the Flypaper Session Document "Tutorial 2-Note On, Note Off" which is in the Tutorials folder.

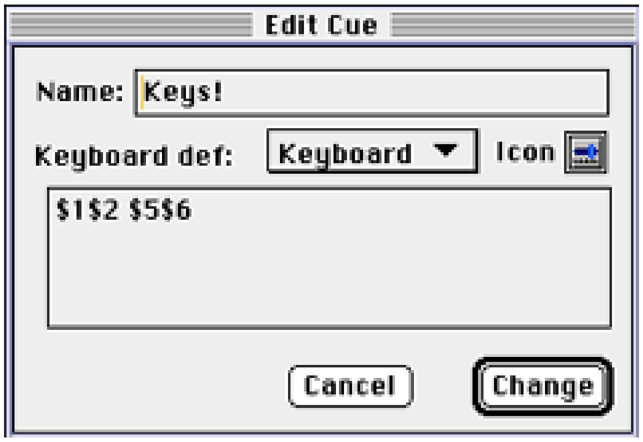
3. Make sure Flypaper's MIDI Manager "Input 1" is connected to a MIDI input device (a synth, sampler, or internal synthesizer) by using the Patchbay application. After starting up Flypaper, switch to the Patchbay application to make your patches (don't Quit Flypaper as you will not see the Flypaper application in the Patchbay window)

4. Play some notes on your MIDI input device. Make sure it is transmitting on Channel 1. You should see the "Note On" cue become selected when you press a key, and the "Note Off" cue become selected when you release a key. You should also see an actual graphic illustration of which keys are pressed down on your keyboard on the "Keys!" window.

This Flypaper Session Document contains the same two Session Definitions that were used in **Tutorial 2**, but now the Session Definitions have reactions defined for Note On and Note Off events. In fact, the reaction defined for the Note-On cue and the reaction defined for the Note-Off cue are the same in this Tutorial; they are both a graphic keyboard.

The graphic keyboard reaction cue takes two arguments for it's descriptor... a note on value and a note-off variable. The note-on variable tells the graphic keyboard which piano key to highlight, and the note-off variable tells the graphic keyboard which piano key to.

As you learned in **Tutorial 2**, the note-on key numbers are being stored into user variables 1 and 2. Additionally, the note-off key numbers are stored into user variables 5 and 6. Since the graphic keyboard wants two values, highlight number and unhighlight number, the graphic keyboard's reaction cue looks like:



Name: Keys!
Keyboard def: Keyboard
Icon

$1$2 $5$6

Now highlight the Note On Session Definition cue in the Session Window. The note-on cue is now "selected." Pull down the **Search** menu and select the "Find" menu item. You will see the Session Log scroll to the first occurrence of the Note-On cue in the log. The Note-On log event will be highlighted. To view the next occurrence of a Note-On cue in the log, pull down the **Search** menu again and select the "Find Next" menu item. You can continue with the "Find Next" menu command until the last occurrence of the Note-On cue is found. When there are no more occurrences of the Note-On cue, your Macintosh will beep. Use the "Find" menu item in the **Search** menu to begin the search again at the top of the Session Log. This is how you can find the location, time, and context in which a MIDI cue occurs in your recorded MIDI log.

4. Graphs

This tutorial demonstrates how Flypaper can

• Display information on the graphic slider

1. Make sure Appletalk is off if you use your Printer port for MIDI activity.

2. Open the Flypaper Session Document "Tutorial 2-Note On, Note Off" which is in the Tutorials folder.

3. Make sure Flypaper's MIDI Manager "Input 1" is connected to a MIDI input device (a synth, sampler, or internal synthesizer) by using the Patchbay application. After starting up Flypaper, switch to the Patchbay application to make your patches (don't Quit Flypaper as you will not see the Flypaper application in the Patchbay window)

4. Move a controller on your MIDI input device (i.e. Mod Wheel, Controller). Make sure it is transmitting on Channel 1. You should see the "Controller" cue become selected when you move the controller. You should also see an actual graphic slider of the controller as the controller is moved on your MIDI device on the "Controller Slider" window.

