

1.58/2

MIDI Synth/synthLAB™

Version 1.0B3

This package contains

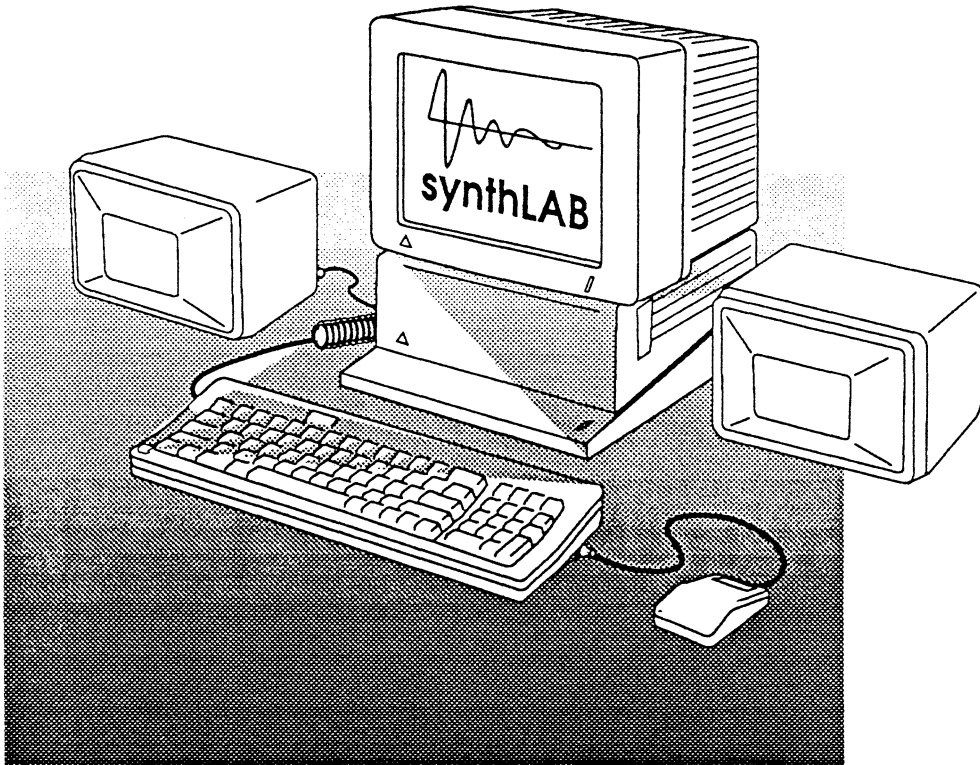
2	Manuals	<i>synthLAB User's Manual</i> <i>MIDI Synth External ERS</i>
1	Disk	<i>synthLAB</i> , Version 1.0B3
1	Software license agreement	

If you have any questions, please call

1-800-282-2732 (U.S.)
1-408-562-3910 (International)
1-800-637-0029 (Canada)

synthLAB™

User's Manual



by
Mark Cecys

Art and text by Mark Cecys
Copyright 1990, Apple Computer, Inc
Apple, AppleTalk, Apple IIGS, Macintosh,
and GS/OS are registered trademarks of
AppleComputer, Inc.

Quickdraw, synthLAB and MIDI Synth are
trademarks of Apple Computer, Inc
6/28/90

Table of Contents

Introduction	1
synthLAB Quick-Start	1
How to use this manual	2
Getting Started	3
Audio Output	4
Connecting a MIDI Device to synthLAB ...	5
Limitations	6
SynthLAB Overview	8
File Types	8
Navigating synthLAB	8
The Sequencer Page	10
The Deck Controls	10
Play	10
Record	11
Stop	11
Fast-Foward	11
Rewind	11
Auto-Rewind	11
Counter Display	11
Tempo Control	11
Track Controls	11
Track Name	12
Track Record	12
Track Play	12
Track Channel	12
Editing Instrument Records	14
Screen Keyboard	14
Envelope Edit Page	16
Envelope Sliders	16
Velocity Gain	17
Decay Gain	17
Pitch Bend	17
WaveList Edit Page	18
Top Key	18
Oscillator Configuration	19
Wave Select	19
Oscillator Tuning	20

Oscillator Volume	20
"Δ" Menu	21
About synthLAB	21
Clock	22
File Menu	23
New Sequence	24
Load Sequence	24
Save sequence	24
Import/Export Sequence	25
Load Instrument	25
Save Instrument	25
Load Waves	25
Quit	25
Edit Menu	26
View Sequence	26
Name Inst	27
Copy	27
Paste	27
Setup Menu	28
MIDI	28
Sequencer	29
Seq Output	29
System	30
Volume	30
All Notes Off	31
Appendix A - synthLAB File Formats ..	32
Instrument File Format	33
Wave File Format	36
Sequence File Format	38
Appendix B- The MIDI CDEV	40

Introduction



As the computer industry grows and matures, developers are under increasing pressure to continually create more complex and more sophisticated applications. At Apple Computer, we try to minimize this burden on developers by providing them with a wealth of OS routines called System Tools. Over the years, these Tools have grown and become very complex and sometimes unmanageable. For example, creating a new system Tool with a few vague notes written down by the engineer is no longer considered adequate support. We realize that you, as developers for the Apple IIGS®, need additional support in dealing with the continually increasing complexity of your applications.

synthLAB™ is a support application for developers who are creating programs using MIDI Synth™ (system Tool035). And like MIDI Synth, it has three basic parts to it: a synthesizer, a sequencer and a MIDI driver. With synthLAB, you can create the instruments you'll need for your application, either by modifying existing ones, or by creating totally new and original ones. With the synthLAB sequencer, you can record your custom sequences used in your application. You can take advantage of MIDI Synth software's multi-timbral feature by experimenting with different instrument combinations until you get the sound that you want. You no longer have to "hand-code" instruments and sequences into your source code without the benefit of hearing exactly what your creating. Using a tool like synthLAB, creating sound in your application should be easier and the result will certainly be more interesting to your users.

Since synthLAB itself is built around MIDI Synth, it makes for a great learning tool in understanding how MIDI Synth works. Most of synthLAB software's parameters directly correspond to MIDI Synth parameters. If you're not quite sure how a certain parameter affects MIDI Synth, with synthLAB you can go and actually try it, playing with different values until it becomes clear. You can also learn and develop the techniques needed in creating your own quality instruments by examining and manipulating the existing instruments provided with synthLAB. Making interesting instruments is an art, and the best way to learn this is by actually trying it yourself. So, synthLAB is also a powerful educational tool for developers.

synthLAB Quick-Start

After booting the synthLAB Program disk, double-click on the "synthLAB" icon to launch the program. Once loaded, you'll get the "player keyboard" title screen. If you don't hear any output sound or if you get an error message, see the topic "Getting Started" later in this section. When you're through listening to the demo, press either the IIGS keyboard "return" key or click on the *arrow* button on the bottom right of the screen to exit. The screen now shows the Sequencer Page, with deck control buttons to the left and eight sets of track controls on the right of the screen.

If you want, you can load and play some of the sample sequences and instruments on the Sequences and Instruments disk. Pull down the File menu button and select the "Load Sequence..." item. Remove the synthLAB Program disk and insert the Sequences and Instruments disk. Click on

the *Drive* button to list the sequences found on this disk. After double-clicking on your selection, the file dialog box goes away and synthLAB will load in the selected sequence with the appropriate instruments. When the arrow cursor returns, you're ready to play the sequence. You can either click on the deck *Play* button, or if you want to see the actual notes being played, select the "About synthLAB..." item from the "Δ" menu button. Either way will play the sequence for you.

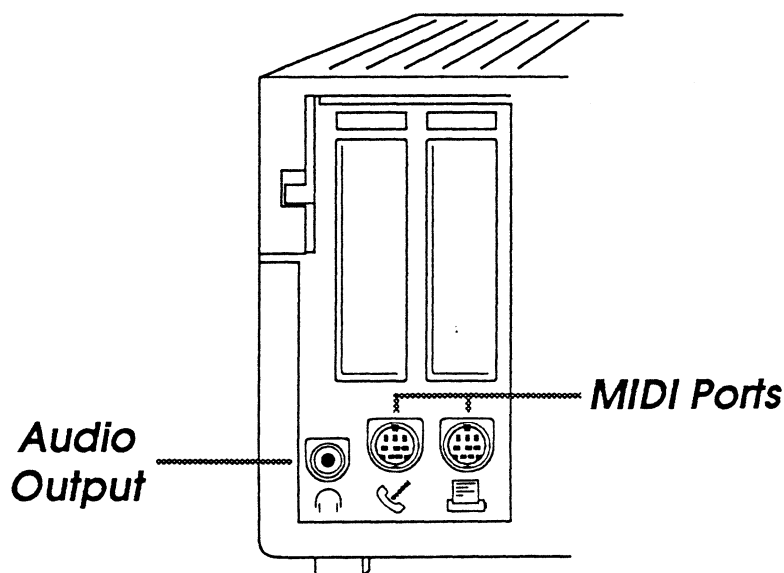
To get the full effect from synthLAB, try using a stereo interface card with the output going to an external amplifier/speaker system. You'll miss quite a lot if you only play synthLAB in "mono" through the low-quality internal speaker.

How to use this manual

This manual is a general reference guide for starting and navigating around synthLAB. By itself, it certainly is not complete, especially if you're going to create your own instruments. synthLAB software's primary purpose is to support and encourage developers working with MIDI Synth. Although in some ways it could be used as such, synthLAB is not a consumer product. The system architecture and many of the parameters and functions which synthLAB controls are described in the "MIDI Synth External ERS" and are not duplicated here. The "ERS" is very technically oriented, written primarily for programmers and developers. It is intended that you use this manual, synthLAB and the "ERS" together, referencing various sections in each until you get solid understanding of how things work. It is important that you read the "ERS", since this information is not covered here.

This manual and the "ERS" assumes that you have a good understanding of how MIDI works. Explaining and describing all the many aspects of MIDI is beyond the scope of this manual. There are many excellent books written about MIDI, which are available at your local music dealer if you need additional information.

Finally, there are some terms used in this manual that might cause some confusion. The term *keyboard* will always refer to an external MIDI keyboard (the musical kind). The keyboard that's part of your IIGS computer system that you type with, will be called the *IIGS keyboard*. Also, new names are given to some of the hardware ports on your IIGS. Since MIDI devices are connected to either the Modem or Printer ports on the back of the IIGS, either one will be referred to as a *MIDI port*. The headphone jack, which usually goes to some sort of external amplifier (if you're the least bit serious about sound quality), is called the *audio output port*. If you're using a stereo interface card, then the audio output port is the stereo output connector on the card.



Getting Started

As usual, the first thing you should do is make a back-up copy of the synthLAB disk, then store the original away somewhere safe. If you're using a hard disk, just copy the "synthLAB" folder over to your hard disk.

Next, copy "Tool035" (MIDI Synth) over to your "Tools" folder, which is found inside your boot "System" folder. If your boot disk is already full and you don't have any more room available then leave MIDI Synth in the same folder with synthLAB. When it initially runs, synthLAB tries to load MIDI Synth from the system "Tools" folder. If it can't find it there, it then looks for it in its own folder. If you have a hard disk, this should not be a problem.

If you plan on using MIDI along with synthLAB, you'll need to install MIDI into your system with the new MIDI CDEV. Simply move the file from the synthLAB disk called "MIDI" over to the "CDev" folder on your boot disk (found inside the "System" folder). Before running synthLAB, run the Control Panel NDA from the Apple menu. Select the CDEV called "MIDI" and enter the correct information which describes your MIDI setup (see Appendix B). Return back to the Finder and launch synthLAB.

After synthLAB has loaded, you should hear music playing at the title screen. If synthLAB encountered an error while trying to start, you'll get an error message telling you what is wrong.

You'll get an error if the AppleTalk® port is active. Due to the way interrupts work on the IIGS, you can't run MIDI and AppleTalk together (AppleTalk also eats up CPU time, which would degrade our performance). Go to the Control Panel and de-select AppleTalk. Reset, re-boot and try again.

Try to avoid using Inits and DA's which install themselves in the background (menu clocks, screen savers, etc.). Remember, nothing comes for free. These background programs continually steal valuable CPU time, making applications run slower. For 'real-time' programs like synthLAB, this is especially critical since they are competing for the same background interrupt time.

Requirements

- Apple® IIGS system
- 1M of system memory or greater
- GS/OS® 5.02 or greater
- One 3.5" drive
- "MIDI Synth External ERS" manual

Recommended

- External amplifier/speakers (home stereo, powered monitors, etc.)
- MIDI interface (Apple MIDI, *Audio Animator*, etc.)
- MIDI controller (external MIDI keyboard, MIDI guitar, wind controller, etc.)
- Stereo interface card (*Audio Animator*, etc)

Optional

- Reverb or Effects Processor
- MIDI synthesizer, sampler, etc.
- Hard disk

synthLAB will work on all IIGS models with a Rom version 01 or higher. Like most new applications these days, you'll need a minimum 1 meg of system ram. If your IIGS does not have 1 MB on the motherboard, you'll need a 1 meg. expansion card installed.

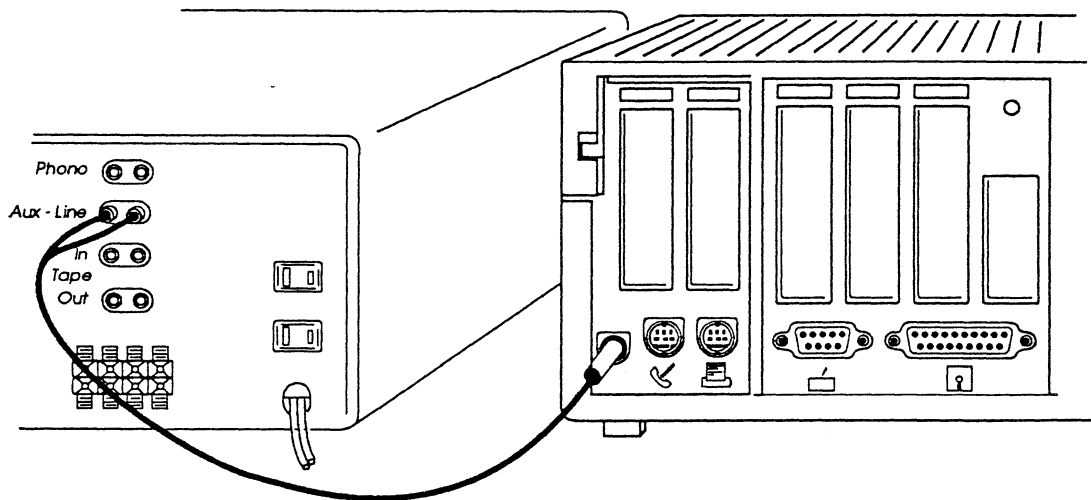
We don't know why anyone would have a IIGS with a monochrome monitor, but if you do, it won't be pretty.

If you're using an older version of GS/OS, then see your dealer for an upgrade to the latest version.

Audio Output

Unless you're the type who enjoys listening to music through a cheap 25-cent speaker, hook the GS audio output to an external audio system. Any home stereo or powered monitor speaker will do. If you're using a home stereo system, you'll need a *1/8" stereo mini plug to phono (RCA) plug* adapter cable to connect your GS to your amplifier. The *mini* plug goes to the audio output (headphone) jack on the back of your GS or your stereo interface card, while the two *phono* ends go to the *line* or *Aux* inputs usually on the back of your stereo amplifier (Even though the plugs are called *phono*, do NOT plug them into the *phono* input on your amplifier, you could damage your speakers if you do).

There are several things you can do to improve the overall quality of sound coming from your GS. First, try turning the *treble* control on your amplifier down a bit to filter out the "hiss" if you have a noisy GS or a noisy stereo interface.



Second, both synthLAB and MIDI Synth are designed to be most effective when played in stereo. By being able to assign each Generator (1/2 of an Instrument) to either stereo channel, many interesting stereo effects can be created. If you have a stereo interface card, see the owner's manual for installation instructions.

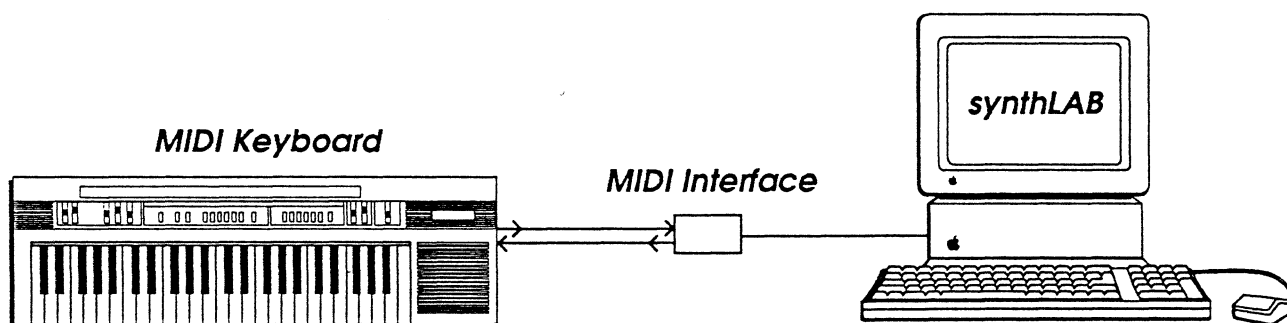
The author's personal recommendation is Applied Engineering's *Audio Animator*. Not only does this card produce very good quality stereo output, but it has a built-in MIDI interface, an excellent stereo digitizer and a convenient external control box for all your connections and level controls. This audio system includes useful software for digitizing and editing your own sounds, and for recording and playing MIDI sequences.

Finally, if you have a reverb unit or a special effects box, try using it on your GS.

Connecting a MIDI Device to synthLAB

If you happen to have one, hook an external MIDI keyboard up to your GS. Whether you're creating instruments and sequences, you'll get the most out of synthLAB if you control it thru MIDI.

Since synthLAB is both a synthesizer and a sequencer, you can do many useful things with your GS connected to a MIDI keyboard. Play notes on the keyboard, and your GS will track them by playing selected instruments. It will even respond to note velocity, pitch-bend and your volume and sustain pedals. Next, you can record your performances into an eight-track sequencer, recording each track separately using different instruments. You can save these sequences on disk,



building your own personal library. Finally, you can have synthLAB play your MIDI keyboard. If you run out of instruments or voices with synthLAB, you can assign any tracks to output their MIDI data out to your keyboard.

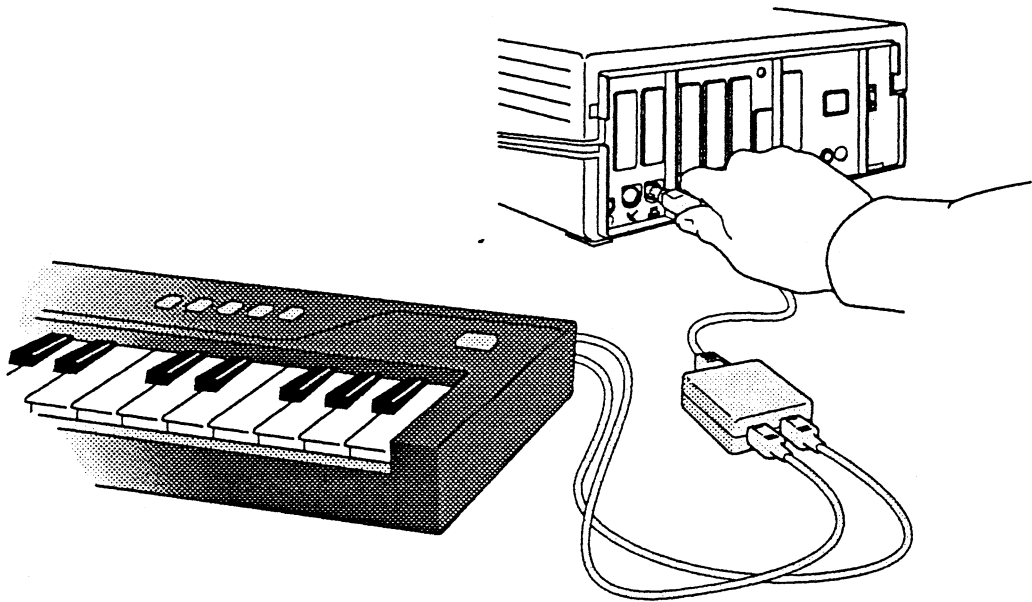
When creating instruments, the keyboard gives you more control in defining the various parameters. You can quickly test your instrument over the complete note and velocity range. Many instruments sound different when they interact with other voices, so you can play chords or octaves and hear this effect. If the instrument is multi-sampled (different waves for note zones), you can make sure that the transitions between WaveLists are seamless and that each multi-sample is balanced correctly with the others.

Since MIDI signals are electrically different from signals that computers use, you'll need to connect your keyboard through a special MIDI interface. Any 1mhz external type of interface (Apple MIDI for example) or an internal card interface (like Applied Engineering's *Audio Animator*) will work. For example, when using the Apple MIDI interface, you simply plug the MIDI cables into one end of the interface which connects to your MIDI keyboard, then plug the other end into a MIDI port on your IIGS System (Modem or Printer, see diagram below).

Once you start synthLAB, you'll need to enable the MIDI port. Pull the "Setup" menu down and select the "MIDI..." item. Then from the dialog box, click on the *MIDI In Enable* button to enable it (it's the icon with the arrow pointing into the computer). Press a few notes on the keyboard, and you should hear your IIGS play them. Notice that the "MIDI In" display on the top right corner of the screen lights up for every MIDI message received.

Limitations

When writing any application, you must limit the scope in which you intend to cover, in order to maintain a realistic schedule. Because of this, all applications have their limitations, and do not have all the features necessary to meet all user's needs. Just as it's important to know what an application does, it's also just as important to know what an application will not do.