The controller reaction cue takes one variable, a value between 0 and 7F hexadecimal. This Tutorial reinforces the concepts learned in **Tutorial 2** and **Tutorial 3** but with a graphic slider instead of a keyboard. These values were stored into the user variables 3 and 4 when the controller cue was identified.

The controller cue looks for MIDI "B" followed by a "0" hexadecimal ("0" being channel 1). Controller messages contain the controller byte ("B" followed by the channel), the controller number (another byte), and then the controller value (another byte):

```
Bc      vw      xy
```

where

c is the channel
v is the upper four bits of the controller number
w is the lower four bits of the controller number

x is the upper four bits of the controller value
y is the lower four bits of the controller value

5. MIDI Delays

This tutorial demonstrates how Flypaper can

- Use delays
- Do more than one reaction per cue using the "And Also..." cue type
- Issue MIDI responses
- Use the Value Display response cue

1. Make sure Appletalk is off if you use your Printer port for MIDI activity.

2. Open the Flypaper Session Document "Tutorial 5-MIDI Delays" which is in the Tutorials folder.

3. Make sure Flypaper's MIDI Manager "Input 1" is connected to a MIDI input device (a synth, sampler, or internal synthesizer) by using the Patchbay application. After starting up Flypaper, switch to the Patchbay application to make your patches (don't Quit Flypaper as you will not see the Flypaper application in the Patchbay window). Make sure Flypaper's "Output 1" is connected to the same device that is connected to Flypaper's "Input 1".

4. Play some notes on your MIDI input device. Make sure it is transmitting on Channel 1. You should hear a repeat of each note, arpreggiations of chords, and a general backlog of time dependent upon your Mod-wheel position or other MIDI controller between note-on events.

Notice that the second Session Definition cue, titled "then do this" is a Session cue of the type "And Also...". When the Note-On Session Definition cue on the first line is triggered by a MIDI Note on message on the line above, Flypaper checks the Session Definition below it to see if it is a cue of the "And Also..." type. After the delay reaction cue on the first line Session Definition is invoked, Flypaper invokes the reaction cue on the second line Session Definition. The reaction of the second line Session Definition is to create a note-on exactly like the note on that came in. The note-on cue stored the note number and key velocity into user variables so it could reproduce the note after the delay. The note-off Session Definition cue on line three turns the duplicate note off.

Since each note-on event must pass through the delay, chords will cause a delayed arpreggiation effect.

6. Alerts and System Beeps while recording SYSEX

1. Make sure Appletalk is off if you use your Printer port for MIDI activity.

2. Open the Flypaper Session Document "Tutorial 2-Note On, Note Off" which is in the Tutorials folder.

3. Make sure Flypaper's MIDI Manager "Input 1" is connected to a MIDI input device (a synth, sampler, or internal synthesizer) by using the Patchbay application. After starting up Flypaper, switch to the Patchbay application to make your patches (don't Quit Flypaper as you will not see the Flypaper application in the Patchbay window)

4. Issue a MIDI System Exclusive (Sysex) message from your MIDI controller device, if possible. This is usually referred to as a "dump" of user parameters, samples, or some other data exclusive to the MIDI device.

You should see the Sysex Start Session definition cue become selected as soon as you issue the MIDI Sysex Exclusive transmission from your MIDI controller device, and you will hear a beep. The Sysex Start Session Definition cue searches for occurrences of "F0" in the incoming MIDI messages. If one is located, Flypaper issues the beep reaction. Note on larger system exclusive messages Flypaper may first attain to recording the MIDI information to disk before it recognizes the Sysex Start cue.

When the Sysex End Session definition cue is triggered (by an incoming "F7" in the MIDI messages), the Note... Session definition reaction cue is triggered. The Alert type reaction cue brings up a user-dialog giving the time of the reaction, and a user-specified text message. You can edit the user message by double clicking on the Note... Session Definition cue on the second line of the Session Window.

10. Apple Events automation of Flypaper (Filemaker Pro™)

You will need the Filemaker Pro™ application from Claris to use this example.

1. Make sure Appletalk is off if you use your Printer port for MIDI activity. If you've come this far, you probably already have automatic connections to Flypaper's MIDI Manager input and output ports. If you don't launch Flypaper, connect the ports, and quit.

2. Open the Filemaker Pro™ document "FMPro Tutorial 10" which is in the Tutorials folder.

After Filemaker Pro starts up, you should see a window with three buttons on it: setup, play notes, and stop notes. Each button has a Script associated with it (a series of Filemaker Pro actions, such as issuing Apple Events), so when you click on the button, the script is performed.

Click on the setup button. The setup button sends AppleEvents to open up the Flypaper application with the "Tutorial 10-Apple Events" Session Document. You can look at the script from Filemaker Pro™ by selecting the "ScriptMaker™..." menu item from the **Scripts** menu and then double clicking on the script titled "Load Tutorial-10".

The Flypaper **Tutorial 10** document contains two Apple Event cues, titled AE #1 and AE #2. The AE #1 Session cue contains a four-character identifier that identifies the Apple Event type to search for in incoming events. This can be any four characters you choose, in this case, "pln1" and "pln2" were chosen. When an Apple Event comes into Flypaper that matches "pln1" or "pln2", the appropriate reaction cues for the Session Definitions are invoked.

Click on the play notes button. You should hear two notes from the MIDI device connected to Output 1 of Flypaper's MIDI Manager ports. When you click on the stop notes button, the notes should stop playing. The play notes button sends an Apple Event type "pln1" to Flypaper. If you examine the Apple Event description of the "Play Notes" script in Filemaker Pro™, you will also find that the class is "flae".

Flypaper will ignore any AppleEvents that are not of the class ID "flae".

Similarly, the stop notes button sends an Apple Event type "pln2" to Flypaper, which causes the note-off messages to be sent to the two notes turned on with the play notes button.

With Apple Events automation, you can quickly assemble Filemaker Pro™ interfaces that control your MIDI hardware. External editors can quickly be assembled. You might even try to build a database that automatically loads presets into memory on your synthesizer and then auditions a sound for you.

Appendix A:Standard MIDI Messages

Bold hexadecimal characters are constant, and identify the type of MIDI Message.

Note On

9c Nn Vv

c = Channel number (range 0 - F)
N=Upper 4 bits of note number
n=Lower 4 bits of note number
V=Upper 4 bits of key velocity
v = Lower 4 bits of key velocity

Note Off

8c Nn Vv

c = Channel number (range 0 - F)
N=Upper 4 bits of note number
n=Lower 4 bits of note number
V=Upper 4 bits of key velocity
v = Lower 4 bits of key velocity

Touch

Dc Xx

c = Channel number (range 0 - F)
X = Upper 4 bits of key touch amount
x = lower 4 bits of key touch amount

Preset Change

Cc Xx

c = Channel number (range 0 - F)
X = Upper 4 bits of preset number
x = lower 4 bits of preset number

Controller

Bc Nn Xx

c = Channel number (range 0 - F)
N = Upper 4 bits of controller number
n= lower 4 bits of controller number
X = Upper 4 bits of controller amount
x = Lower 4 bits of controller amount

Pitch Bend

Ec Xx Yy

c = Channel number (range 0 - F)
X = Upper 4 bits of MSB (Most Significant Byte=1st byte of 2) of bend amount
x= lower 4 bits of MSB of bend amount

Y = Upper 4 bits of LSB (Least Significant Byte=2nd byte of 2) of bend amount
v= Lower 4 bits of LSB of bend amount

Start of System Exclusive Message

F0

End of System Exclusive Message

F7

Time Code

F8

Appendix B:Resources for Beginners

The USENET MIDI Primer
Bob McQueer

PURPOSE

It seems as though many people in the USENET community have an interest in the Musical Instrument Digital Interface (MIDI), but for one reason or another have only obtained word of mouth or fragmentary descriptions of the specification. Basic questions such as "what's the baud rate?", "is it EIA?" and the like seem to keep surfacing in about half a dozen newsgroups. This article is an attempt to provide the basic data to the readers of the net.

REFERENCE

The major written reference for this article is version 1.0 of the MIDI specification, published by the International MIDI Association, copyright 1983. There exists an expanded document. This document, which I have not seen, is simply an expansion of the 1.0 spec. to contain more explanatory material, and fill in some areas of hazy explanation. There are no radical departures from 1.0 in it. I have also heard of a "2.0" spec., but the IMA claims no such animal exists. In any event, backwards compatibility with the information I am presenting here should be maintained.