Connecting an Apple MIDI interface

- synthLAB is intended to be an instrument editor. It works on banks of 16 instruments at a time. It does not have any "librarian" features enabling you to move instruments between banks. This would certainly be useful.

- synthLAB is not a waveform editor. The waves must be created and edited elsewhere. We at Apple will provide wave files for you to use, but it would be nice if you could sample and edit your own.

- The sequencer in synthLAB is very limited and lacks many of the editing features found in more advanced types of sequencers.

- Since synthLAB does not use QuickDraw to draw all images on the video screen, there are times when it does not know anything about “foreign” windows. Therefore, it can’t support NDA’s.

Developers should note that these are limitations of only synthLAB and not the System Tool, MIDI Synth. A professional style sequencer can very well be written around MIDI Synth. The custom interface was the author's personal preference: MIDI Synth was designed to work well with the standard IIGS Desktop (Windows, Menus, Controls, etc.).

synthLAB Overview



synthLAB has three basic parts to it: a synthesizer, a sequencer and a MIDI driver. Everything synthLAB does relates to one of these three parts. Since synthLAB's main purpose is to create instruments for MIDI Synth, it obviously has controls to manipulate all the parameters defined in an Instrument Record. To play an instrument, we need the synthesizer, so synthLAB has controls for that too. And finally, there are controls for the sequencer and the MIDI driver.

File Types

When you use a word processor, you typically load in your text file, manipulate the file by adding and editing words and sentences to the file, and when you're done, you save this file back to disk. The word processor creates and manipulates text files. synthLAB is, in many ways, like a word processor. But instead of working on text files, synthLAB works on instrument files and sequence files.

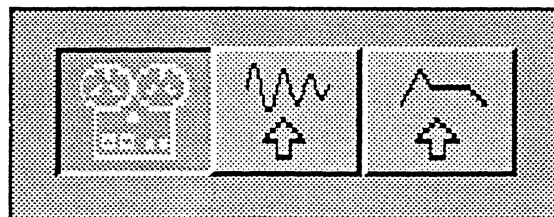
An *instrument file* is a collection of instruments grouped together in what is called an instrument *bank*. Each bank has 16 instruments and each instrument is actually an Instrument Record defining how the sound is built. In other words, a bank is simply a group of 16 Instrument Records. The instruments in a bank are all related to one another in that they all share the same *wave file*. This file contains the actual waveforms used in producing the output sound. synthLAB does not manipulate the wave file in any way, it only loads it into DOC ram after you load an instrument file. Once an instrument file is loaded, synthLAB can now edit and/or play any one of these 16 instruments.

A *sequence file* is the data output created by the sequencer. A Sequence File can contain only one sequence.

Navigating synthLAB

Since all of synthLAB features can't fit onto a single screen, the various functions have been divided up and placed into either separate video screens or grouped inside the menu buttons at the top of the screen. The three main video screens are called *pages*, and you can access any one of them them by pressing one of the three page buttons.

The left button selects the *Sequencer* page. You use this page whenever you wish to control sequence record or playback. The middle button selects the *WaveList Edit* page, which has controls to change the WaveList Record parameters. The button on the right, the *Envelope Edit* page, selects the controls used to edit Envelope Record parameters. These last two pages are used to define Instrument Records (see "MIDI Synth ERS").



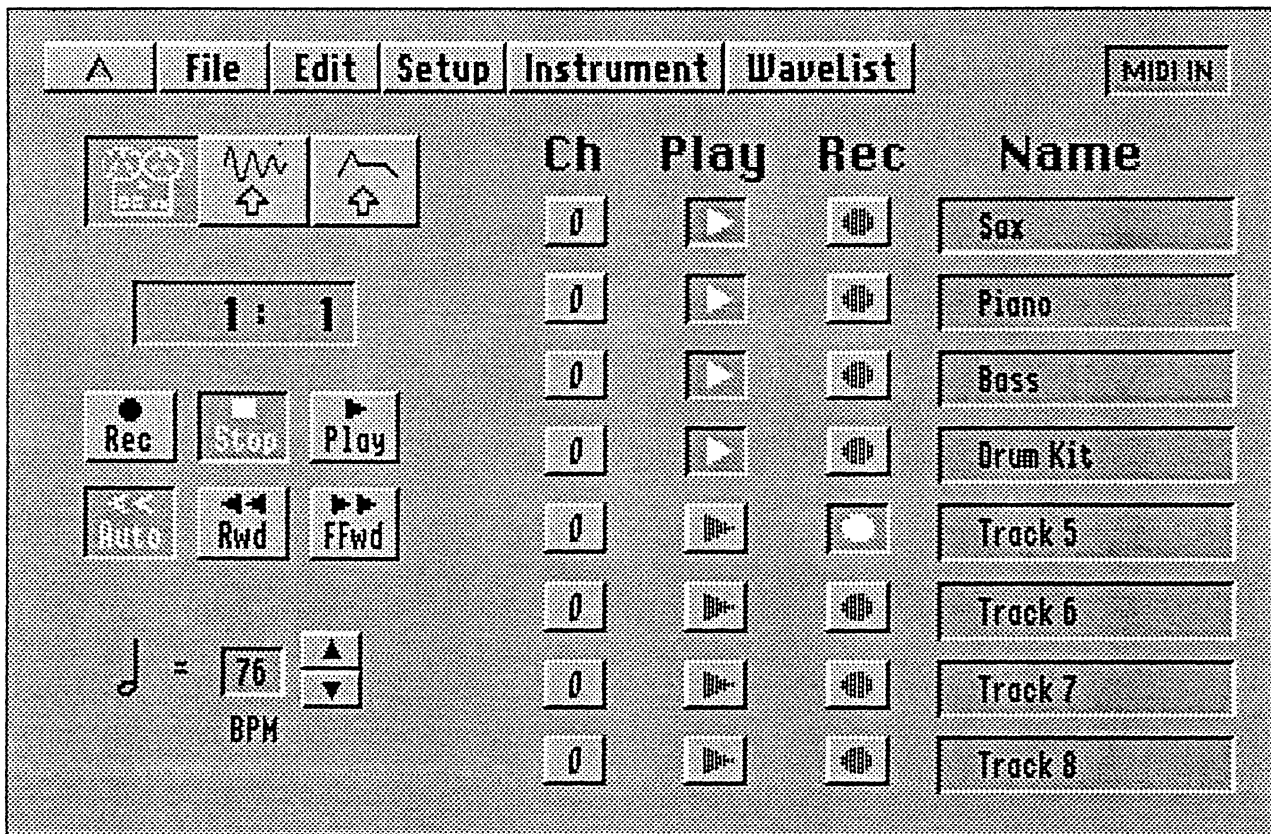
Page Buttons

The *menu buttons* have functions which are common for all three pages. Things like file I/O, program setup and instrument selection, are all initiated from these buttons.

The Sequencer Page



The Sequencer page can be grouped into three main areas. On the left are the *deck controls*. They behave like controls on a typical tape recorder. In the lower left corner is the *tempo control*, which sets the speed or playback rate of the sequence. And finally, the right half of the page has the *track controls*. These are eight duplicate sets of controls, one set for each of the eight tracks used by the sequencer.

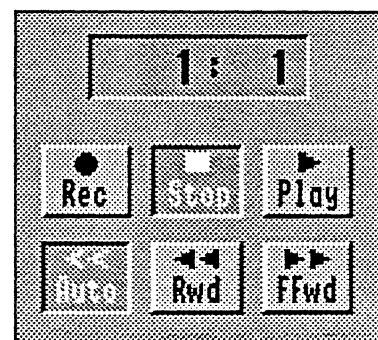


The Deck Controls

If you're familiar with how to control a tape recorder, then you should find these controls very straightforward. Play, Stop, Rec, Fast-Forward, Rewind and Auto-Rewind all work the same as their tape deck counterparts. They control the basic sequencer functions. Above the deck control buttons is the position counter, which shows you where you are in the sequence.

Play

To play a sequence from memory, press the Play button. It starts at the current location as displayed by the counter. The sequencer will automatically stop when it reaches the end.



Deck Controls

Record

Pressing this button will start recording MIDI messages received at the MIDI port. In other words, if you have a keyboard connected to the MIDI port and MIDI is enabled (see “MIDI Setup”), then synthLAB will start recording everything you play. At the same time it can also play-back selected tracks while you record. Like the play function, recording starts at the position indicated by the counter.

Stop

Will immediately halt a play or record process.

Fast-Forward

If you want to advance the the counter forward, then press and hold this button until you reach the desired position. If you hold this button down while a sequence is playing, then the tempo will double and stay at that speed until you release the button.

Rewind

This button will advance the position counter backward. Double-clicking this button will reset the counter to the beginning of the sequence (1:1).

Auto-Rewind

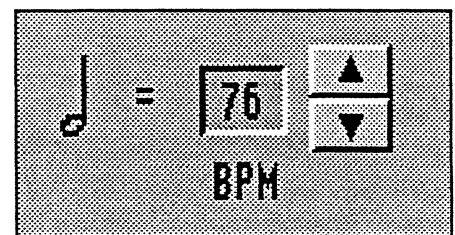
When this button is on, anytime the sequence stops it will automatically rewind to the beginning of the sequence. Otherwise, if it's off, then it remains at the current location.

Counter Display

The counter shows you the where you are positioned in the sequence. It displays the position in measures and beats. You can't directly edit the value, you move it forward or backward by using the deck buttons.

Tempo Control

This control sets the speed in which the sequencer plays. It sets this speed in units of half-note beats per minute. If you are used to working with quarter-note beats, then the quarter-note BPM value is twice the half-note value displayed. For example, when the tempo display shows 60, this is equivalent to 120 quarter-note beats per measure.



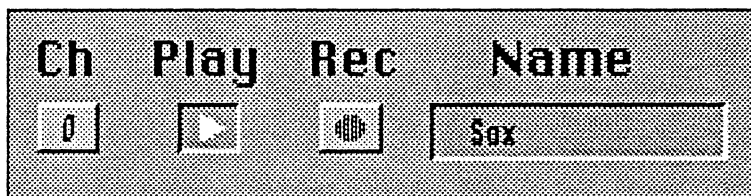
Tempo Control

Track Controls

The synthLAB sequencer is in many ways like an eight-track tape recorder. And like a multi-track tape deck, you can record one track of music at a time, using different instruments or combinations on each separate track. Then during play-back, you can select any or all of the tracks you wish to hear. You can play-back selected tracks at the same time that you are recording on another track. You can even over-dub any track until it sounds right.

But unlike a tape recorder, you are recording only MIDI messages, and not the actual sound being heard. When the sequencer plays, it sends these MIDI messages over to the synthesizer, which then reproduces the instruments you recorded in the first place. The advantage over a tape recorder is that you have the flexibility to experiment and re-orchestrate your sequence even after your done recording.

The track controls consist of eight duplicate sets of controls, one set for each track. Track number one is the top set, while the bottom most set is track eight. There are four controls for each track. You can give each track a name, select one track for recording, enable any or all tracks for play-back and finally, you can force any track to play a selected instrument.



Track Controls

Track Name

This control lets you reference the track by an entered name. You can put any type of notes or labels in here, which you may find useful. After clicking on the control, you'll get a line-edit type of dialog box. Type the name you want and click the "Done" button to exit. Up to 15 characters can be entered.

Track Record

Before you start recording, you have to tell the sequencer which one of the eight tracks it must record on. This button selects the record track. Note that synthLAB allows you to select only one track for this at a time. If you change the record track by pressing a new button, synthLAB will automatically turn the previous selected button off. Pressing a track record button while it's already on will turn it off, leaving no tracks selected for record.

Recording on a track that already has information on it will overwrite the previous data with new data.

Track Play

During a play or record process, individual tracks can be turned on or off for play-back by pressing these buttons. Unlike the track record buttons, any number or combinations of tracks can be selected for play-back. Clicking on a selected button will turn it off.

Track Channel

With this control, you can force all MIDI messages in the track to play a specific instrument. Since the sequencer always plays in "multi" mode (see the "ERS"), the channel part of the MIDI message specifies which one of the 16 instruments to play. This control makes all messages in the track to have one specified channel number, and thus one specified instrument. Channel one plays instrument number one, channel two plays instrument number two and so on. A channel number of zero will leave the message alone, letting it play whatever channel or instrument was originally recorded.

The reason you have to specify channel number is that it also affects the MIDI data that gets sent out the MIDI port, if it's enabled. All MIDI data sent out from the track will now be on the entered channel. .

When you click on the button, you'll get a dialog box with a control that lets you change the channel number. Press the up or down arrow keys to set it to the value you want, then exit.

Editing Instrument Records



The WaveList Edit and Envelope Edit pages are used to edit Instrument Records, which define the instruments used by MIDI Synth (see “MIDI Synth ERS” for details on Instrument Records). Remember, each Instrument Record has two Generators (Gen 1 and Gen 2), and each Generator has one Envelope Record and eight WaveList Records (WaveList 1-8). If you look at the two Edit pages, you’ll notice that they both have Generator select buttons (Gen 1, Gen 2). You use this to select which one of the two Generators you wish to work on. In the Envelope Edit page, it selects one of the two Envelope Records, while in the WaveList Edit page, this selects which of the two sets of eight WaveList Records you want. You use the WaveList menu button to further select which specific WaveList to edit while in the WaveList Edit page.



*Gen Select
Buttons*

Since each synthLAB instrument bank has 16 instruments, you first have to select which instrument you want to edit. The Instrument menu button lets you make this selection. Pull it down and select the instrument. Next, you select a Generator by clicking one of the Generator select buttons. Finally, if you are editing a WaveList parameter, you need to select a WaveList from the WaveList menu button.

Before editing a WaveList, check the *Top Key* parameters to make sure that you have the correct WaveList selected for the desired note range. This is the most common mistake that users make when editing WaveLists.

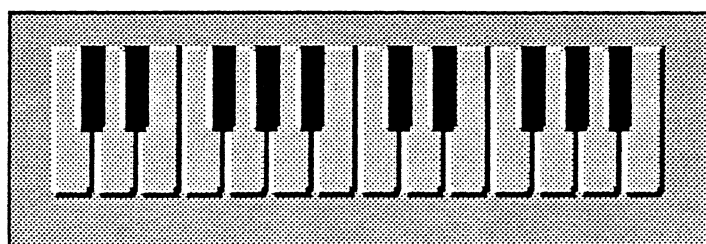
To get to any specific Instrument Record parameter, you need to:

- 1. Select the correct instrument from the Instrument menu.*
- 2. Select one of the two Generators with the Gen buttons.*
- 3. If it's a WaveList parameter, then select the correct WaveList number from the WaveList menu.*

Because there is so much data to edit, this probably sounds confusing at first. Once you start working with it, this organization will start to make some sense to you.

Screen Keyboard

The two-octave screen keyboard will play whatever instrument is currently selected for edit (from the *Instrument* menu button). Clicking on the keys will play them. You can change the keyboard



Screen Keyboard

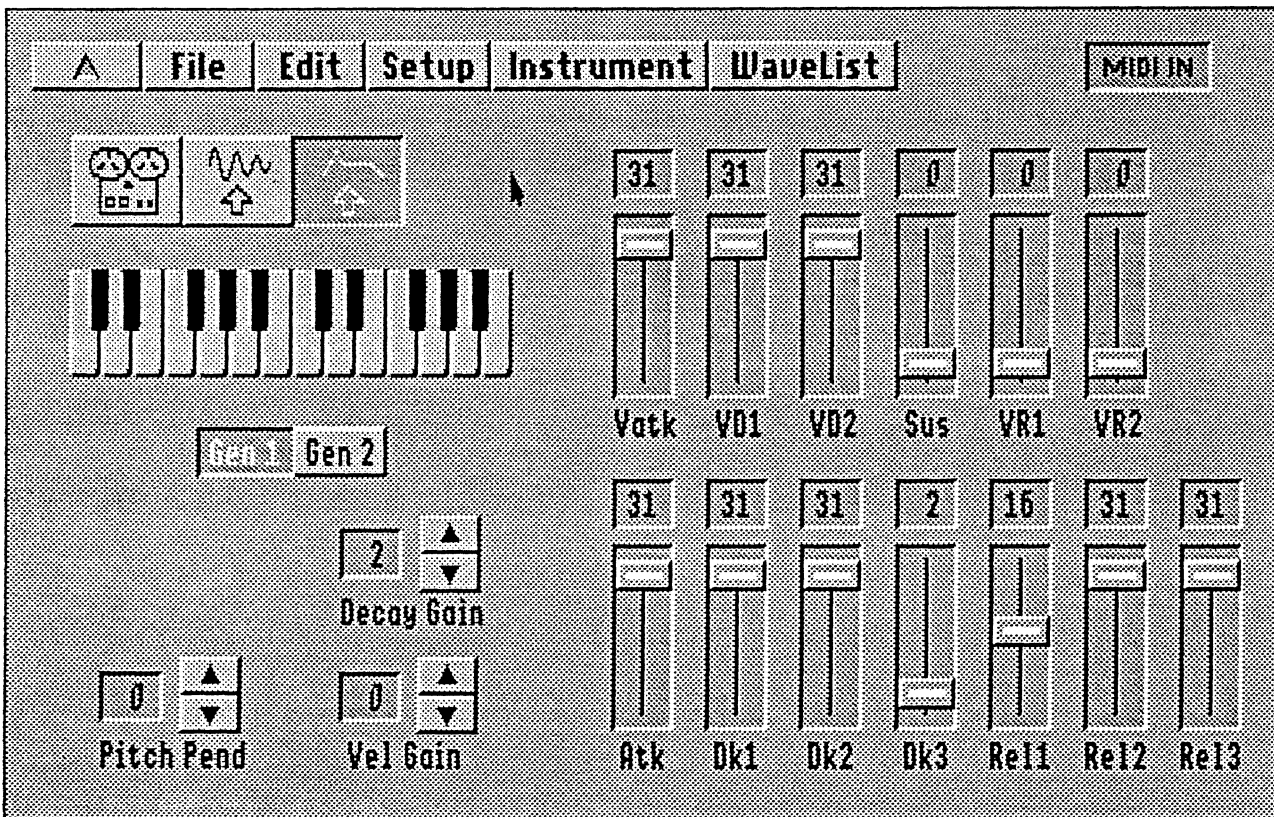
range and volume from the *System Setup* menu item found under the *Setup* menu button on the top of the screen (see “Setup Menu”).

If you click on the keyboard while you’re recording a track, synthLAB will record those notes.

Envelope Edit Page



This page lets you edit Envelope Records (see “MIDI Synth ERS” for information on Instrument Records). There are two independent envelopes for each instrument, one for Gen 1 and another one for Gen 2. The sliders found on the right set the envelope parameters, while the buttons on the bottom left control the remaining parameters found in an Envelope Record.



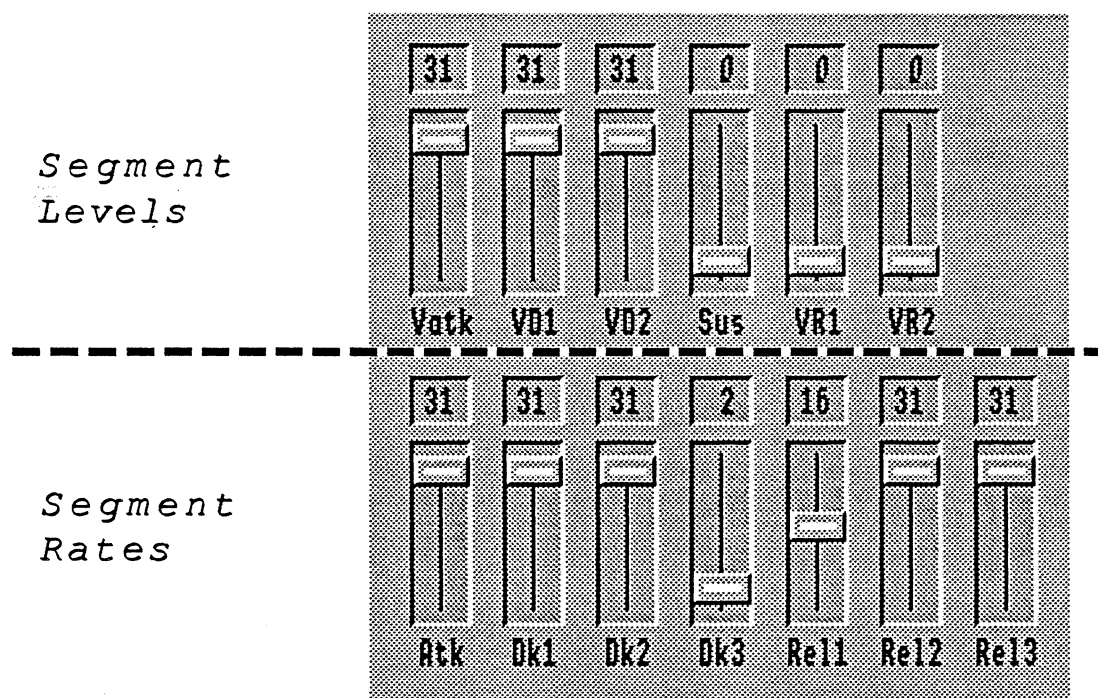
Envelope Sliders

These sliders let you edit the eight envelope segments. The top six sliders set the **level** values while the bottom seven sliders set the **rates**.

Here's a brief summary of the envelope (see “Instrument Records” in the “ERS” for details):

1. Start ramping from zero at *Atk* rate to *Vatk* level.
2. Ramp at *Dk1* rate to *VD1* level.
3. Ramp at *Dk2* rate to *VD2* level.
4. Ramp at *Dk3* rate to *Sus* level.
5. Hold at *Sus* level until “Note Off”.
6. Ramp at *Rel1* rate to *VR1* level.
7. Ramp at *Rel2* rate to *VR2* level.
8. Ramp at *Rel3* rate to a zero level.