CONVENTIONS

I will give constants in C syntax, ie. 0x for hexadecimal. If I refer to bits by number, I number them starting with 0 for the low order (1's place) bit. The following notation:

>>

text

<<

will be used to delimit commentary which is not part of the "bare- bones" specification. A sentence or paragraph marked with a question mark in column 1 is a point I would kind of like to hear something about myself.

OK, let's give it a shot.

PHYSICAL CONNECTOR SPECIFICATION

The standard connectors used for MIDI are 5 pin DIN. Separate sockets are used for input and output, clearly marked on a given device. The spec. gives 50 feet as the maximum cable length. Cables are to be shielded twisted pair, with the shield connecting pin 2 at both ends. The pair is pins 4 and 5, pins 1 and 3 being unconnected:

```
                    2                                            5        4
          3                     1
```

A device may also be equipped with a "MIDI-THRU" socket which is used to pass the input of one device directly to output.

>> I think this arrangement shows some of the original conception of MIDI more as a way of allowing keyboardists to control multiple boxes than an instrument to computer interface. The

"daisy-chain" arrangement probably has advantages for a performing musician who wants to play "stacked" synthesizers for a desired sound, and has to be able to set things up on the road. <<

ELECTRICAL SPECIFICATION

Asynchronous serial interface. The baud rate is 31.25 Kbaud (+/- 1%). There are 8 data bits, with 1 start bit and 1 stop bit, for 320 microseconds per serial byte.

MIDI is current loop, 5 mA. Logic 0 is current ON. The specification states that input is to be opto-isolated, and points out that Sharp PC-900 and HP 6N138 optoisolators are satisfactory devices. Rise and fall time for the optoisolator should be less than 2 microseconds.

The specification shows a little circuit diagram for the connections to a UART. I am not going to reproduce it here. There's not much to it - I think the important thing it shows is +5 volt connection to pin 4 of the MIDI out with pin 5 going to the UART, through 220 ohm load resistors. It also shows that you're supposed to connect to the "in" side of the UART through an optoisolator, and to the MIDI-THRU on the UART side of the isolator.

>> I'm not much of a hardware person, and don't really know what I'm talking about in paragraphs like the three above. I DO recognize that this is a "non-standard" specification, which won't work over serial ports intended for anything else. People who do know about such things seem to either have giggling or gagging fits when they see it, depending on their dispos- itions, saying things like "I haven't seen current loop since the days of the old teletypes". I also know the fast 31.25 Kbaud pushes the edge for clocking commonly available UART's. <<

DATA FORMAT

For standard MIDI messages, there is a clear concept that one device is a "transmitter" or "master", and the other a "receiver" or "slave". Messages take the form of opcode bytes, followed by data bytes. Opcode bytes are commonly called "status" bytes, so we shall use this term.

>> very similar to handling a terminal via escape sequences. There aren't ACK's or other handshaking mechanisms in the protocol. <<

Status bytes are marked by bit 7 being 1. All data bytes must contain a 0 in bit 7, and thus lie in the range 0 - 127.

MIDI has a logical channel concept. There are 16 logical channels, encoded into bits 0 - 3 of the status bytes of messages for which a channel number is significant. Since bit 7 is taken over for marking the status byte, this leaves 3 opcode bits for message types with a logical channel. 7 of the possible 8 opcodes are used in this fashion, reserving the status bytes containing all 1's in the high nibble for "system" messages which don't have a channel number. The low order nibble in these remaining messages is really further opcode.

>> If you are interested in receiving MIDI input, look over the SYSTEM messages even if you wish to ignore them. Especially the "system exclusive" and "real time" messages. The real time messages may be legally inserted in the middle of other data, and you should be aware of them, even though many devices won't use them. <<

VOICE MESSAGES

I will cover the message with channel numbers first. The opcode determines the number of data bytes for a single message (see "running status byte", below). The specification divides these into "voice" and "mode" messages. The "mode" messages are for control of the logical channels, and the control opcodes are piggybacked onto the data bytes for the "parameter" message. I will go into this after describing the "voice messages". These messages are:

status byte   meaning   data bytes

0x80-0x8f	note off	2 - 1 byte pitch, followed by 1 byte velocity
0x90-0x9f	note on	2 - 1 byte pitch, followed by 1 byte velocity
0xa0-0xaf	key pressure	2 - 1 byte pitch, 1 byte pressure (after-touch)
0xb0-0xbf	parameter	2 - 1 byte parameter number, 1 byte setting
0xc0-0xcf	program	1 byte program selected
0xd0-0xdf	chan. pressure	1 byte channel pressure (after-touch)
0xe0-0xef	pitch wheel	2 bytes gives a 14 bit value, least significant 7 bits first

Many explanations are necessary here:

For all of these messages, a convention called the "running status byte" may be used. If the transmitter wishes to send another message of the same type on the same channel, thus the same status byte, the status byte need not be resent.

Also, a "note on" message with a velocity of zero is to be synonymous with a "note off". Combined with the previous feature, this is intended to allow long strings of notes to be sent without repeating status bytes.

>> From what I've seen, the "zero velocity note on" feature is very heavily used. My six-trak sends these, even though it sends status bytes on every note anyway. Roland stuff uses it. <<

The pitch bytes of notes are simply number of half-steps, with middle C = 60.

>> On keyboard synthesizers, this usually simply means which physical key corresponds, since the patch selection will change the actual pitch range of the keyboard. Most keyboards have one C key which is unmistakably in the middle of the keyboard. This is probably note 60. <<

The velocity bytes for velocity sensing keyboards are supposed to represent a logarithmic scale. "advisable" in the words of the spec. Non-velocity sensing devices are supposed to send velocity 64.

The pitch wheel value is an absolute setting, 0 - 0x3FFF. The 1.0 spec. says that the increment is determined by the receiver. 0x2000 is to correspond to a centered pitch wheel (unmodified notes)

>> I believe standard scale steps are one of the things discussed in expansions. The six-trak pitch wheel is up/down about a third. I believe several makers have used this value, but I may be wrong.

The "pressure" messages are for keyboards which sense the amount of pressure placed on an already depressed key, as opposed to velocity, which is how fast it is depressed or released.

? I'm not really certain of how "channel" pressure works. Yamaha is one maker that uses these messages, I know. <<

Now, about those parameter messages.

Instruments are so fundamentally different in the various controls they have that no attempt was made to define a standard set, like say 9 for "Filter Resonance". Instead, it was simply assumed that these messages allow you to set "controller" dials, whose purposes are left to the given device, except as noted below. The first data bytes correspond to these "controllers" as follows:

data byte

0 - 31	continuous controllers 0 - 31, most significant byte
32 - 63	continuous controllers 0 - 31, least significant byte

64 - 95	on / off switches
96 - 121	unspecified, reserved for future.
122 - 127	the "channel mode" messages I alluded to above. See below.

The second data byte contains the seven bit setting for the controller. The switches have data byte 0 = OFF, 127 = ON with 1 - 126 undefined. If a controller only needs seven bits of resolution, it is supposed to use the most significant byte. If both are needed, the order is specified as most significant followed by least significant. With a 14 bit controller, it is to be legal to send only the least significant byte if the most significant doesn't need to be changed.

>> This may of, course, wind up stretched a bit by a given manufacturer. The Six-Trak, for instance, uses only single byte values (LEFT justified within the 7 bits at that), and recognizes >32 parameters <<

Controller number 1 is standardized to be the modulation wheel.

? Are there any other standardizations which are being followed by most manufacturers?

MODE MESSAGES

These are messages with status bytes 0xb0 through 0xbf, and leading data bytes 122 - 127. In reality, these data bytes function as further opcode data for a group of messages which control the combination of voices and channels to be accepted by a receiver.

An important point is that there is an implicit "basic" channel over which a given device is to receive these messages. The receiver is to ignore mode messages over any other channels, no matter what mode it might be in. The basic channel for a given device may be fixed or set in some manner outside the scope of the MIDI standard.