If any level is set to zero, then the envelope ends at that point. The level sliders display 1/4th the real value. In other words, a level slider value of 25 sets a value of 100 into the Envelope Record.



Velocity Gain

This sets the envelope's sensitivity to MIDI velocity data (see "Instrument Records" in the "ERS" for details). Note that the Gen 1 control affects the entire instrument; both the Gen 1 envelope and the Gen 2 envelope.

Decay Gain

Increases the the Decay rates for higher pitched notes (see "Instrument Records" in the "ERS" for details).

Pitch Bend

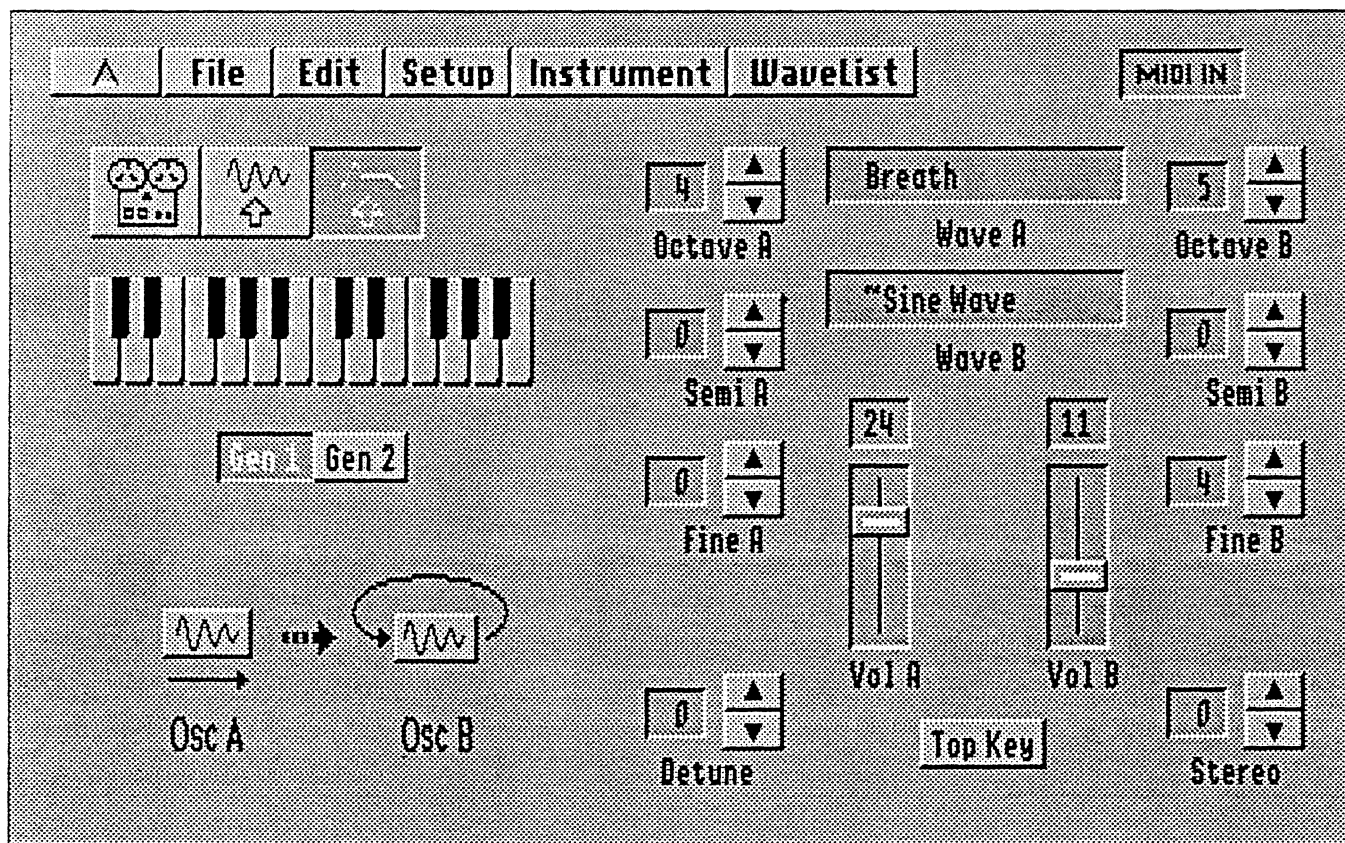
This sets the instrument's sensitivity to MIDI pitch bend data (see "Instrument Records" in the "ERS" for details). Like Velocity Gain, the Gen 1 control affects the entire instrument; both the Gen 1 envelope and the Gen 2 envelope.

WaveList Edit Page



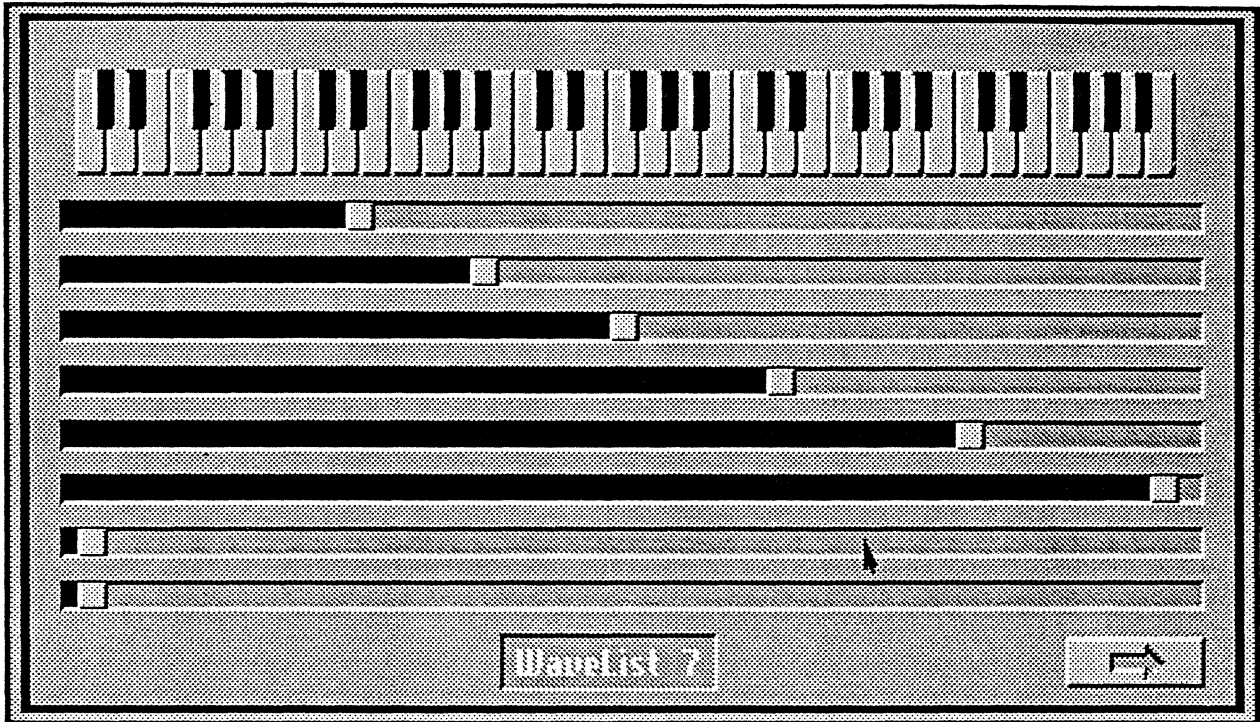
This page lets you edit WaveList Records (see "MIDI Synth ERS" for information on WaveList Records). There are 16 WaveLists for each instrument, eight for Gen 1 and another eight for Gen 2. Make sure that you have the correct WaveList selected from the WaveList menu button whenever you are editing in this page.

Keep in mind that there are four oscillators for each instrument, two for Gen 1 and two for Gen 2. The two oscillators (Osc A and Osc B) in each generator are independently controlled. Each oscillator can have its tuning (*Octave*, *Semi* and *Fine*), volume, wave and configuration set from this page.



Top Key

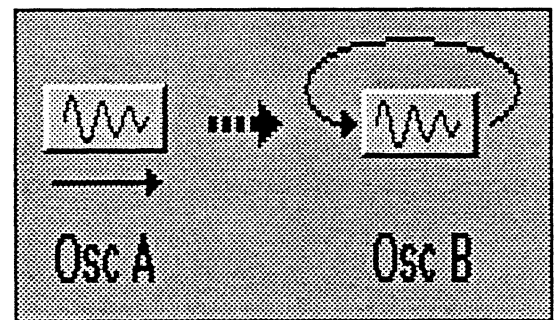
Each one of the eight WaveLists can be made active to play in only a specific range of notes. This allows you to create "multi-sampled" instruments with a different wavesample for each WaveList (see WaveList Records in the "ERS" for details). Clicking the Top Key button will display a dialog box with eight horizontal sliders, one for each WaveList. Remember that the Top Keys are prioritized in order from WaveList 1 to WaveList 8. This means that WaveList 2 starts from where WaveList 1 ended, WaveList 3 starts from where WaveList 2 ended, and so on.



Top Key Dialog

Oscillator Configuration

The two oscillators in each generator can be set to any one of six different modes or *configurations* (see WaveList Records in the “ERS” for details). This is a very powerful feature of MIDI Synth since it gives you great flexibility in controlling how the two oscillators interact with each other. Clicking on the control will step to the next configuration (Type 0 thru Type 5).

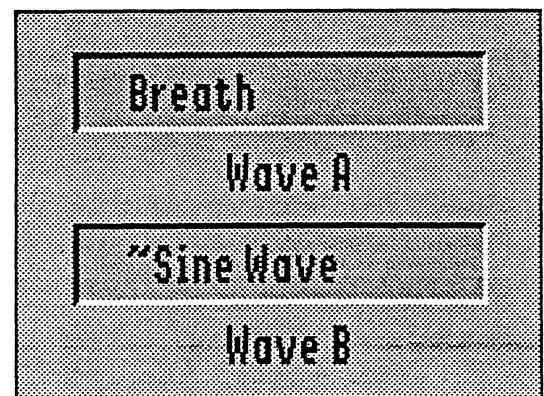


Type 3 Configuration

Wave Select

Holding down the mouse button on either of the two *Wave Select* controls (Wave A and Wave B) will produce a pop-up menu of available waves from the current Wave File.

The convention used by synthLAB to notate the difference between wave types is to start ‘loop’ type wave names with the character “~”. For example, if you had two waves titled “Piano” and “~Piano”, the wave “Piano” would be a sampled recording of a piano attack while the wave “~Piano” would be a single-cycle wave



Wave Select Controls

of the piano's sustain and release. The "Piano" wave should be played with a 'one-shot' configuration and the "~Piano" wave should be played with a 'loop' type configuration. In this example, you may want to use Configuration *Mode 3* with Osc A playing the sampled "Piano" attack and Osc B playing the single-cycle "~Piano" wave.

Oscillator Tuning

There are several important reasons why you need to independently tune each individual oscillator. Since sampled waves are dependent on the input and output sampling rate and on the pitch of the original source, they most probably are not going to play back at the correct pitch. If you're using multi-sampled sounds, they'll need to be tuned to the correct octave. So the most obvious use is to tune the waves to the correct pitch.

Many effects can be created by altering the tuning ratio between different oscillators. If you fine tune two "loop" mode oscillators with a slight difference in pitch, you get an interesting effect of "movement" while the two oscillators slowly beat against each other. When you tune the two oscillators further apart in pitch, the sound's "texture" increases giving a richer and thicker sound. If you slightly detune two oscillators playing the same sampled sound, you get the familiar "phasing" effect.

There are three tuning controls for each oscillator. The Octave control is centered around a normalized value of 3. This means that single-cycle waves will produce the correct pitch (middle A is 440 hz) when The Octave is set to 3, and both Semi and Fine are set to zero. The Semi control increases the pitch by semi-tone increments and the Fine control increases the pitch by fractions of a semi-tone.

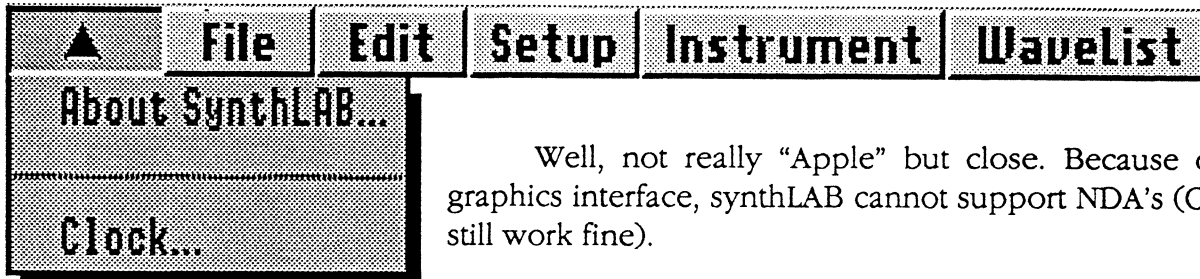
Oscillator Volume

These sliders let you set each oscillator's output level. If you want to turn an oscillator off, set the *Volume* to zero. Like the Envelope sliders, these sliders display 1/4th the real value set in the Instrument Record.

There are a couple of points you should keep in mind when you're making instruments. Since sampled waves usually have a recorded envelope in them, their output volume will sound lower than single-cycle waves. So if you're mixing the two types, you'll usually need to play single-cycle waves at a lower level to get the correct balance.

There's a slight amount of cross-talk between Osc A and Osc B. If you're not using one of the oscillators (zero volume), make sure that you select a wave for the unused oscillator that has very little high harmonic content (like a sine wave). Also, tune the unused oscillator to a low frequency. Even though the volume is set to zero, you may still be able to hear the oscillator if its selected wave is very bright or at a high pitch.

“△” Menu

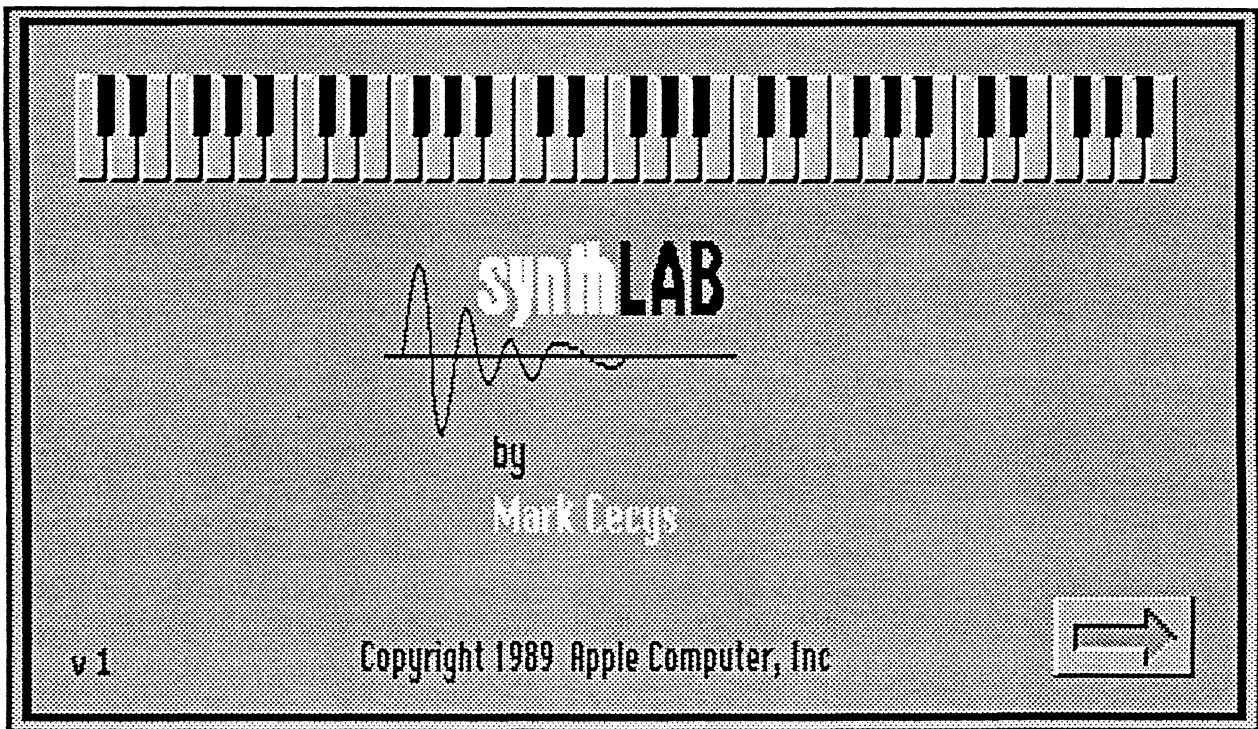


Well, not really “Apple” but close. Because of its graphics interface, synthLAB cannot support NDA’s (CDA’s still work fine).

This interface is a result of the author’s personal choice, and does not reflect on MIDI Synth compatibility with the “Mac® desktop” interface. MIDI Synth is optimized for speed, so it works well with any desktop/non-desktop interface.

About SynthLAB..

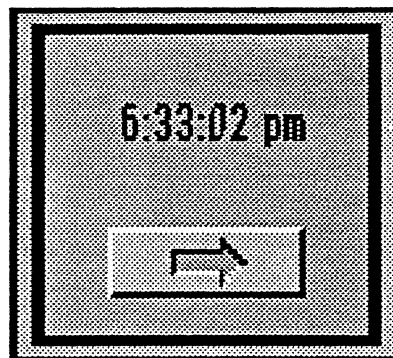
Selecting this item brings up the “player keyboard” dialog box. The “keyboard” will play whatever sequence is currently in memory. The displayed number on the lower left side of the dialog box shows the program version number.



About synthLAB... Dialog

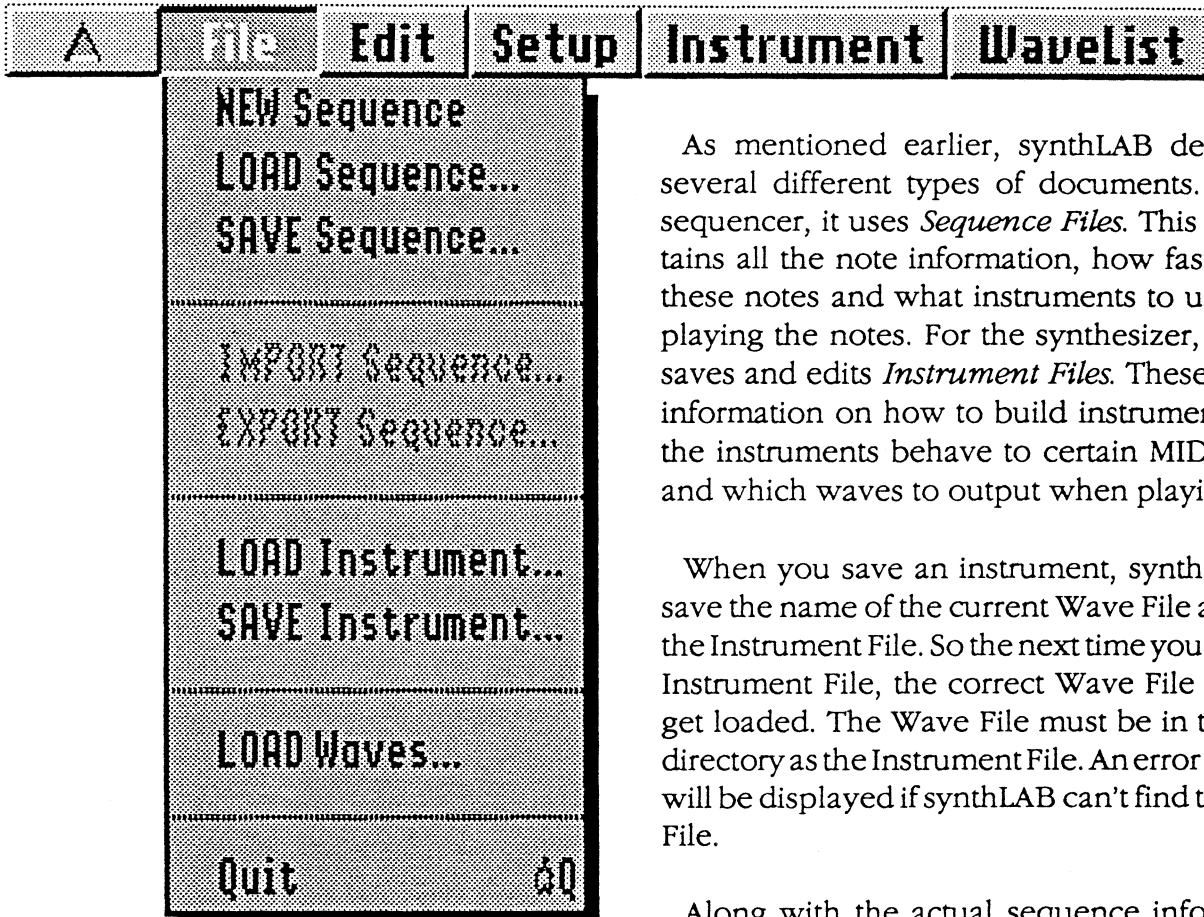
Clock...

Even if you can't use your favorite NDA's, you still have your clock. This impressive "synthLAB NDA" alone is well worth the price you paid for synthLAB.



Clock

File Menu



As mentioned earlier, synthLAB deals with several different types of documents. For the sequencer, it uses *Sequence Files*. This file contains all the note information, how fast to play these notes and what instruments to use when playing the notes. For the synthesizer, it loads, saves and edits *Instrument Files*. These contain information on how to build instruments, how the instruments behave to certain MIDI events and which waves to output when playing.

When you save an instrument, synthLAB will save the name of the current Wave File as part of the Instrument File. So the next time you load this Instrument File, the correct Wave File will also get loaded. The Wave File must be in the same directory as the Instrument File. An error message will be displayed if synthLAB can't find the Wave File.