The meaning of the values 122 through 127 is as follows:

```
data byte     name                     second data byte
122           local control            0 = local control off, 127 = on
123           all notes off            0
124           omni mode off            0
125           omni mode on             0
126           monophonic mode          number of
                                       monophonic channels, or 0
                                       for a number equal to
                                       receivers voices
127           polyphonic mode          0

124 - 127 also turn all notes off.
```

Local control refers to whether or not notes played on an instruments keyboard play on the instrument or not. With local control off, the host is still supposed to be able to read input data if desired, as well as sending notes to the instrument. Very much like "local echo" on a terminal, or "half duplex" vs. "full duplex".

The mode setting messages control what channels / how many voices the receiver recognizes. The "basic channel" must be kept in mind. "Omni" refers to the ability to receive voice messages on all channels. "Mono" and "Poly" refer to whether multiple voices are allowed. The rub is that the omni on/off state and the mono/poly state interact with each other. We will go over each of the four possible settings, called "modes" and given numbers in the specification:

mode 1 - Omni on / Poly - voice messages received on all channels and assigned polyphonically. Basically, any notes it gets, it plays, up to the number of voices it's capable of.

mode 2 - Omni on / Mono - monophonic instrument which will receive notes to play in one voice on all channels.

mode 3 - Omni off / Poly - polyphonic instrument which will receive voice messages on only the basic channel.

mode 4 - Omni off / Mono - A useful mode, but "mono" is a misnomer. To operate in this mode a receiver is supposed to receive one voice per channel. The number channels recognized will be given by the second data byte, or the maximum number of possible voices if this byte is zero. The set of channels thus defined is a sequential set, starting with the basic channel.

The spec. states that a receiver may ignore any mode that it cannot honor, or switch to an alternate - "usually" mode 1. Receivers are supposed to default to mode 1 on power up. It is also stated that power up conditions are supposed to place a receiver in a state where it will only respond to note on / note off messages, requiring a setting of some sort to enable the other message types.

>> I think this shows the desire to "daisy-chain" devices for performance from a single master again. We can set a series of instruments to different basic channels, tie 'em together, and let them pass through the stuff they're not supposed to play to someone down the line.

This suffers greatly from lack of acknowledgement concerning modes and usable channels by a receiver. You basically have to know your device, what it can do, and what channels it can do it on.

I think most makers have used the "system exclusive" message (see below) to handle channels in a more sophisticated manner, as well as changing "basic channel" and enabling receipt of different message types under host control rather than by adjustment on the device alone.

The "parameters" may also be usurped by a manufacturer for mode control, since their purposes are undefined.

Another HUGE problem with the "daisy-chain" mental set of MIDI is that most devices ALWAYS shovel whatever they play to their MIDI outs, whether they got it from the keyboard or MIDI in. This means that you have to cope with the instrument echoing input back at you if you're trying to do an interactive session with the synthesizer. There is DRASTIC need for some MIDI flag which specifically means that only locally generated data is to go to MIDI out. From device to device there are ways of coping with this, none of them good. <<

SYSTEM MESSAGES

The status bytes 0x80 - 0x8f do not have channel numbers in the lower nibble. These bytes are used as follows:

byte	purpose	data bytes
0xf0	system exclusive	variable length
0xf1	undefined	
0xf2	song position	2 - 14 bit value, least significant byte first
0xf3	song select	1 - song number
0xf4	undefined	
0xf5	undefined	
0xf6	tune request	0
0xf7	EOX (terminator)	0

The status bytes 0xf8 - 0xff are the so-called "real-time" messages. I will discuss these after the accumulated notes concerning the first bunch.

Song position / song select are for control of sequencers. The song position is in beats, which are to be interpreted as every 6 MIDI clock pulses. These messages determine what is to be played upon receipt of a "start" real-time message (see below).

The "tune request" is a command to analog synthesizers to tune their oscillators.

The system exclusive message is intended for manufacturers to use to insert any specific messages they want to which apply to their own product. The following data bytes are all to be "data" bytes, that is they are all to be in the range 0 - 127. The system exclusive is to be terminated by the 0xf7 terminator byte. The first data byte is also supposed to be a "manufacturer's id", assigned by a MIDI standards committee. THE TERMINATOR BYTE IS OPTIONAL - a system exclusive may also be "terminated" by the status byte of the next message.

>> Yamaha, in particular, caused problems by not sending terminator bytes. As I understand it, the DX-7 sends a system exclusive at something like 80 msec. intervals when it has nothing better to do, just so you know it's still there, I guess. The messages aren't explicitly terminated, so if you want to handle the protocol (esp. in hardware), you should be aware that a DX-7 will leave you in "waiting for EOX" state a lot, and be sending data even when it isn't doing anything. This is all word of mouth, since I've never personally played with a DX-7. <<

some MIDI ID's:

Sequential Circuits	1
Bon Tempi	0x20
Kawai	0x40
Big Briar	2
S.I.E.L.	0x21
Roland	0x41
Octave / Plateau	3
Korg	0x42
Moog	4
SyntheAxe	0x23
Yamaha	0x43
Passport Designs	5
Lexicon	6
PAIA	0x11
Simmons	0x12
Gentle Electric	0x13
Fairlight	0x14

>> Note the USA / Europe / Japan grouping of codes. Also note that Sequential Circuits snarfed id number 1 - Sequential Circuits was one of the earliest participators in MIDI, some people claim its originator.

Two large makers missing from the original lineup were Casio and Oberheim. I know Oberheim is on the bandwagon now, and Casio also, I believe. Oberheim had their own protocol previous to MIDI, and when MIDI first came out they were reluctant to ? go along with it. I wonder what we'd be looking at if Oberheim had pushed their ideas and made them the standard. From what I understand they thought THEIRS was better, and kind of sulked for a while until the market forced them to go MIDI.

? Nobody seems to care much about these ID numbers. I can only imagine them becoming useful if additions to the standard message set are placed into system exclusives, with the ID byte to let you know what added protocol is being used. Are any groups of manufacturers considering consolidating their efforts in a standard extension set via system exclusives? <<

REAL TIME MESSAGES.

This is the final group of status bytes, 0xf8 - 0xff. These bytes are reserved for messages which are called "real-time" messages because they are allowed to be sent ANYPLACE. This includes in between data bytes of other messages. A receiver is supposed to be able to receive and process (or ignore) these messages and resume collection of the remaining data bytes for the message which was in progress. Realtime messages do not affect the "running status byte" which might be in effect.

? Do any devices REALLY insert these things in the middle of other messages?

All of these messages have no data bytes following (or they could get interrupted themselves, obviously). The messages:

0xf8	timing clock
0xf9	undefined
0xfa	start
0xfb	continue
0xfc	stop
0xfd	undefined
0xfe	active sensing
0xff	system reset

The timing clock message is to be sent at the rate of 24 clocks per quarter note, and is used to sync. devices, especially drum machines.

Start / continue / stop are for control of sequencers and drum machines. The continue message causes a device to pick up at the next clock mark.

>> These things are also designed for performance, allowing control of sequencers and drum machines from a "master" unit which sends the messages down the line when its buttons are pushed.

I can't tell you much about the trials and tribulations of drum machines. Other folks can, I am sure. <<

The active sensing byte is to be sent every 300 ms. or more often, if it is used. Its purpose is to implement a timeout mechanism for a receiver to revert to a default state. A receiver is to operate normally if it never gets one of these, activating the timeout mechanism from the receipt of the first one.

>> My impression is that active sensing is largely unused. <<

The system reset initializes to power up conditions. The spec. says that it should be used "sparingly" and in particular not sent automatically on power up.

AND NOW, CLIMBING TO THE PULPIT ....

>> - from here on out.

There are many deficiencies with MIDI, but it IS a standard. As such, it will have to be grappled with.

The electrical specification leaves me with only one question - WHY? What was wanted was a serial interface, and a perfectly good RS232 specification was to be had. WHY wasn't it used? The baud rate is too fast to simply convert into something you can feed directly to your serial port via fairly dumb hardware, also. The "standard" baud rate step you would have to use would be 38.4 Kbaud which very few hardware interfaces accept. The other alternative is to buffer messages and send them out a slower baud rate - in fact buffering of characters by some kind of I/O processor is very helpful. Hence units like the MPU-401, which does a lot of other stuff, too of course.