Along with the actual sequence information, Sequence Files also contain the name of the Instrument File used when the Sequence File was saved. When loading the Sequence File, synthLAB will search the same directory for the Instrument File (along with its Wave File). It's important to understand that synthLAB does not save the Instrument File when it saves a Sequence File, it only saves the name of the current Instrument File as part of the Sequence File. If you changed some instruments and plan on saving them under a new file name, make sure to save the new Instrument File first, before saving the sequence. This way the Sequence File will have the name of your new Instrument File, since the sequence will be saved after you gave the Instrument File a new name. Otherwise synthLAB will still load the old instruments in whenever that sequence gets loaded.

As you can see, synthLAB expects Sequence, Instrument and Wave Files to all be in the same directory. Loading a Sequence File will automatically cause an Instrument File to be loaded, which in turn will automatically cause a Wave File to be loaded. If these files are not in the same directory, you'll have to load them in manually. For example, If your Instrument and Wave Files are in a different directory from your sequences, after loading in a sequence, you'll get an error message telling you that synthLAB couldn't find the appropriate Instrument File. This message is useful since it gives you the name of the Instrument File that goes with the sequence. At this point, you'll have to select the "Load Instrument..." item from the "File" menu and find the correct Instrument File.

New Sequence

Selecting this item will set the sequencer to default values. This is used if you're making a new sequence from scratch. Be careful with this selection because it will erase any sequence currently recorded in memory.

All eight tracks will be given default names ("Track 1" - "Track 8") and track channels will be set to 'thru' (0) mode. Track 1 will be set to both record and play with the counter reset to the first beat (1:1). Finally, all 16 instrument volumes (see "Volumes..." in the menu "Setup" section) will be set to maximum values.

New Sequence does not affect the tempo value, metronome, count-off, key start, beat value, beats per measure and the sequencer clock source. Nor does it affect the instruments in any way.

Load Sequence...

After selecting this item, select a Sequence File from the displayed dialog box. The appropriate instruments and waves will also automatically be loaded along with the sequence. Remember, synthLAB expects the Instrument and Wave Files used by the sequence to be in the same directory as the sequence. If they're not, you'll get an error message telling you what file synthLAB couldn't find.

All the information that was saved (see "Save Sequence...") with the sequence will be restored when the Sequence File is loaded. Loading a sequence will over-write any current sequence you may have in memory. So be sure to save your old sequence before loading in a new one.

synthLAB can't handle sequences which are larger than 128k bytes in size. If you're loading sequences made by synthLAB, this should not be a problem. If you're importing sequences from other programs, make sure that they are less than this amount.

Save Sequence...

This item brings up the standard dialog box for saving files. Type in the name of your Sequence File at the bottom and click on the Save button. If you want synthLAB to automatically load your instruments in the next time you load this sequence, make sure that you save the sequence in the same directory where the Instrument and Wave Files are located. synthLAB will save the current Instrument File name along with the sequence, and tries to load these instruments whenever the sequence is loaded.

Besides saving all the note and MIDI information in your sequence, all track names, track channel, track play, track record, instrument volumes, beat value, beats per measure and the tempo value will also be saved with the sequence (see section "Sequence File Format").

Import/Export Sequence

These two items are not currently implemented. They may be implemented in a future version of synthLAB.

Load Instrument..

After selecting this item, select an Instrument File from the displayed dialog box. The appropriate Wave File will also automatically be loaded along with the instruments. Remember, synthLAB expects the Wave File to be in the same directory where the Instrument File is located. If it's not, you'll get an error message telling you the Wave File name synthLAB couldn't find.

Save Instrument..

This item brings up a dialog box for saving Instrument Files. Type in the name of your Instrument File at the bottom and click on the Save button. If you want synthLAB to automatically load the Wave File in the next time you load these instruments, make sure that you save the Instrument File in the same directory where the Wave File is located. synthLAB will save the current Wave File name along with the instruments (see section "Instrument File Format"), and tries to load this Wave File whenever the instruments are loaded.

Load Waves...

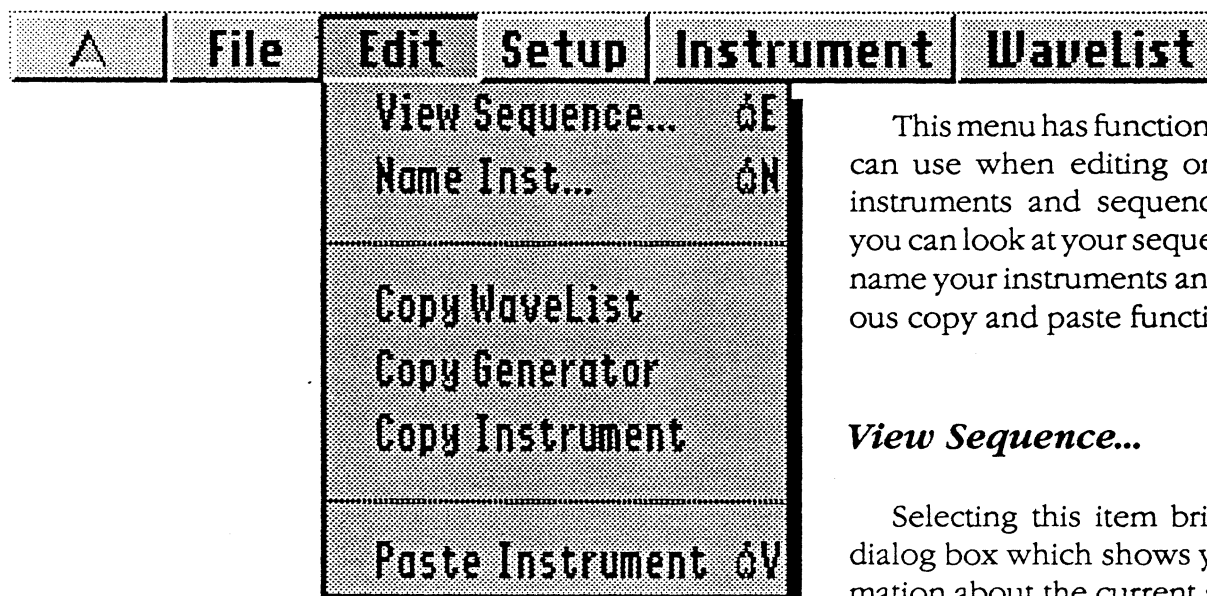
This item can be used to load Wave Files located in different directories from their Instrument Files. If synthLAB can't find the Wave File when loading instruments, it will display an error message telling you the name of the Wave File needed. You can then select this menu item and load the correct Wave File from its directory.

If you load a new Wave File that is different from the one that was used when the instruments were created, you'll need to redo most of the instrument controls to work with the new waves. In other words you'll have to build the instruments from scratch.

Quit

Selecting this item will terminate synthLAB and return program control back to the launching application (usually the Finder). Before selecting this, make sure that you saved any changed sequence and instruments. If you don't, they'll be gone forever.

Edit Menu

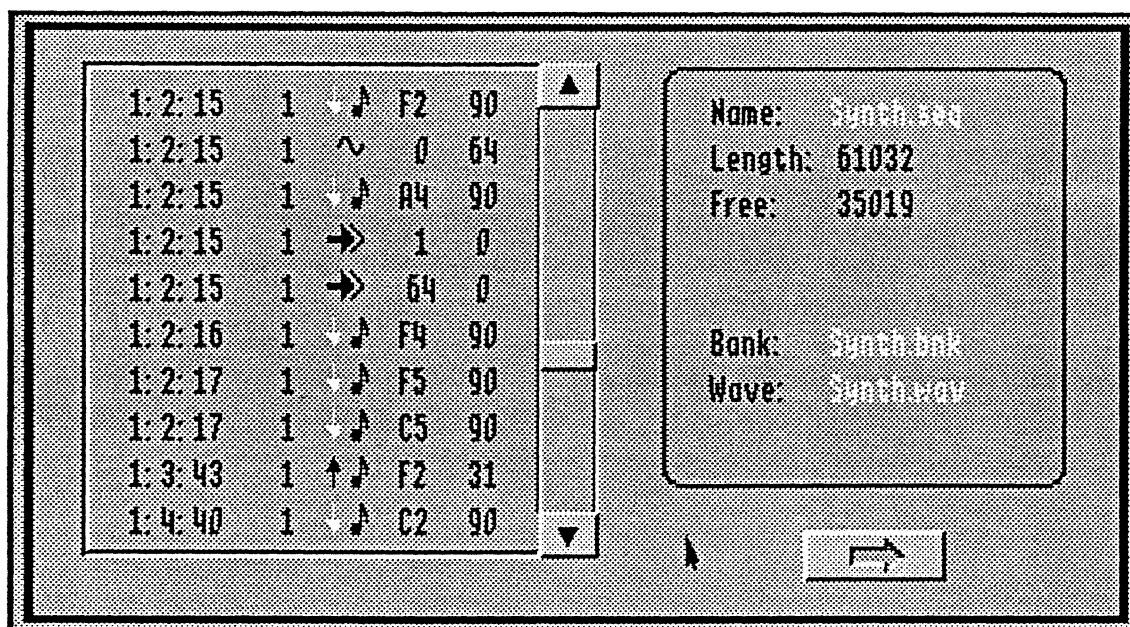


This menu has functions that you can use when editing or creating instruments and sequences. here, you can look at your sequence data, name your instruments and do various copy and paste functions.








View Sequence...

Selecting this item brings up a dialog box which shows you information about the current sequence in memory. On the right side you'll

see the name of the sequence (actually the name of the last loaded or saved sequence), its length in bytes and the number of bytes available in the seq buffer. Below this is the name of the current Instrument File (Bank) and the name of the current Wave File (Wave).



On the left side is a list control which shows you each individual seq item in your sequence (the scroll thumb does not function in the current version). The left-most value displayed in the seq item is the time-stamp shown in *measure: beat: remainder ticks* format. Remember that beats are based on a value of 96 ticks in a quarter note. So when displaying any seq item that does not fall on a beat, the remainder field will show the number of ticks past the last beat. If the beat is a

	Note Off - \$8x
	Note On - \$9x
	Poly After-Touch - \$Ax
	Control Change - \$Bx
	Program Change - \$Cx
	Mono After-Touch - \$Dx
	Pitch Wheel- \$Ex

MIDI Status Messages

entered as an instrument name.

Copy WaveList Copy Generator Copy Instrument

Selecting any one of these items will copy the specified parameters into an internal clipboard. This is used to duplicate the contents of a WaveList, Generator or Instrument. Once the parameters have been copied into the clipboard, you can then select another WaveList, Generator or Instrument, and paste those parameters into the new location.

A “Copy WaveList” selection will only copy the 16 parameters of the selected WaveList. Copying a Generator will copy one Envelope and eight WaveLists. Finally, when you copy an Instrument, all parameters from both Generators (both Envelopes and all 16 WaveLists), including the instrument name, will be placed into the clipboard. The original parameters will not be affected in any way by this.

Paste

Once a Copy function has been completed, you can use this item to place the contents of the clipboard into some other location. If the clipboard is empty, this item will be disabled. Whenever the clipboard has something in it, this item will show you what type of parameters that are in there. For example, if you copied a WaveList, the Paste item will read “Paste WaveList”. Selecting the Paste item will over-write the original parameters, so use this carefully.

quarter note, then the remainder will have a value between 0 and 95. An eight note beat will have remainders between 0 and 47, and so on.

Displayed to the right of the time-stamp, is the channel number. If you’re playing the synthesizer, then this is also the instrument number.

Finally, the remaining three fields on the right show the MIDI message and data bytes. The MIDI message status field is represented by any one of the symbols shown in the chart. Since only valid data bytes are shown, you may see one, two or no data bytes following the message status.

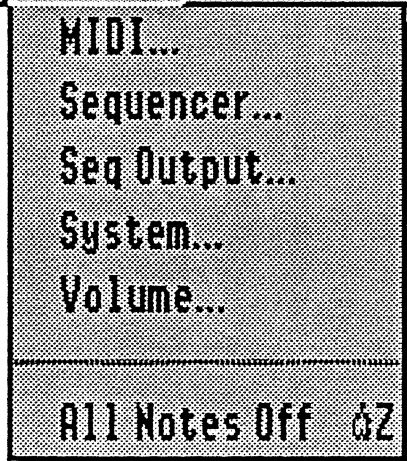
Name Inst..

This item lets you modify the name of the instrument which is currently selected for editing (selected from the Instrument menu). Up to 15 characters can be

Setup Menu



Whenever you start synthLAB, many of the program variables are set to their default values. Other program variables get set whenever you load a sequence. The Setup menu allows you to change these variables and to tailor the program environment to your specific needs.



MIDI...

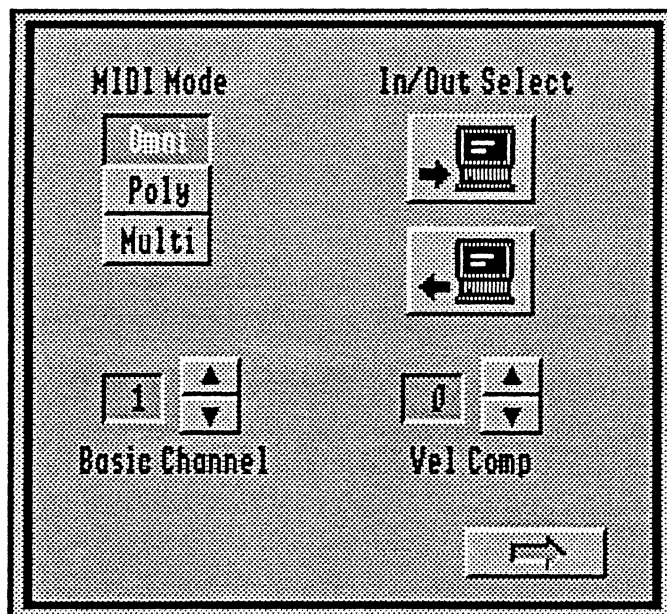
synthLAB always starts with MIDI disabled. If you have a MIDI interface connected to your IIGS, then you must first enable and configure the MIDI port from here, before synthLAB can respond to your MIDI keyboard.

A dialog box is displayed with controls for the various MIDI parameters. You can select the MIDI mode, enable MIDI input or output, enter the Basic Channel number and set the velocity compensation value (see the "ERS" for details on MIDI parameters).

The top button of the "In/Out Select" enables or disables the *MIDI In* function (note that in the icon, the arrow points IN to the computer, indicating the direction of MIDI data). When *MIDI in* is enabled you'll see the MIDI IN display (top right corner of the screen) become active. This display will light-up whenever a MIDI message is received at the MIDI port.

The lower button enables or disables the *MIDI Out* function. If you're not using this function, then make sure it is off (button is 'out').

If there's a problem with your MIDI setup, you'll get an error dialog instead of the setup dialog. See Appendix B for configuring MIDI correctly into your system.

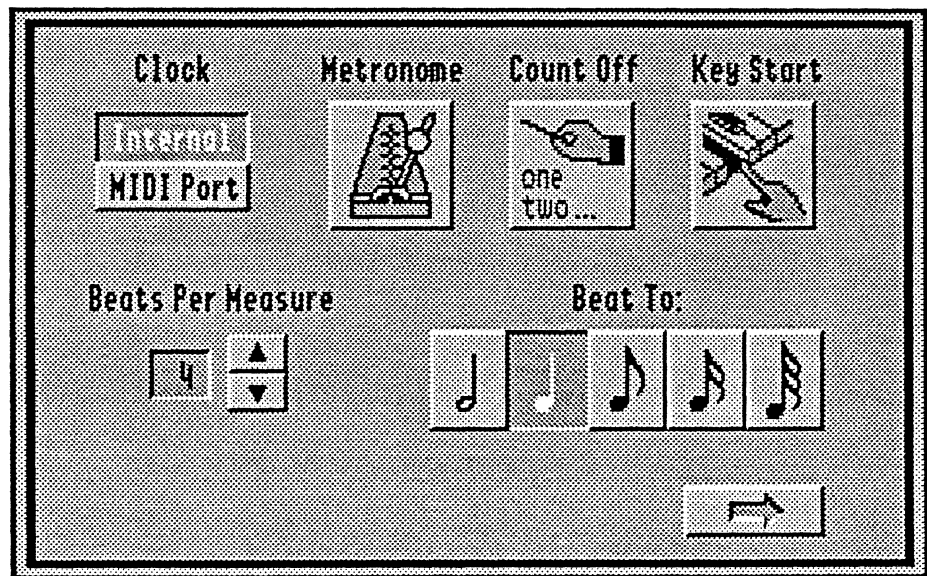


MIDI Setup Dialog

Sequencer...

Selecting this item displays a dialog box that has controls which affect the sequencer operation. The bottom two controls specify the values for measures and beats. The counter display, metronome and tempo control are all affected by these.

The Metronome will cause an audible woodblock tick to sound on every beat whenever you play or record with the sequencer. When Count Off is enabled, the sequencer counts off one measure before starting to play or record. You can force the sequencer to wait until it receives a MIDI note off message thru the MIDI port before it starts playing or recording by enabling the Key Start control.



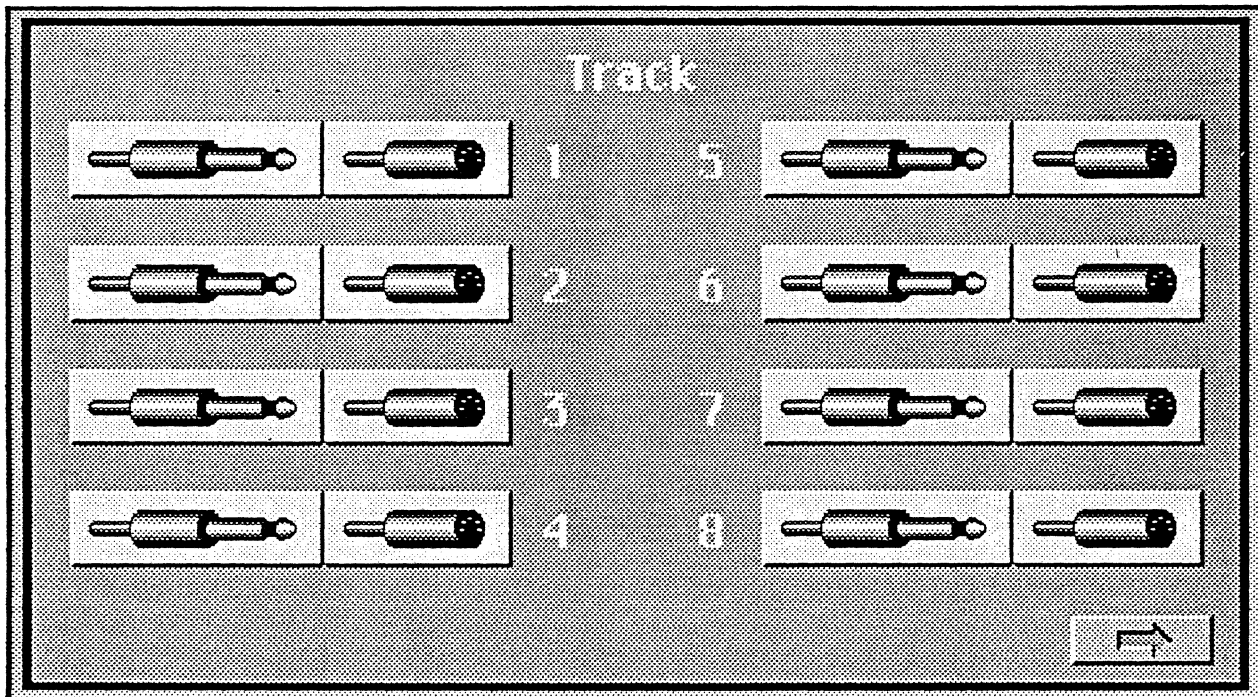
Sequencer Setup Dialog

Normally you use the Internal clock to advance the sequencer timer. This means that whenever you start the sequencer, it starts to increment its timer based on the current tempo value. If you need to synchronize the sequencer to an external MIDI device (like a drum machine or another sequencer), you can slave synthLAB thru MIDI by selecting the MIDI Port clock. Note that synthLAB can only receive MIDI timing clocks, it does not send them. This means that synthLAB can be slave device and not a master.

Seq Output...

With synthLAB (and MIDI Synth), you can specify where to send the output from each track whenever the sequencer is playing. You can have the sequencer for example, play only the synthesizer on tracks 1,2 and 5, while playing only a MIDI device on track 4, and have it play both the synthesizer and the MIDI device on track 3. This sequencer output configuration is mapped from the dialog box which is displayed whenever this menu item is selected.

Think of the sequencer as a box with eight audio jacks which can connect to the synthesizer and eight MIDI ports which can connect to your external MIDI device, one set for each track. This dialog box has two controls for each track. The left button with the "audio plug" icon connects the sequencer track to the synthesizer, while the "MIDI plug" button connects the track to the MIDI port.



Sequencer Output

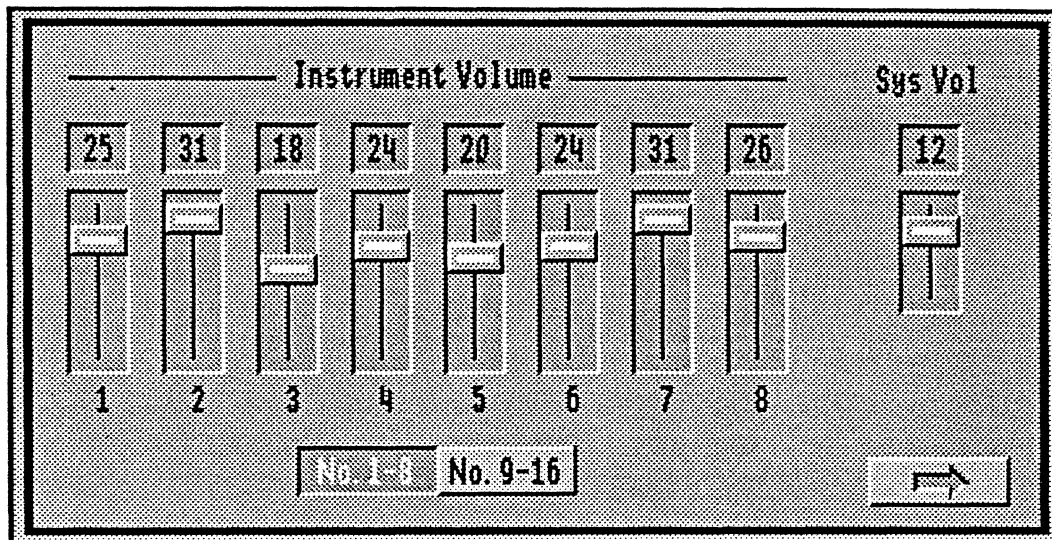
If you're using MIDI output, make sure that the *MIDI Out* button in the MIDI Setup dialog is on. This button acts as a master control, enabling or disabling MIDI out globally on all tracks.

System...

This item lets you adjust the pitch and volume of the screen keyboard found on the WaveList and Envelope pages.

Volume...

The volume level of each one of the 16 instruments can be adjusted independently from this dialog box. These controls are intended to be used with the sequencer to balance instrument levels when playing many instruments together. These settings are saved in the Sequence File, and are restored whenever you load a sequence.



Volume Setup Dialog

The System Volume control sets the overall output level for the headphone jack on the back of your IIGS. It will have no effect on the volume if you're using a stereo interface card (you'll have to use the volume control on your stereo amplifier).

All Notes Off

This is the equivalent of the "panic" button found on some synthesizers used to kill any hung notes (notes that never got an off message). Selecting this item will send a 'note off' message to all voices active in the synthesizer.

Appendix A

synthLAB File Formats

Instrument File Format



File Type: \$D6

Aux Type: \$0001

Instrument File - \$1600 bytes

\$0000	Instrument Header
\$0400	Instr #1 Def Record
\$0520	Instr #2 Def Record
.	.
.	.
\$14E0	Instr #16 Def Record

Instrument Header - \$400 bytes

\$000	Long	File Type: "INST"
\$004	Word	Version: \$0100
\$006	Word	Header Size (\$400)
\$008	Str[16]	Owner application name*
\$018	Str[16]	Wave File name
\$028	Byte	Master Semi-tone tuning*
\$029	Byte	Master Fine tuning*
\$02A	Byte	Reserved
\$02B	Byte	Master Volume (0-15)*
\$02C	Byte	Number of instruments (1-16)*
\$02D	Word	Reserved
\$030	Bytes[512]	Wave Ref Block
\$230	Bytes[208]	Free
\$300	Str[16]	Instrument #1 name
\$310	Str[16]	Instrument #2 name
.	.	.
.	.	.
\$3F0	Str[16]	Instrument #16 name

* not used by synthLAB

Instrument Def Record - \$120 bytes

\$0000	Envelope Record	
\$0010	WaveList Record 1	
\$0020	WaveList Record 2	
.		
.		
\$0080	WaveList Record 8	Generator 1
\$0090	Envelope Record	
\$00A0	WaveList Record 1	
\$00B0	WaveList Record 2	
.		
.		
\$0110	WaveList Record 8	Generator 2

Envelope Record - \$10 bytes

\$00	Byte	Attack level
\$01	Byte	Attack rate
\$02	Byte	Decay 1 level
\$03	Byte	Decay 1 rate
\$04	Byte	Decay 2 level
\$05	Byte	Decay 2 rate
\$06	Byte	Sustain level
\$07	Byte	Decay 3 rate
\$08	Byte	Release 1 level
\$09	Byte	Release 1 rate
\$0A	Byte	Release 2 level
\$0B	Byte	Release 2 rate
\$0C	Byte	Release 3 rate
\$0D	Byte	Decay Gain
\$0E	Byte	Velocity Gain**
\$0F	Byte	Pitch Bend**

WaveList Record - \$10 bytes

\$00	Byte	Top Key**
\$01	Byte	Configuration
\$02	Byte	Channel
\$03	Byte	Detune
\$04	Byte	Wave Address A
\$05	Byte	Wave Size A
\$06	Byte	Volume A
\$07	Byte	Octave Tuning A
\$08	Byte	Semi-tone Tuning A
\$09	Byte	Fine Tuning A
\$0A	Byte	Wave Address B
\$0B	Byte	Wave Size B
\$0C	Byte	Volume B
\$0D	Byte	Octave Tuning B
\$0E	Byte	Semi-tone Tuning B
\$0F	Byte	Fine Tuning B

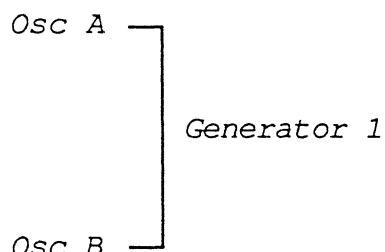
** Generator 1 value used for both Generators

Wave Ref Block - \$200 bytes

\$000	Instr #1 Wave Ref
\$020	Instr #2 Wave Ref
•	
•	
\$1E0	Instr #16 Wave Ref

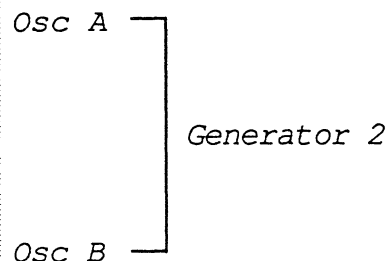
Wave Ref - \$20 bytes

\$00	Byte	WaveList 1 wave number
\$01	Byte	WaveList 2 wave number
•	•	•
•	•	•
\$07	Byte	WaveList 8 wave number



\$08	Byte	WaveList 1 wave number
\$09	Byte	WaveList 2 wave number
•	•	•
•	•	•
\$0F	Byte	WaveList 8 wave number

\$10	Byte	WaveList 1 wave number
\$11	Byte	WaveList 2 wave number
•	•	•
•	•	•
\$17	Byte	WaveList 8 wave number



\$18	Byte	WaveList 1 wave number
\$19	Byte	WaveList 2 wave number
•	•	•
•	•	•
\$1F	Byte	WaveList 8 wave number

wave number = \$00-\$3F (see Wave File format)

Wave File Format



File Type: \$D8
Aux Type: \$0004

Wave File - \$10900 bytes

\$0000	Wave Header Record
\$0100	Wave Def Record #1
\$0120	Wave Def Record #2
	•
	•
\$08E0	Wave Def Record #64
\$0900	PCM Data

Wave Header Record - \$100 bytes

\$00	Long	File Type: "WAVE"
\$04	Word	Version: \$0100
\$06	Word	Offset to Wave data
\$08	Str[16]	Owner application name*
\$18	Word	Number of valid Wave Defs
\$1A	Bytes[230]	Free*

Wave Def Record - \$20 bytes

\$00	Str[16]	Wave name
\$10	Word	DOC Address
\$12	Byte	Set to zero
\$13	Byte	Size
\$14	Byte	Volume*
\$15	Byte	Octave Tuning*
\$16	Byte	Semi-tone Tuning*
\$17	Byte	Fine Tuning*
\$18	Bytes[8]	Free*

* not used by synthLAB

PCM Data - \$10000 bytes

64k image of DOC ram.
Must not contain samples with a value of zero.

* Wave sizes

```
s256      equ 0
s512      equ 1
s1K       equ 2
s2K       equ 3
s4K       equ 4
s8K       equ 5
s16K      equ 6
s32K      equ 7
```

WHeader

```
    DC.B 'WAVE'      ; file type
    DC.W $0100      ; version = 1.00
    DC.W 2048+256   ; offset to wave data (256 bytes for header,
;                                     2k bytes for Wave Defs)
    DC.B 'SynthLAB'; owner (your application name)
    DS.B 8          ; filler (owner field is 16 bytes)
    DC.W 1          ; # of valid Wave Defs
    DS.B 230       ; free (your app can use this)
```

WaveStr1

```
    str 'Flute'      ; name
    DS.B 16-(*-WaveStr1); filler (Wave Name field is 16 bytes)
    DC.W $2000       ; DOC ram address
    DC.B 0           ; reserved, set to zero
    DC.B s4K        ; size, 4k bytes
    DC.B 0           ; vol- not used
    DC.B 0           ; oct- not used
    DC.B 0           ; semi- not used
    DC.B 0           ; fine- not used
    DS.B 8           ; your app can put additional info here
```

*MPW IIGS Example of Wave File
(Header and Wave Def)*

Sequence File Format



File Type: \$D5

Aux Type: \$0001

Sequence Header

\$000	Long	File Type: "MSEQ"
\$004	Word	Version: \$0100
\$006	Word	Offset to Seq Data
\$008	Str[16]	Owner application name
\$018	Str[16]	Instrument #1 File name
\$028	Str[16]	Track #1 name
\$038	Str[16]	Track #2 name
.	.	.
.	.	.
\$118	Str[16]	Track #16 name

Sequence Initialization

70 \$46
 4 \$0A
 1 \$01
 96 \$60

\$128	Word[16]	Channel Volumes
\$148	Word[16]	Track Channels
\$168	Word[16]	Track Play
\$188	Word	Record Track
\$18A	Word	Tempo
\$18C	Word	Beats per Measure
\$18E	Word	Beat Value
\$190	Word	Ticks per Beat
\$192	Word[16]	Track Output Map
\$1B2	Word[16]	Reserved

Seq Data

Image of MIDI Synth Sequence.
 Must not exceed 128k length.
 Must terminate with EOS (\$FFFF).

HeadStart

```
DC.B 'MSEQ'      ; file type
DC.W $0100      ; version = 1.00
DC.W HeadEnd-HeadStart  ; index to seq data (header size)
str 'SynthLAB'  ; owner
DS.B 8         ; filler (owner field is 16 bytes)
str 'Synth.bnk'; instrument
DS.B 7,0       ; filler (instrument field is 16 bytes)
str 'the Demo'
DS.B 8,0       ; filler (track name field is 16 bytes)
str 'Track 2'
DS.B 9;0
.
.           ; insert names for Tracks #3 - #15
.
str 'Track 16'
DCB.B 8,0
```

* Seq Initilization

```
DCB.W 16,127    ; Channel volumes = $7F
DCB.W 16,-1     ; Track channels = 'Thru' mode
```

```
DC.W -1,0,0,0,0,0,0,0,0
DC.W 0,0,0,0,0,0,0,0,0; play Track #1 only
```

```
DC.W 2          ; Record on Track #2
DC.W 60         ; tempo
DC.W 4          ; beats/measure
DC.W 1          ; beat to: (quarter note)
```

```
;           This is used to hilite the "Beat To:" buttons
;           in the Seq Setup dialog.
```

```
;           Not needed to play a seq with MIDI Synth.
;           0= half note  1= quarter  2 = 1/8th, etc.
```

```
DC.W 96         ; ticks/beat = quarter note
```

```
DCB.W 16,0      ; Output map
DCB.W 16,0      ; reserved
```

HeadEnd

*MPW IIGS Seq File Example
(Header and Initialization)*

Appendix B

The MIDI CDEV

The MIDI CDEV was created to encourage developers to specify MIDI in a uniform manner inside their applications. Whenever a user runs a MIDI program for the first time, the user usually has to input information about their specific MIDI setup through some sort of dialog box. When they run another MIDI program, they must give that program the exact same information all over again. The problem occurs when the user changes their MIDI setup. Now they must enter this new setup into all their MIDI programs, one by one. Another problem is that the user might not remember exactly what their setup is a year later when they buy a new MIDI program.

With the MIDI CDEV, the user enters their specific MIDI setup only once through the NDA Control Panel. The MIDI CDEV creates a setup file (*Midi.Setup*) inside the "Drivers" folder which contains information about the user's MIDI connection. From there, all MIDI applications can use the *Midi.Setup* file to configure themselves.

Installing the MIDI CDEV

1. Copy the file called "MIDI" from the synthLAB disk to the "CDevs" folder, which is found inside the "System" folder on your boot disk. You have now added a new selection to the NDA Control Panel called *MIDI*.

2. From the Finder, select the "Control Panel" NDA from the Apple Menu. Scroll down the selection list until you find the icon called *MIDI* (shown in Fig. 1 on the left side). With your mouse, click on it to select it.

3. Once MIDI is selected, you'll get the setup options, shown in Fig. 1 on the right. The options on the top select where the MIDI interface is connected to your GS. The list on the bottom lets you select the appropriate MIDI Driver file.

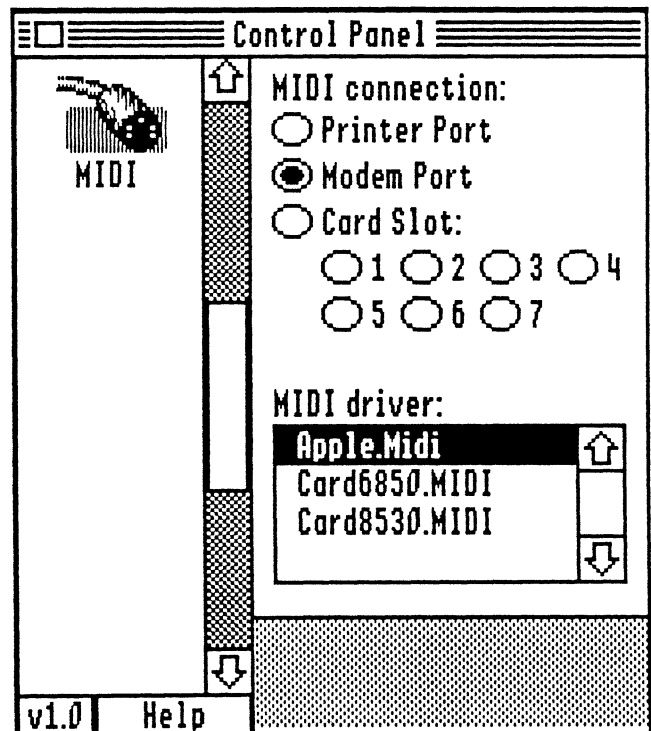


Fig. 1 - MIDI CDEV

MIDI connection

If you're using an external type of MIDI interface (*Apple MIDI*) that connects to a serial port on the back of your GS, then only select either the Printer or Modem ports. If you are using an internal card as a MIDI interface, such as the Applied Engineering *Audio Animator*, then select the "Card slot" button and one of the seven slot numbers.

MIDI Driver

The MIDI CDEV scans your "Drivers" folder and displays all of the MIDI type of driver files it finds there. From this list, you must select the appropriate driver for your interface. If there are no drivers shown in the list then consult your MIDI interface manual for instructions.

Midi.Setup File Format

WORD	External/Internal (0,1)
WORD	Slot number (1-7)
Str[32]	MIDI Driver file name Pascal string. (Not a pathname, file name only. Assumes path of */System/Drivers/)

MIDI SynthTM

External ERS

by
Mark Cecys

Table of Contents

Introduction	3
Limitations	4
Overview	4
Starting MIDI Synth	5
Interrupts	6
MIDI Synth Oscillators and Voices	6
Using MIDI Synth with the Sound Tool Set	6
MIDI	7
MIDI Data Flow	9
MIDI Input Path	9
Seq Output Path	10
Other MIDI Paths	10
Voice Architecture	12
Instrument Records	14
Envelope Record	14
Attack, Decay, Sustain and Release.....	14
Decay Gain	17
Velocity Gain	17
Pitch Bend	17
Wavelist Record	18
Top Key	18
Osc Config	19
Stereo	21
Wave Address and Size	21
Wave Volume	23
Osc Tuning	23
Note Tuning Table	23
Seq Record	24
Time-Stamps	25
Non-MIDI Messages	25
Call-back Routines	26

<i>Tool Call Reference</i>	29
MSBoot	31
MSStartUp	32
MSShutDown	33
MSVersion	34
MSStatus	35
 GetMSData	36
SetCallBack	37
MSSuspend	38
MSResume	39
 InitMIDIriver.....	40
RemoveMIDIriver.....	41
SetMIDIPort	42
SetMIDIMode	43
SetBasicChan	44
SetVelComp	45
SysExOut	46
 SetInstrument	47
PlayNote	48
StopNote	49
MIDIMessage	50
KillAllNotes	51
SetTuningTable	52
GetTuningTable	53
 SetTempo	54
SetBeat	55
SetRecTrack	56
SetPlayTrack	57
SeqPlayer	58
SetMetro	60
TrackToChan	61
SetTrackOut	62
Merge	63
Locate	64
LocateEnd	65
DeleteTrack	66
ConvertToTime	67
ConvertToMeasure	68

Introduction

MIDI Synth is a second generation note synthesizer Tool for the Apple IIGS. By integrating a completely new sequencer, MIDI interface and synthesizer into one program environment, MIDI Synth offers developers a powerful but simple solution to many of their sound needs. Because of this integration, most of the work required by an application to produce music is handled by this Tool. Whether you're writing a music education application to teach elements of music, or if you're writing an arcade game which needs both music and sound effects in the background, you'll find MIDI Synth to be an important tool.

Some important features of MIDI Synth are:

- Integrated synthesizer, sequencer and MIDI interface in one Tool.
- Simple programmer's interface - you don't need to understand the complex IIGS sound hardware.
- It's fast - uses only 25-30% CPU overhead under typical conditions.
- Does not complicate your program structure since it runs completely in the background.

Synthesizer

- Produce complex and interesting sounds with a 4 oscillator per voice architecture.
- Two 8-stage volume envelopes per voice with velocity control and variable note position decay.
- Pitch bend up to ± 1 octave.
- "Multi-sample" up to 8 waveforms per instrument across the keyboard range.
- Flexible control of oscillators - 6 oscillator configurations to choose from.
- Multi-timbral - simultaneously play any combination of instruments up to 7 voices.
- Sophisticated voice "stealing" algorithm extends the 7 voice limit to sound like more.
- Automatically handles MIDI Volume and Sustain Switch messages.
- Allows for alternate scale tuning arrangements.
- Supports Omni, Poly and Multi MIDI modes.

Sequencer

- 16 track sequencer with 96 ticks/quarter note resolution at tempos from 10 to 265 BPM.
- Sync to external MIDI devices
- Can wait for a MIDI key input to start sequence.
- Can count off a measure before starting sequence.
- Audible metronome is built in.
- Play/Record buffer limited in size only by available memory.
- Buffer support routines - fast merging, locating, and time (ticks) to measure conversions.

Support

- "SynthLAB" - an Apple application for designing your own instruments

- Apple Instrument Library available for use in your applications.
- Example source code available.

Limitations

In order to gain speed and to simplify the programmer's interface, most of the MIDI, sequencer and synthesizer functions have been "packaged" to do specific things. The price paid for this is comes in the form of reduced flexibility and "system" considerations. This Tool completely takes over the GS sound hardware. Sound interrupts, DOC ram and registers, and serial ports are all exclusively controlled by the MIDI Synth and are unavailable for application or system use. Also MIDI Synth does not support most of the other sound Tools. For example, an application which uses this tool can not use the AppleTalk port, support "system beep" inits or use the Sequencer (Tool 26) or Note Synthesizer (Tool 25) while this tool is active (Although one generator is available for use with The Sound Tool Set).

Previous sound Tool data formats are not compatible with MIDI Synth. Sequencer Tool sequences, MIDI Tool Set sequences and Note Synthesizer instruments cannot be used with the MIDI Synth.

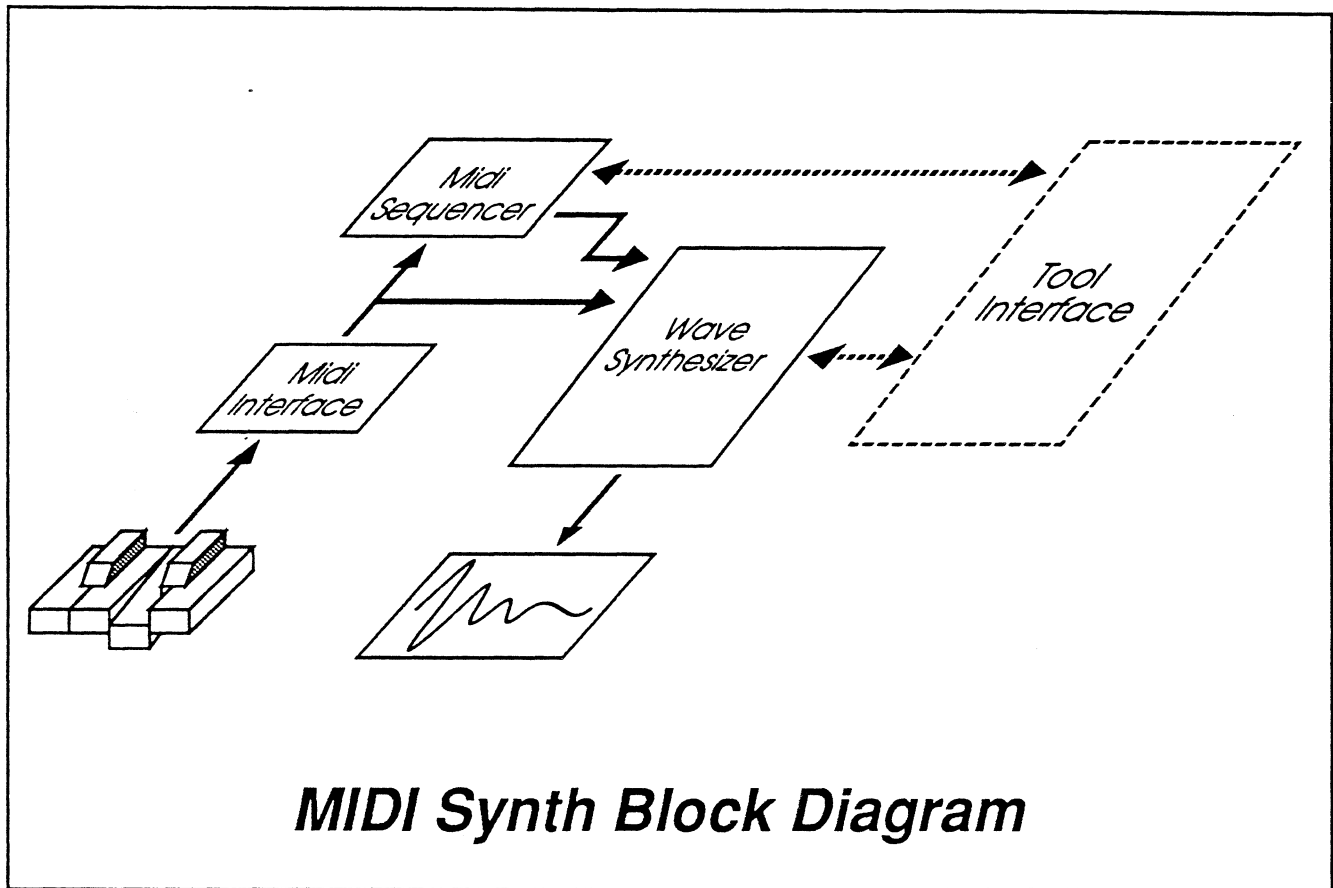
Overview

Referring to the "MIDI Synth Block Diagram", the three components which make up this tool are 1) the MIDI Interface, 2) MIDI Sequencer and 3) the Wave Synthesizer. How these three parts interact with each other is controlled by your application through the Tool Interface.