The fast baud rate with MIDI was set for two reasons I believe:

1) to allow daisy-chaining of a few devices with no noticeable end to end lag.

2) to allow chords to be played by just sending all the notes down the pipe, the baud rate being fast enough that they will sound simultaneous.

It doesn't exactly work - I've heard gripes concerning end to end lag on three instrument chains. And consider chords - at two bytes (running status byte being used) per note, there will be a ten character lag between the trailing edges of the first and last notes of a six note chord. That's 3.2 ms., assuming no "dead air" between characters. It's still pretty fast, but on large chords with voices possessing distinctive attack characteristics, you may hear separate note beginnings.

I think MIDI could have used some means of packetizing chords, or having transaction markers. If a "chord" message were specified, you could easily break even on byte count with a few notes, given that we assume all notes of a chord at the same velocity. Transaction markers might be useful in any case, although I don't know if it would be worth taking over the remaining system message space for them. I would say yes. I would see having "start" and "end" transaction bytes. On receipt of a "start" a receiver buffers up but does not act on messages until receipt of the "end" byte. You could then do chords by sending the notes ahead of time, and precisely timing the "end" marker. Of course, the job of the hardware in the receiver has been complicated considerably.

The protocol is VERY keyboard oriented - take a look at the use of TWO of the opcodes in the limited opcode space for "pressure" messages, and the inability to specify semitones or glissando effects except through the pitch wheel (which took up yet ANOTHER of the opcodes). All keyboards I know of modify ALL playing notes when they receive pitch wheel data. Also, you have to use a continuous stream of pitch wheel messages to effect a slide, the pitch wheel step isn't standardized, and on a slide of a large number of tones you will overrun the range of the wheel.

? Some of these problems would be addressed by a device which allowed its pitch wheel to have selective control - say modifying only the notes playing on the channel the pitch wheel message is received in, for instance. The thing for a guitar synthesizer to do, then, would be to use mode 4, one channel per string, and bends would only affect the one note. You could play a chord on a voice with a lot of release, then bend a note and not have the entire still sounding chord bend. Any such devices?

I think some of the deficiencies in MIDI might be addressed by different communities of interest developing a standard set of system exclusives which answer the problem. One perfect area for this, I think, is a standard set for representation of "non- keyboard / drum machine" instruments which have continuous pitch capabilities. Like a pedal steel, for instance. Or non-western intervals. Like a sitar.

There is a crying need to do SOMETHING about the "loopback" problem. I would even vote for usurping a few more bytes in the mode messages to allow you to TURN OFF input echo by the receiver. With the local control message, you could then at least deal with something that would act precisely like a half or full duplex terminal. Several patchwork solutions exist to this problem, but there OUGHT to be a standard way of doing it within the protocol. Another thought is to allow data bytes of other than 0 or 127 to control echo on the existing local control message.

The lack of acknowledgement is a problem. Another candidate for a standard system exclusive set would be a series of messages for mode setting with acknowledgement. This set could then also take care of the loopback problem.

The complete lack of ability to specify standardized waveforms is probably another source of intense disappointment to many readers. Trouble is, the standard lingo used by the synthesizer industry and most working musicians is something which hails back to the first days of synthesizer design, deals with envelope generators and filters and VCO / LFO hardware parameters, and is very damn difficult to relate to Fourier series expressing the harmonic content or any other abstractions some people interested in doing computer composition would like. The parameter set used by the average synthesizer manufacturer isn't anyplace close to orthogonal in any sense, and is bound to vary wildly in comparison to anybody elses. There are essentially no abstractions made by most of the industry from underlying hardware parameters. What standardization exists reflects only the similarity in hardware. This is one quagmire that we have a long way to go to get out of, I think. It might be possible, eventually, to come up with translation tables describing the best way to approximate a desired sound on a given device in terms of its parameter set, but the difficulties are enormous. MIDI has chosen to punt on this one, folks.

Well, that's about it. Good luck with talking to your synthesizer.

Bob McQueer 22 Bcy, 3151

All rites reversed. Reprint what you like.