The Wave Synthesizer produces the output sound by playing a specified **Instrument Record**. An Instrument Record tells the Synthesizer how to build a particular sound, how it should behave over time and how it should respond to certain MIDI events. Commands to play an instrument can come from either the MIDI Interface, the Sequencer, directly from your application, or from all three simultaneously.

The Sequencer can either *PLAY* a pre-recorded MIDI sequence or it can *RECORD* a MIDI sequence for you. When it records a sequence, it reads MIDI data from the MIDI port, time-stamps this data to remember when it happened, and then sends this final time-stamped MIDI event (called a **Seq Item**) to a user-specified buffer somewhere in system memory (which is called a **sequence**). Later, you can then tell the Sequencer to *PLAY* back this same sequence. The Sequencer will now send all the Seq Items in the sequence out to either the MIDI port, the Synthesizer or both, in exactly the same order and relative time as it was when recorded. In other words, all notes and rhythms are accurately preserved. You can also simultaneously play one buffer while recording MIDI events into a second buffer, that is, *PLAY* and *RECORD* at the same time.

The MIDI Interface monitors the MIDI port for input data and notifies the Sequencer and the Synthesizer when data is received. It can also send output data through the same port when the Sequencer is playing a sequence.



Starting MIDI Synth

Here is a general outline on starting MIDI Synth. More specific examples can be found in the section "Using MIDI Synth".

1. Make sure AppleTalk is OFF.
2. Start the Sound Tool Set (Tool 08) with a *SoundStartUp* call.
- 3 Start the Misc Tools Set (Tool 03).
4. Start MIDI Synth with an *MSSStartUp* call.
5. Load your Instrument waves into DOC ram with the Sound Tool Set call *WriteRamBlock* .
6. Define all your Instruments with repeated *SetInstrument* calls.
7. If you're using MIDI in/output, load a MIDI driver. Call *InitMIDI*Driver.
Enable MIDI I/O with a *SetMIDI*Port call.

When your application finishes, MIDI Synth should be shut-down (*MSShutDown*) before closing the Sound Tool Set.

Interrupts

Since MIDI Synth runs almost exclusively in the background, special consideration must be given if the application disables interrupts. Disabling interrupts will halt most of the tool's operations and MIDI data will be lost to over-run errors. Since the GS interrupt handler is not re-entrant, routines at interrupt time (VBL tasks, MIDI Synth call-backs) should be short and fast. The recommended technique is to only set a flag or increment a counter at interrupt time, servicing any action required while outside the interrupt in the foreground.

If at intermittent times when you're doing heavy graphic drawing or intense calculations you may want to temporarily stop MIDI Synth (even when it's not playing a sound, update interrupts are still being serviced while MIDI Synth is active, this takes about 150us of time every 5ms). Issuing an *MSuspend* call will turn the 5ms update interrupts off. If any notes are playing at this time they will freeze at whatever level they were at when the call was made. If MIDI input is enabled, MIDI input data will still be buffered (keep in mind that the internal MIDI buffer is only 128 bytes long, so it can easily overflow causing you to lose notes). To turn MIDI Synth back on and continue from where you left off, make an *MSResume* call. If there are any buffered MIDI notes at this time, then they will all sound at once.

MIDI Synth Oscillators and Voices

The Apple IIGS hardware supports 32 digital oscillators. MIDI Synth uses one of these (Oscillator #31) as a time-base for background updating every 5ms. Another oscillator (Oscillator #30) is used by MIDI Synth to play the built-in metronome. Two oscillators (Oscillators #28 and #29 - Generator #7) are reserved for your application to play sampled sounds from system memory with the Sound Tool Set (see next section). This leaves MIDI Synth with 28 oscillators (#0 thru #27) to create its Synthesizer voices. Since each MIDI Synth voice uses 4 oscillators (see "Voice Architecture" section), a maximum 7 voices or instruments can be active at any given time.

These 7 voices are *dynamically* assigned to you by MIDI Synth. This means that when a request is given to play a note (either thru MIDI, the Sequencer or by your application), MIDI Synth will decide which of the 28 oscillators it will use to play the requested Instrument. If all 7 voices are currently being used, MIDI Synth will "steal" (turn the least noticeable note off) a voice, in order to free four oscillators for the new note. In most cases, your application should not concern itself with voices or oscillators, since MIDI Synth manages these automatically for you.

Using MIDI Synth with the Sound Tool Set

There may be some sounds needed by your application which are either impossible to synthesize, or too large to fit into DOC ram (>64K) as Instrument waveforms. These are special cases where you may want to play back sampled digital recordings stored somewhere in system ram. MIDI Synth itself cannot play these, but special "hooks" have been provided for your application so that you can use certain Sound Tool Set (Tool #08) calls to play these sampled recordings when needed.

Since MIDI Synth takes full control of the GS sound hardware and sound interrupts, there are some restrictions to keep in mind when you use the Sound Tool Set with MIDI Synth. The only available Sound Tool Set calls you can use are:

1. *SoundStartUp* and *SoundShutDown* at the beginning and end of your application.
2. *ReadRamBlock* and *WriteRamBlock* - only when there are no active MIDI Synth voices.
3. *SetSoundVolume* - set only the system volume or Generator #7.
4. *FFStartSound* and *FFStopSound* using only Generator #7.
5. *FFSoundDoneStatus* and *FFGeneratorStatus* for Generator #7 only.
6. You can use the *low level routines* at your risk only when there are no active MIDI Synth voices.

You will have to use *WriteRamBlock* to enter your Instrument waveforms into DOC ram. Make sure that interrupts are disabled when you do this. Be aware that *FFStartSound* writes to 2 buffers in DOC ram when playing a sample (see the "Apple IIGS ToolBox Ref." manual), so it may wipe out Instrument waveforms used by MIDI Synth. Both the Sound Tool Set and MIDI Synth will be competing for interrupt time when you use *FFStartSound*, so output quality and reliability will probably degrade when you use them simultaneously. It is recommended that you turn MIDI Synth off temporarily with a *MSSuspend* call, play your sampled sound (*FFStartSound*), when the sample is done then re-enable MIDI Synth with a *MSResume* call.

MIDI

With MIDI Synth, most of the interface to MIDI has been done for you. *Channel Voice* messages and *System Real Time* messages are both processed automatically by the tool. Since this processing is done at interrupt time in the background, no servicing is required by your application when the Sequencer is playing or recording, or when MIDI data is received at the MIDI port. This frees the application from all the tedious work required to service the Synthesizer, Sequencer and MIDI interface together.

The chart "Recognized MIDI Messages" shows the MIDI messages supported by MIDI Synth. They can be sent either by external MIDI devices into the SCC MIDI port, played by a sequence or sent directly by your application using the *MIDIMessage* call. Those *Channel Voice* messages not shown on the chart (for example a Modulation Wheel) will have no effect on MIDI Synth. They can, however, be recorded and played back through the Sequencer to an external MIDI device connected to the MIDI port.

Channel Mode messages (Controllers 122-127) function as "All Notes Off" messages only. An All Notes Off message does not immediately shut the voices off. Instead, all voices are forced into the Release 1 segment of the envelope.

System Common messages (\$F1-\$F6), *Active Sensing* (\$FE) and *System Reset* (\$FF) are not supported and are filtered out by MIDI Synth.

System Real-Time messages: *Start* (\$FA), *Stop* (\$FC) and *Continue* (\$FB) are supported through call-backs (see "Call-Back Routines"). *Stop* will halt the Sequencer if it is active.

Status	Data #1	Data #2	Description
\$8x	Note # 0-127	Velocity 0-127	Note Off Velocity is ignored
\$9x	Note # 0-127	Velocity 0-127	Note On Note Off when Velocity =0
\$Bx	Control # 7 64 123-127	Value 0-127 0=Off 127=On 00	MIDI Volume Sustain Switch All Notes Off
\$Cx	Instrument # 0-15	-----	Program Change
\$Ex	LSB 00	MSB 0-127	Pitch Bend Center = 64
\$F8	-----	-----	Sequence Timing Clock
\$FA	-----	-----	Start Sequence
\$FB	-----	-----	Continue Sequence
\$FC	-----	-----	Stop Sequence

Recognized MIDI Messages

When enabled, the *Sequence Timing Clock* (\$F8) is handled internally. MIDI Synth can only sync to an external Timing Clock, it will not generate them while playing. This means that MIDI Synth can only be a slave device and not a master.

System Exclusive messages (\$F0 xx xx ...) are supported only through Call-Backs (see "Call-Back Routines"). Once MIDI Synth receives a Sys Ex message, all further MIDI data is ignored internally until another MIDI Status (MSB = 1) or an EOX (\$F7) is received.

The *Program Change* message (\$Cx) is supported in MIDI Synth Call-Back (see "Call-Back Routines").

MIDI Data Flow

To get a conceptual idea on how MIDI Synth is structured, you can think of it as consisting of three separate modules which communicate with each other and to your application via MIDI messages. The three modules, as mentioned earlier, are the MIDI Interface, the Sequencer and the Synthesizer. Each module is basically independent of each other and can be used by your application without the others. In other words, your application can use the Synthesizer without the MIDI Interface and Sequencer, or you can use the Synthesizer with the Sequencer but without the MIDI Interface. The final combination is decided by your application and can be dynamically changed in the course of your program. By putting these three modules into one Tool, you get the important advantage of efficiency and streamlined operation. Each module intimately knows how the others operate and how to communicate with them.

The internal communication paths are altered by your application via data “filters”, which modify and control the MIDI data. The “MIDI Data Flow” diagram shows the communication paths, location of each filter (dotted triangles in the diagram), the Tool calls which affect the filters, and finally, this diagram shows where your routines get called during call-back.

MIDI INPUT PATH

Starting at the bottom left corner in the diagram, the source for MIDI data from external devices is set by the *SetMIDIPort* call. This call selects which SCC port MIDI Synth will “listen” to for MIDI data. If neither the Modem nor the Printer port is selected, then the MIDI port is disabled, and data cannot flow through this path. The first thing the MIDI Interface does when it receives an input byte is to intelligently accumulate the input data in order to build a MIDI Message packet. It examines the current protocol and collects the input until it has a complete MIDI packet.

After the full MIDI Message packet is received, MIDI Synth calls your *PacketIn* call-back routine. This is where your application can modify and filter the MIDI messages as they come in. If the ‘carry’ flag is set when your application returns to MIDI Synth, the Message will be discarded and will not be sent to the Synthesizer and Sequencer. If the ‘carry’ flag is clear when returning from your *PacketIn* call-back, then the Message continues to the next filter.

This next filter does several things. It can modify or ignore this MIDI message, depending on the current MIDI mode (Omni, Poly or Multi), and by the Basic Channel value. If the mode is Omni, then the Message channel number is forced to the Basic Channel number. If we’re in Poly mode, then MIDI Synth compares the Message channel number with the Basic Channel, and if they don’t match, the Message is discarded and ignored. The Message channel number is not modified if we’re in Multi mode. The last thing this filter does is to “pad” the velocity value (if the Message is “note on”) with the value set by the *SetVelComp* call. If the message makes it through this filter, the Synthesizer will play it.

If the Sequencer is currently recording MIDI data, a track number reference (set by the *SetRecTrack* call) is attached to this Message by one final filter before the Sequencer receives it. If no tracks are set for Record, then the Message gets discarded, and will not be recorded by the Sequencer.

SEQUENCER OUTPUT PATH

As shown in the diagram, the first main MIDI communication path is from the MIDI port, to the Synthesizer and finally to the Sequencer. The other major path is the reverse of this, sending MIDI Messages from the Sequencer, to the Synthesizer and finally out through the MIDI port.

After leaving the Sequencer, the Message's channel number may get translated to the value set by the *TrackToChan* call. This call enables you to force the channel numbers of all Messages (Seq Items) in a particular track to a specified value. Since the Message channel number is used as a reference to Instrument numbers in the Synthesizer, this forces all Messages in a track to play a specific Instrument.

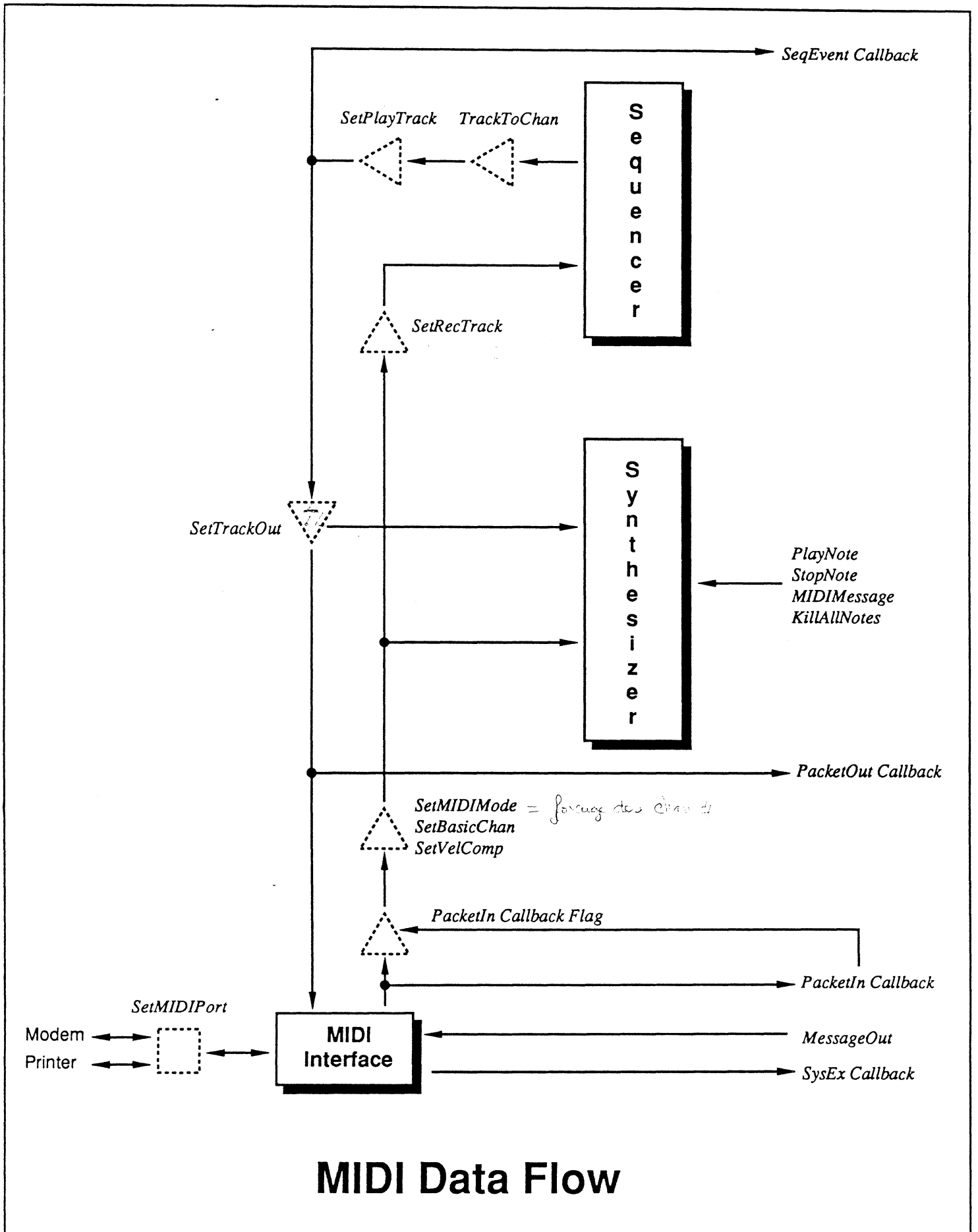
MIDI Synth then checks the track number in the Message to see if that track is active when playing (*SetPlayTrack* enables or disables the track). If it is set as active, then the Message continues down the path, otherwise it gets discarded here and goes no further.

At this point, MIDI Synth calls your *SeqEvent* call-back routine and then checks to see where to output this Message, based on its track number. With the *SetTrackOut* call you can specify a track to play only the Synthesizer or to only output the track thru the MIDI port, or both. Your application can monitor those Messages which will be sent out the MIDI port with the *PacketOut* call-back routine.

OTHER MIDI PATHS

There are two other MIDI Message paths in MIDI Synth. The first is between your application and the Synthesizer with the *PlayNote*, *StopNote*, *MIDIMessage* and *KillAllNotes* calls. The second path enables you to send MIDI Messages directly out the MIDI port via the *MessageOut* call.

More detailed information on call-backs can be found in the "Call-back Routines" section.



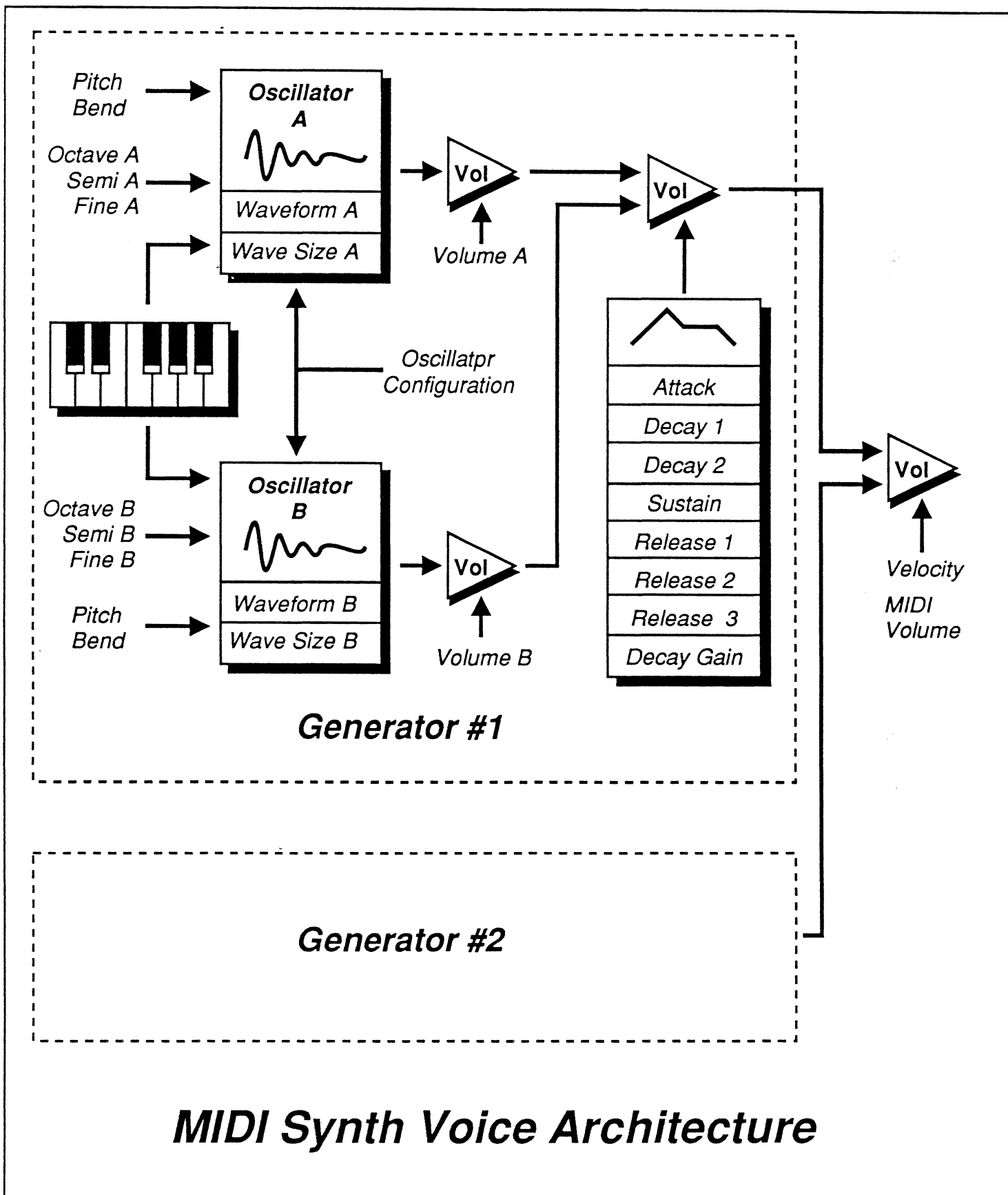
Voice Architecture

In the early days of music synthesis, synthesizers were made by connecting a group of analog sound modules together with wires and adjusting the various parameters with control knobs to create an instrument. A new instrument was created by reconfiguring the sound modules and setting the control knobs to new values. Modern digital synthesizers are similar except the sound modules are now subroutines, the knobs are parameters in memory and the wires are control instructions changing the flow of the program. Using this analogy, an Instrument Record contains instructions on which one of the “sound modules” to use, where to connect the “wires” and how to adjust the various “knobs”. The basic synthesizer structure and its elements is called the “voice architecture”. This defines what parameters are available to the programmer, where the sound sources come from and the types of control devices that are built in the synthesizer.

The MIDI Synth voice architecture is designed around 2 identical *generators*. Each generator has 2 oscillators and level controls for each. The oscillators are mixed into a variable amplifier whose gain is controlled by an 8 stage envelope. The envelope is a series of linear ramps that give the sound a volume contour that changes over time. Finally, both generators are summed into another variable amplifier that is modified by the note velocity parameter and by any ongoing MIDI Volume message for that instrument.

The oscillators are most unique part of this architecture. Each oscillator can play any waveform in DOC memory of various lengths. The *Oscillator Configuration* parameter controls how the 2 oscillators of each generator behave and how they interact. Some configurations play the oscillators continually while others force the oscillators to play the waveform only once and stop. Other configurations start an oscillator only when another oscillator has finished its waveform.

The Synthesizer has 7 dynamically assigned output voices. This means at any one time, no more than 7 notes can be active and playing simultaneously. The 7 voices can be any combination or number of instruments assigned to the Synthesizer. If the Synthesizer is asked to play more than 7 voices, then a voice will be “stolen” from an active note to make room for the requested voice. When stealing a note, the synthesizer tries to pick the note that will be least noticed when turned off.



Instrument Records

As mentioned above, the Instrument Record sets the parameters that define a particular instrument. It tunes the oscillators, points to waves, sets the oscillator mode, controls the amplifiers and defines the envelope. The Instrument Record itself is made up of 2 groups of records. They are Envelope Records and Wavelist Records. Since there are 2 generators for each instrument, these records are grouped according to which generator they control.

Generator 1

Generator 2

Offset

0	Envelope Record
16	Wavelist Record
32	Wavelist Record
48	Wavelist Record
64	Wavelist Record
80	Wavelist Record
96	Wavelist Record
112	Wavelist Record
128	Wavelist Record
144	Envelope Record
160	Wavelist Record
176	Wavelist Record
192	Wavelist Record
208	Wavelist Record
224	Wavelist Record
240	Wavelist Record
256	Wavelist Record
272	Wavelist Record

The **Envelope Record** contains rate and level values for the ramps that make up the envelope and a few other parameters that affect all wavelists. There are two Envelope Records per Instrument, one for each generator.

The **Wavelist Record** controls the oscillators. How the oscillators are tuned, what wave they play and what configuration they're in, are all controlled by this record. There are 8 of them for each generator, each of which can be active only when the note that they must play falls into their note zone or range. Each WaveList Record has a note range parameter called Top Key that sets this zone (see "WaveList Record" for example and details on Top Key).

Instrument Record

Envelope Record

<i>Attack Level</i>	0-127	<i>Attack Rate</i>	0-31
<i>Decay 1 Level</i>	0-127	<i>Decay 1 Rate</i>	0-31
<i>Decay 2 Level</i>	0-127	<i>Decay 2 Rate</i>	0-31
<i>Sustain Level</i>	0-127	<i>Decay 3 Rate</i>	0-31
<i>Release 1 Level</i>	0-127	<i>Release 1 Rate</i>	0-31
<i>Release 2 Level</i>	0-127	<i>Release 2 Rate</i>	0-31
<i>Release 3 Level</i>	0-127	<i>Release 3 Rate</i>	0-31

The envelope parameters control how the volume changes for the instrument while the note is playing. All natural sounds have some sort of envelope. They may start softly and gradually become louder or they might start loud and slowly fade out. With the envelope parameters we imitate this by piecing together a series of volume segments to the shape that matches the effect we want. Each segment has a target volume level and a rate value which sets how fast the volume changes until it reaches the target level. In other words, the volume starts at the previous target level, increases or decreases at a programmed rate until it gets to the new target level. Then it goes on to the next segment. With 8 segments in the envelope, very complex contours can be defined.

When a note is started, the volume starts at a zero level (silence) and increases to the Attack Level with a slope set by the Attack Rate. From there, it goes to the Decay 1 Level at the Decay 1 Rate. Next it moves to the Decay 2 Level with the Decay 2 Rate. Using the Decay 3 Rate, the envelope finally reaches the Sustain Level. At this point the envelope stops changing and waits for the note to be released (MIDI note off message). It sits at this level until it gets a command to turn the note off. The envelope then steps through the Release 1, 2 and 3 segments until it is finally off and stops. Notice that Release 3 segment always decreases to a zero level (that's why there is no "Release 3 Level" setting; it's always zero).

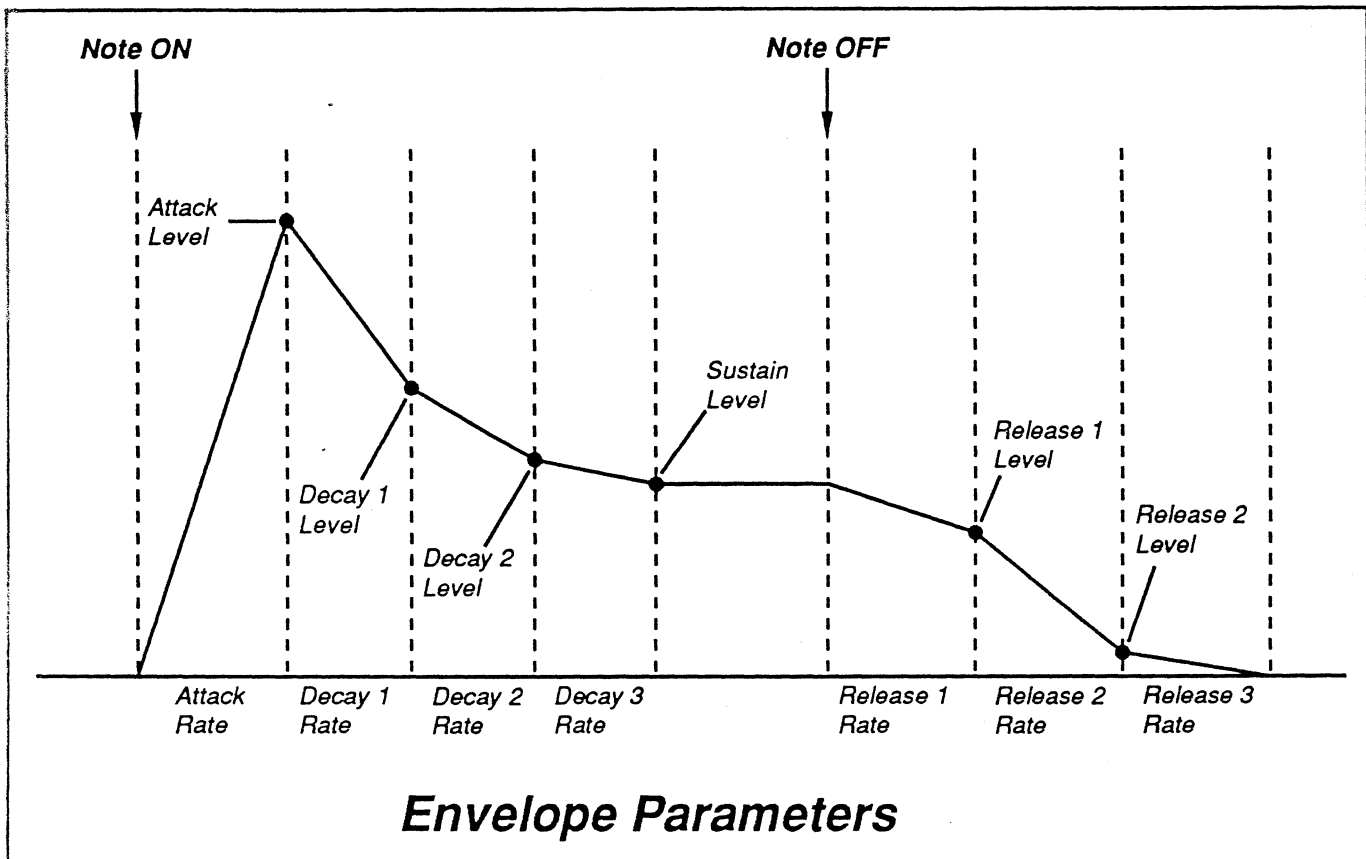
The segments can either increase or decrease to the new target depending on whether the new target level is above or below the previous target level. For example, if Decay 1 Level is higher than the Attack Level, the Decay 1 segment will increase from the Attack Level to the Decay 1 Level. If Decay 1 is lower, then the segment will decrease the volume until it reaches the Decay 1 Level.

Whenever a note off command is sent to the synthesizer (MIDI Note Off message or a StopNote Tool call), and the envelope has not reached the Sustain stage, the envelope will always be forced to the Release 1 segment., unless the envelope already is in one of the Release segments. For instance, if the envelope is somewhere in the Attack segment and a note off command is sent, the envelope will immediately start moving from its present point to the Release 1 Level at the Release 1 Rate.

The envelope will stop and finish whenever it reaches a zero level. If the Sustain Level is set to zero, the end of Decay 3 segment will be considered the end of the envelope. Once it reaches this point, the envelope will remain at zero ignoring note off commands. The same is true for any of the envelope segments. However, before reaching this point, a note off will still force the envelope to the Release 1 segment.

<i>Offset</i>		<i>Value</i>
0	<i>Attack Level</i>	0-127
1	<i>Attack Rate</i>	0-31
2	<i>Decay 1 Level</i>	0-127
3	<i>Decay 1 Rate</i>	0-31
4	<i>Decay 2 Level</i>	0-127
5	<i>Decay 2 Rate</i>	0-31
6	<i>Sustain Level</i>	0-127
7	<i>Decay 3 Rate</i>	0-31
8	<i>Release 1 Level</i>	0-127
9	<i>Release 1 Rate</i>	0-31
10	<i>Release 2 Level</i>	0-127
11	<i>Release 2 Rate</i>	0-31
12	<i>Release 3 Rate</i>	0-31
13	<i>Decay Gain</i>	0-9
14	<i>Velocity Gain</i>	0-10
15	<i>Pitch Bend Range</i>	0-12

Envelope Record



Once both envelopes have finished (both at zero levels), the oscillators will be turned off and the note will be considered as being inactive. Sending Note Off, pitch Bend or MIDI Volume commands at this point will have no effect.

One point to keep in mind, is that the 31 slopes that can be set for each segment are RATE values and not TIME values. This means that if a certain segment had a rate value of 3, and a level of 127, it will take 6.92 sec to complete the segment if it started from a zero level (for example Attack). If the segment started from a value of 64, then the segment will take half as much time to complete since it has only half the distance to cover (from 64 to 127). Likewise, if the level is halved, then the time to complete the segment is also halved to 3.46 sec. The Envelope Rate chart shows times for each rate when the segment must go full range from 0 to 127.

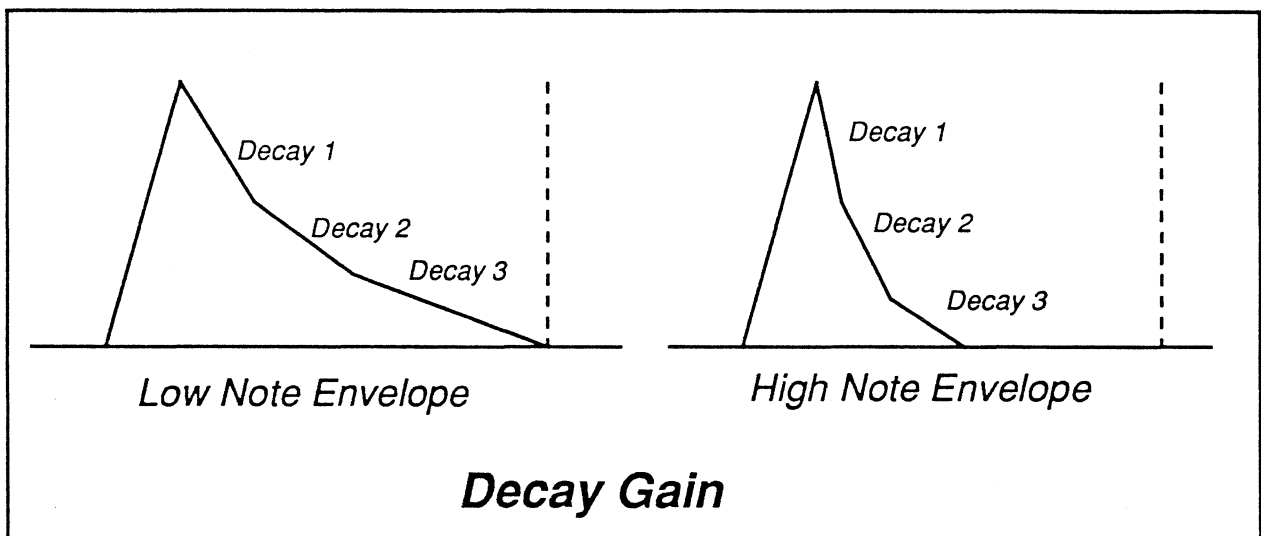
Rate Value	Time*	Rate Value	Time*
31	0.00	15	.54
30	.01	14	.64
29	.02	13	.82
28	.03	12	1.02
27	.04	11	1.26
26	.05	10	1.56
25	.06	9	1.93
24	.08	8	2.39
23	.09	7	2.96
22	.12	6	3.66
21	.16	5	4.52
20	.18	4	5.59
19	.23	3	6.92
18	.28	2	8.55
17	.32	1	10.53
16	.43	0	13.08

* Time is in seconds to ramp full scale (0-127)

Envelope Rates

Decay Gain 0-9

In Nature, percussive sounds have a unique feature. High frequencies die out much quicker than low frequencies. When playing a piano you will notice that striking and holding down a low note causes the strings to sound and resonate for quite a long time. If you try the same on high notes, the strings lose their energy quickly and die out. The Decay Gain parameter imitates this important feature by automatically increasing the entered Decay Rate values for notes as they go up the scale. With Decay Gain set to zero, the Decay Rate for all notes will be the same as the entered value, both low and high notes will decay at the set rates. Increasing the Decay Gain value will cause higher notes to decay faster than the entered amount. This parameter affects higher notes more than lower notes.



Velocity Gain 0-10

This controls the instrument's sensitivity to MIDI velocity data. The larger the value, the more sensitive velocity becomes. A value of zero causes no effect on velocity data. This parameter can be used to match the dynamic characteristics of instruments you wish to create. For example a piano, which has a very large dynamic range may have a Velocity Gain of 1, while a pipe organ, which has no dynamic range, may have Gain of 10. Velocity Gain can also be used to compensate for performance variations on MIDI keyboards. Some players pound the keys when they play while others have a gentler style. Also different manufactures have different velocity ranges on their keyboards.

Pitch Bend Range 0-12

This parameter sets how far MIDI pitch bend data can bend a note up or down. The values are in semitone increments. A zero value disables the pitch bend feature. *It is recommended that this parameter be set to zero unless you really plan to use it.* Disabling Pitch Bend will decrease overall MIDI Synth CPU overhead.

Wavelist Record

Top Key 0-127

If you built an Instrument based on a sampled middle C female voice and played it back with a middle C note it would sound the same as the original. Play a few notes around middle C and it still will sound much like the female voice you sampled. Although the pitch is different, the same female voice characteristics are still there. Now, if you played this same sample an octave below middle C, you will be shocked to hear this pleasant female suddenly sound like Darth Vader. Play the sample an octave above middle C and the voice now sounds like Minnie Mouse. There is only a very small useful range of notes where the sample sounds like a female voice. This is because in Nature, the physics of the instrument produces a different waveform for every new pitch. As in the voice, the mouth acts as a variable filter. For every single pitch, the filter has a different effect on the harmonics produced by the voice. From this, it would seem that in order to reproduce any natural instrument, a separate sample for each note is required. Fortunately, you can usually get away with one sample for a range of notes. The size of a range varies and depends on the particular instrument.

The technique of having many samples of the same instrument across the pitch range is called *multi-sampling*. This is the reason why there are 8 wavelists for each generator. You can assign each wavelist to be active only in a certain pitch range. As in the voice example, each wavelist could be set to use a different voice sample and each one of the 8 Wavelists could be programmed to respond to a different pitch range.

This zone is set by the Top Key parameter. Its values are MIDI Pitch numbers from 0 to 127 with 60 being middle C. Before an instrument starts playing, the wavelists are scanned to find which one of the eight has a Top Key greater or equal to the pitch we wish to play. The first one found with this condition is the one that gets used. The wavelists get scanned in order from 1 to 8. Only Top Keys in Generator 1 are used in the scan. When a match is found, the same numbered wavelist for Generator 2 is used. In other words, if Wavelist 4 from Generator 1 is used, then Wavelist 4 from Generator 2 is also used. If all the Top Keys are less than the desired pitch, then nothing will be played.

Suppose Top Keys from each Wavelist for Gen 1 were set as follows:

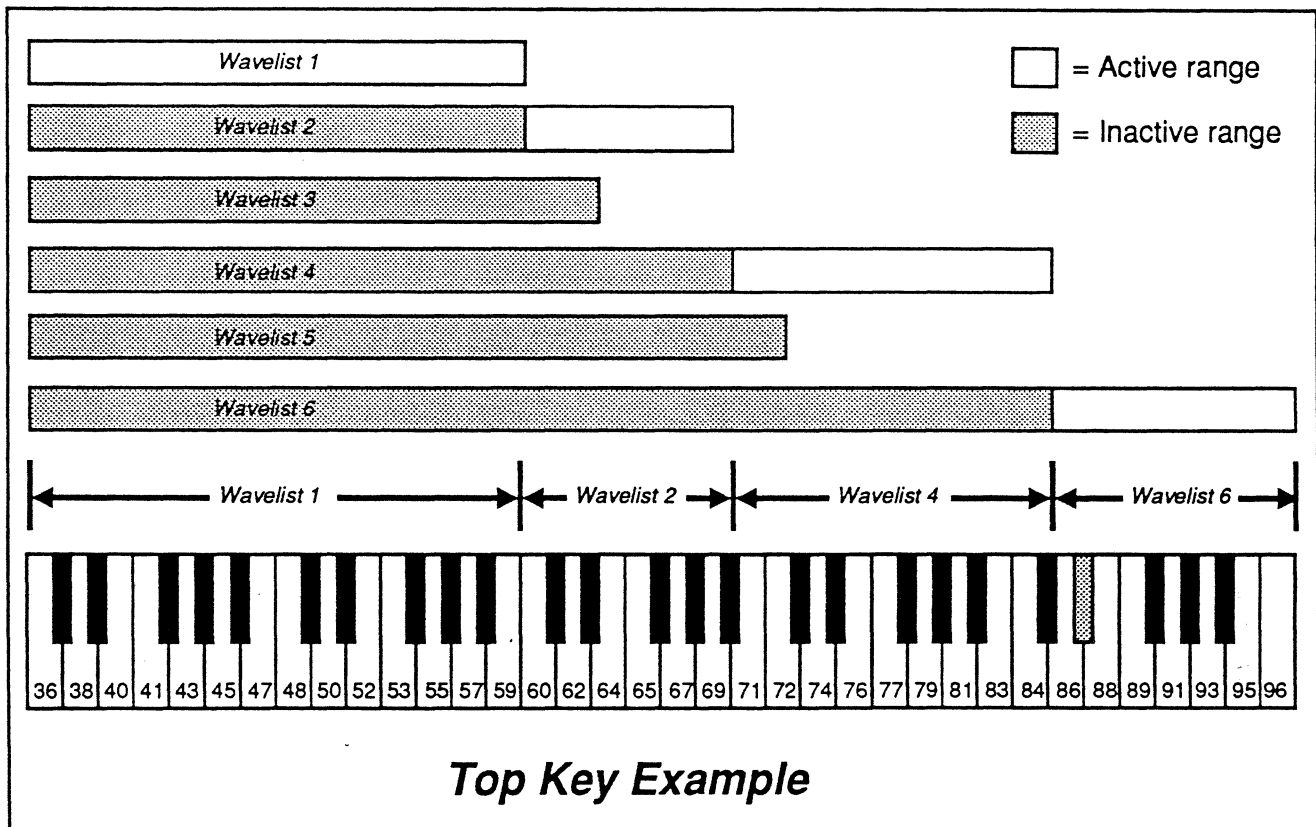
Wavelist 1 Top Key.... 59
Wavelist 2 Top Key.... 70
Wavelist 3 Top Key.... 62
Wavelist 4 Top Key.... 84

Offset		Value
0	Top Key	0-127
1	Osc Config	0-5
2	Stereo	0-7
3	Detune	0-63
4	Wave Address A	0-\$FF
5	Wave Size A	0-7
6	Volume A	0-127
7	Octave A	0-6
8	Semi-tone A	0-11
9	Fine Tune A	0-63
10	Wave Address B	0-\$FF
11	Wave Size B	0-7
12	Volume B	0-127
13	Octave B	0-6
14	Semi-tone B	0-11
15	Fine Tune B	0-63

Wavelist Record

Wavelist 5 Top Key.... 72
 Wavelist 6 Top Key.... 127

It doesn't matter what Wavelist 7 and 8 have for Top Key because the first 6 cover the entire MIDI range, thus they will never be scanned because one of the lower Wavelists will always claim the note. Now suppose the D# two octaves above middle C is to be played by an instrument, MIDI note 87. Wavelists 1 and 2 won't be used because their TopKeys are less than 87. Wavelist 3 will never be used for any note because with a value of 62, either Wavelist 1 or 2 will always claim a note in its range. Wavelist 4 is under and Wavelist 5 is always locked out like Wavelist 3 because Wavelists 1,2 and 4 cover its range. Wavelist 6 is used since its range covers notes 84-127.



Osc Config 0-5

The Osc Config parameter controls how oscillators A and B for each of the two generators are organized and how they interact with one another. In order to understand why these different configurations exist and how to effectively use them, a short discussion on how MIDI Synth creates sounds is necessary.

Synthesizer Basics

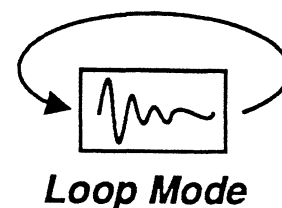
MIDI Synth supports two basic methods of sound generation: additive synthesis and sampling. With additive synthesis, instruments are built from scratch by combining basic simple elements of the sound together to form a more complex sound at the end. If you were building an orchestral string sound, you could take a single cycle wave that approximates the timbre of a violin and assign an oscillator to continually play that wave. To get a full orchestral sound, you might add in other oscillators that are playing viola and cello timbres and slightly detune them to create a complex phase-shifting effect. Then finally you would add a volume envelope to these oscillators so that their attacks and decays match those of a real orchestra. The advantage of additive synthesis is that it is very flexible since you have control over the many elements that make the sound. You can change the envelopes, change waves for different timbres or tune the oscillators, giving subtle or drastic changes in the sound. Additive synthesis is also very memory-efficient. Storing only single-cycle waves takes very little memory. Since you control the timbres and since the waves are always full scale, additive synthesis sound quality is usually very clean. The disadvantage of additive synthesis is that it takes a lot of CPU overhead to manage many oscillators and envelopes in real-time.

With sampling, instead of building a sound from scratch, you simply digitize the final sound you want and have the computer play it back as one complete sample. As in the orchestral strings example, you would feed the output of a CD player or tape deck into your GS and digitally record a section that has orchestral strings in it. When you play it back, all the subtle complexities, richness and characteristics of the sound are all reproduced. The obvious advantage with sampling is that you can very easily produce complex sounds without having to painstakingly build the sound from scratch. You can record sounds that would be almost impossible to synthesize. Sampling requires very little CPU overhead since all you manage is a single oscillator and the envelope is already part of the sample. But there are many problems with sampling. Since the samples are digital images of the sound, even short samples can take huge amounts of memory. The longer the sample, the more memory it takes. Sampling has no flexibility, if you wanted an orchestral string sound with a harder attack or a longer release, you would have to record another huge sample and load it into free memory (if you could in fact find such a recording). Because of sampling and playback rate limitations, and the fact that the envelope is part of the digital information (lower S/N ratio), sampling is usually very noisy and has a perceptible “grainy” quality to it.

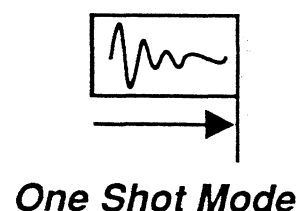
Since MIDI Synth supports both methods, you can overcome many of the problems of each by combining the advantages of the two methods together to form a third hybrid synthesis method. For example, to make the orchestral strings, you would sample the attack portion of the strings, capturing all the complexities of bows striking the strings, etc., and assign one oscillator to play back this sample. Then, the remaining three oscillators can be combined to play the sustain and release portion of the sound using additive synthesis. So now you have built an instrument that has captured the complexities of the sound (the attack would be very hard to synthesize) and at the same time has an clean sustain that can be played infinitely without taking up large amounts of memory. If you needed a longer release, you simply change the envelope release rate. If you wanted to create something never heard before you could use the string attack, but use a piano sustain.

Each one of the MIDI Synth oscillators operates in either one of two basic modes: Loop mode or One-shot mode. In **Loop** mode, the oscillator scans the wave from the beginning to the end, then jumps back to the beginning and starts scanning the wave again. It does this continually until the oscillator is turned off. If the wave is only a single cycle, then you hear a static pitch with with a timbre that is

characteristic of the given wave. For example, when playing a square wave, you would hear a constant square wave “buzz”. On the other hand, if the wave is a recorded sample of a sound with many dynamically varying waves to it, then you will hear the same sound constantly repeat itself over and over again. If, for example, you took a sample of the word “hello”, then playing it in Loop mode would result into repeating pattern of “hello hello hello.....”. Low notes result into a slow repeating pattern while high notes repeat quickly since they get scanned faster and therefore finish sooner.



In **One-shot** mode, the oscillator scans the wave only once and stops. It stays silent for the remaining part of the note. The “hello” sample would sound only once and the square wave would produce only a “click” sound. This mode is primarily used to play back short recorded samples.



The Oscillator Configuration parameter organizes the oscillators with the two modes in various ways.

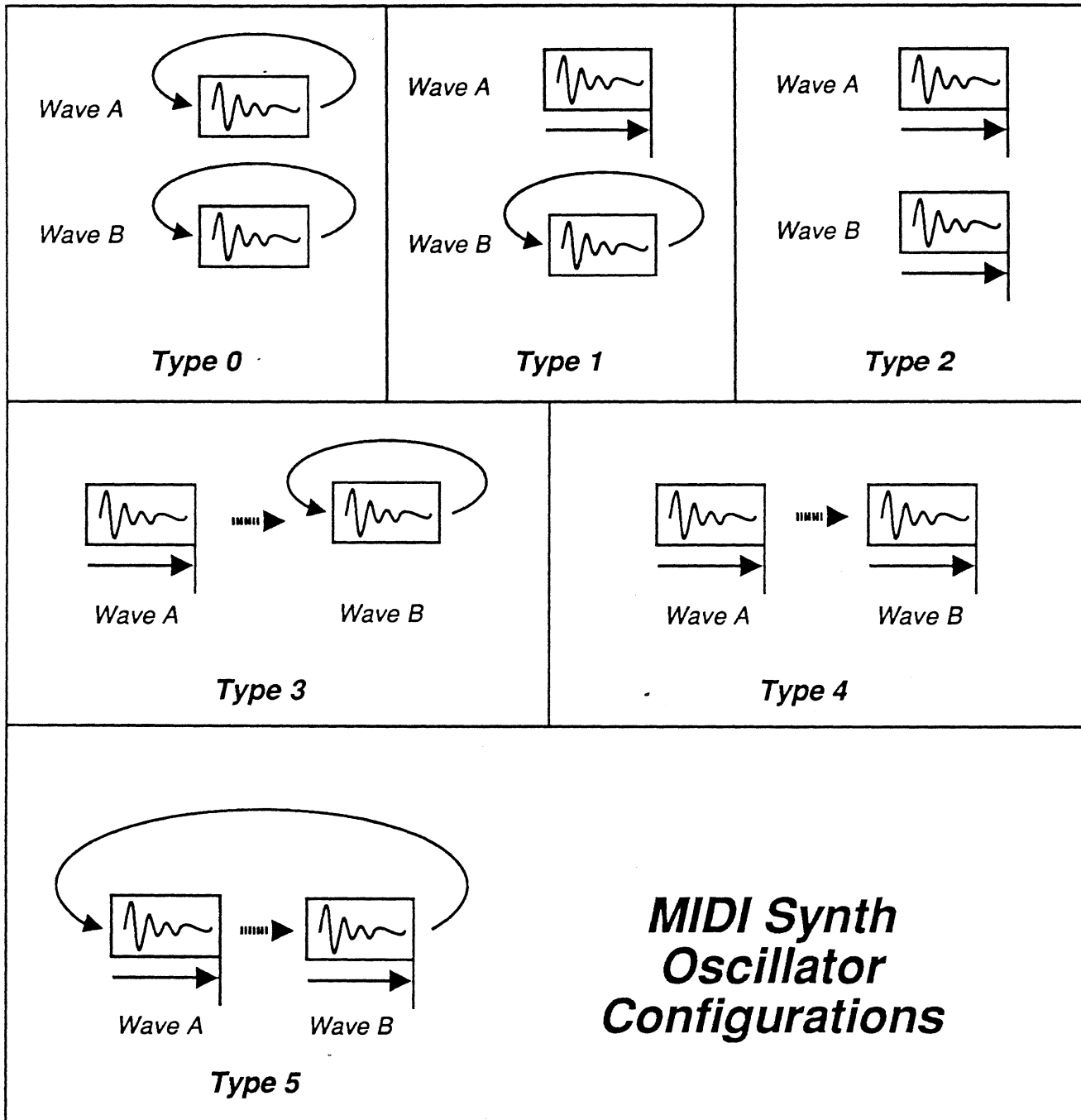
- Type 0 Both Osc A and Osc B play in Loop mode together.*
- Type 1Osc A in One-shot mode with Osc B in Loop mode.*
- Type 2 Both Osc A and Osc B play in One-shot mode together.*
- Type 3 Osc A in One-shot followed by Osc B in Loop mode.*
- Type 4 Osc A in One-shot followed by Osc B in One-shot mode.*
- Type 5 Osc A in One-shot followed by Osc B in One-shot mode, followed by Osc A in One-shot and so on.*

Stereo 0-7

This sets the three bit “channel” output found inside the computer on the sound expansion connector. Normally the sound is unaffected by this parameter unless you are supporting hardware attached to the connector. Presently, several third-party cards use these bits for stereo positioning of the sound output (LSB specifies right or left channel).

Wave Address A	\$00-\$FF
Wave Size A	0-7
Wave Address B	\$00-\$FF
Wave Size B	0-7

Since all the waves reside inside DOC memory, the synthesizer needs to know the location and size of the wave in order to play it. The Wave size parameter can specify a wave between 256 bytes and 32k bytes in length. Although a full 64k is available for wave storage, only the high byte (page number)



is entered for Wave Address since the smallest wave is 1 page (256 bytes) in length. The wave address is highly dependent on the size of the wave. Certain sizes of waves can be stored only on certain boundaries in DOC ram. Waves are aligned on their size boundaries. A 256 byte wave can reside in any of the 256 pages available in DOC ram (\$00, \$01, \$02...), a 512 byte wave can reside only on an even page (\$00, \$02, \$04...), a 1k wave is stored on every 4th page boundary (\$00, \$04, \$08...), and so on.

If you're using the Metronome, you can't use pages \$00 and \$01 because the "wood block" wave used for metronome ticks is stored there by MIDI Synth.

Volume A 0-127
Volume B 0-127

This sets the output level for each oscillator in the generator. A value of 127 outputs the wave at full volume while a value of 0 turns the oscillator off.

Octave A 0-6
Semitone A 0-11
Fine Tune A 0-63
Octave B 0-6
Semitone B 0-11
Fine Tune B 0-63

Size Value	Byte Size	Wave Address Byte Range
0	256	xxxx xxxx
1	512	xxxx xxx0
2	1024	xxxx xx00
3	2048	xxxx x000
4	4096	xxxx 0000
5	8192	xxx0 0000
6	16,384	xx00 0000
7	32,768	x000 0000

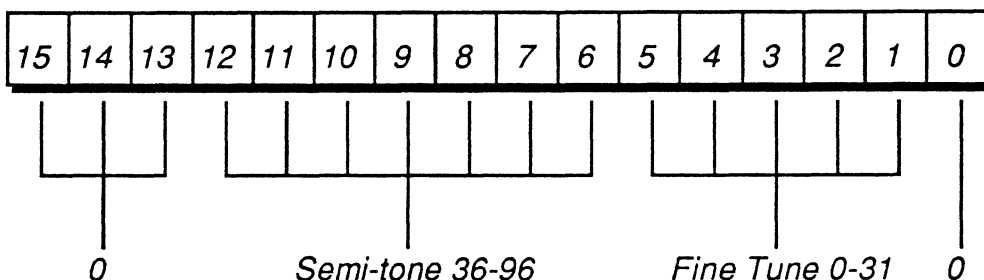
Wave Size and Address

These parameters adjust the base frequency for each oscillator. The Octave and Semitone parameters tune in increments of octaves and semitones respectively while the Fine Tune parameter tunes the oscillator in fractions of 1/64th of a semitone.

Note Tuning Table

The MIDI Synth Synthesizer plays notes tuned to the equal tempered scale. This tuning arrangement gives 12 balanced semitones per octave and each octave is double the frequency of the previous octave. This works well for keyboard instruments and most popular western culture music is tuned to this scale. However, MIDI Synth is not limited to this one scale tuning arrangement. An application can customize their own note tuning to any historical, experimental or non-western culture scale available.

The synthesizer uses an internal look-up table with an entry word for each of the 128 MIDI note values. By sending a new table with the *SetTuningTable* call, the pitch of each MIDI note that the synthesizer plays can be defined by the application. Each entry word in the table defines a semitone and fractional semitone value. *Note that the semi-tone must be within the 36 to 96 value range.*



Note Tuning Word

Seq Record

The MIDI Synth sequencer records and plays MIDI events. These events are grouped and stored by the application in what is called a **Seq Record**. To play a sequence, the application passes the sequencer a pointer to where in memory the Seq Record is located. The Sequencer then examines each MIDI event in the Seq Record and sends it to the Synthesizer or out to the MIDI port. Because the Sequencer has to know in what order and when to play a MIDI event, each event has a time-stamp associated with it (see "TIME-STAMPS" in next sub-section). This time-stamped MIDI event is called a **Seq Item**. The Seq Items are stored in increasing time-stamp order. When playing, the Sequencer increments its own internal clock. At every tick of the clock, the Sequencer looks at the next Seq Item's time-stamp. If the time-stamp matches the internal clock, then the Item is used and the Sequencer examines the next Seq Item. When an Item's time-stamp is at a value greater than the current internal clock, the Sequencer does nothing with the Item and waits until the clock tick increases to match the Item's value.

Offset		Value
0	Track Number	0-15
1	Time-Stamp High	0- $\$FF$
2	Time-Stamp Low	0- $\$FF$
3	Time-Stamp Mid	0- $\$FF$
4	Data Byte Count	0-2
5	MIDI Status & Channel	$\$8x-\Fx
6	Data Byte 1	0- $\$7F$
7	Data Byte 2	0- $\$7F$

Seq Item

The **Track Number** is a way of grouping Seq Items to some application defined reference. Sixteen tracks are available in the MIDI Synth Sequencer. Using the *SetPlayTrack* call, you can prevent MIDI messages with specified track values from being processed by the sequencer during playback. Likewise, when recording a sequence, the *SetRecTrack* call attaches this track reference value to all incoming MIDI Messages. The **Data Byte Count** specifies the number of bytes in the data byte field of the MIDI message. the **MIDI Status Byte** contains MIDI status message (in the high nibble), and the MIDI channel number (in the low nibble). When playing the Synthesizer, the MIDI channel field specifies which of the 16 Synthesizer Instruments gets the MIDI message. A Seq Item is always 8 bytes in length. If the MIDI message has less than 2 bytes, the unused Data Byte field(s) will have undefined values.

The end of a sequence must be marked with an $\$FFFF$ (**end-of-seq marker**) value following the last Seq Item. The buffer can be larger than the actual sequence since an end of sequence marker will always halt playback. This has no effect on the record buffer. If you are playing and recording at the same time, playback will halt at the end marker, while recording will continue until instructed by the *SeqPlayer* call to stop. The sequencer will insert an end marker at the record buffer end when told to stop recording.

To record and playback simultaneously, two buffers are needed: one buffer is used to record the incoming data, while the second is used to playback a previously recorded sequence. It's the application's responsibility to merge the two buffers in correct increasing time-stamp order if both buffers need to be played together at a later time (see info on the *Merge* call). Likewise, if a certain track is to be re-recorded, the application must remove all Seq Items from the play buffer with the desired track number (see info on the *DeleteTrack* call).

All running status messages at the MIDI port are constructed back to full MIDI messages. The sequencer does not convert Seq Items to running status messages when sending them out the MIDI port.

TIME-STAMPS

The term “time-stamp” is somewhat misleading in that it really does not directly represent time. The Sequencer is calibrated so that each tick of the clock represents a unit of 1/96th of a quarter note. For example, if two Seq Items have a difference of 96 time-stamp units between them, then they are separated from another by a value of a quarter note. So one time-stamp unit is a relative value representing a fraction of a beat. The Tempo value is what converts these time-stamp ticks into “real” time by setting the Sequencer clock to increment at a certain rate. If you double the Tempo, the the clock will now increment twice as fast, which will half the time it takes to play a quarter note. By referencing all your Seq Items to 96 ticks per quarter note (half note = 192, eighth note = 48, sixteenth note = 24, etc.), tempos and beats will all work correctly.

If you’re not using tempos and beats in your application, and you need to work only with time, then setting the Tempo (see *SetTempo* call) to a value of 125 BPM will give each clock tick a time value of approximately 5ms. This way if a Seq Item has a time-stamp of 100, for example, then it will play at about 1/2 second ($100 \times .005 = .5$) from the beginning of the sequence.

NON-MIDI MESSAGES

Besides storing MIDI Messages as Seq Items, the Sequencer can also interpret special **Sequencer Commands**. Since the Status field, when used for MIDI messages always has the high bit set (\$80-\$FF), this field is extended downward (\$00-\$7F) to store these special Commands. The Data 1 and Data 2 fields contain arguments for the specific commands.

SeqMarker \$00

A call to your *SMarker* call-back routine will be made whenever the sequencer encounters this command. You can put whatever you need as a reference in the Data Byte fields. This can be used to synchronize your application to a sequence.

SetSeqBeat \$02

This command will force the number of clock ticks per beat to the value specified in the Data 1 (low) and Data 2 (high) fields. Does the same thing as a *SetBeat* Tool call. This call is used when the meter must dynamically change within the sequence.

SetRelTempo \$04

A signed word displacement value stored in the Data 1 (low) and Data 2 (high) fields will be added to the current Tempo value. This call is used when the tempo must dynamically change within the sequence.

Shown here is a simple 4 item sequence which plays 2 notes. It is assumed that a quarter note is one beat.

1. On the first beat of the sequence, a middle C note is started. It will play Instrument #6.

2. On the second beat, a middle E note is started. It will play Instrument #12

3,4. On the 4th beat, both notes are turned off.

5. At the end marker (\$FFFF), the sequencer stops playing.

Trk	Time Stamp	Cnt	St	D1	D2
1	08 00 00 00	02	95	3C	7F
2	00 00 60 00	02	9B	40	7F
3	08 00 20 01	02	85	3C	7F
4	00 00 20 01	02	8B	40	7F
5	FF FF FF FF				

All values are in hex notation

4 Item Sequence

Call-back Routines

Since MIDI Synth is a control-orientated program, it behaves slightly different from most other tools written for the GS. With non-control type Tools, all actions are completed one at a time, in sequential order. If you call the Dialog Mgr to draw a certain dialog window, it will be drawn before before control is returned to your program. QuickDraw™ software tool will draw a circle at the time it is called, then it will return to your program. Outside events (key events and mouse movements) are queued and later pulled out and passed to your program only when you ask for them. This is because most Tools run in the foreground with your application. You call a Tool, it does what was requested, then returns back to you. On the other hand, MIDI Synth behaves in a completely opposite way of foreground type tools. Since it runs almost exclusively in the background (serviced every 5ms at DOC interrupt time), actions are not synchronized with your application program. A MIDI Synth Tool call can start a whole series of actions, all running in the background, independent of your application. A *StartNote* call only queues the request and returns to your program; the note will not be started until the next 5ms interrupt comes along. In the meantime, many more calls to MIDI Synth could be made with no immediate result. But once started, envelopes are generated, DOC registers are updated, the sequencer is serviced and many more actions are all managed in the background, sharing CPU time with your application at intervals of 5ms. Outside MIDI events interrupt the CPU when they occur and are processed by MIDI Synth. From a programmer's point of view, this foreground-background type of processing gives the application a parallel structure. It acts as though there is a separate processor running MIDI Synth in parallel with your application.

Because of this independent parallel architecture, Tool-to-program communication must be done somewhat differently. For example, when MIDI Synth needs to report a Sequencer error message to your program, it's not going to happen when you make a SeqPlayer tool call since the call returns back to your program before it even started to play the sequence. Instead of forcing you to poll at regular intervals to collect messages, MIDI Synth dispatches messages through application defined "call-back" vectors. These are application routines that MIDI Synth will call on specified events. You give it the address of your routine and MIDI Synth will call it when it needs to. If you don't need to use a specific call-back, then have its address in the call-back table set to zero.

Several points must be kept in mind when writing call-back routines. They run at interrupt time so severe restrictions are placed on your routines. You cannot make any Tool or rom calls and you have virtually no stack space available. When returning, the Stack Pointer, Data Bank and Direct Page registers must all be restored to the values they had when your routine was called (you don't have to restore X, Y and the accumulator). If you plan on using any registers, it's up to you to initialize them correctly. You will get called in native mode with index and memory set at 16 bits wide, so you must return the same. Exit the call back routine with an "RTL" instruction. Finally, you should never spend much time in a call-back routine. Most of the time, all you have to do is set a flag somewhere and exit. Let your foreground program examine the flag, and then process whatever action needs to be taken. Stealing too much time during interrupt service only deteriorates overall performance.

EndSeq

Called when the Sequencer encounters an end-of-seq marker during sequence PLAY. Does not get called if you're PLAYING and RECORDING since the sequence ends only when you stop recording.

UserMeter

This routine gets called on every beat. The number of ticks per beat is specified by the value passed with the *SetBeat* Tool call.

Mstart, Mstop

These get called when a MIDI Stop and MIDI Start message is received by the MIDI Interface.

PacketIn

MIDI Synth calls this routine every time a complete MIDI Message is received by the MIDI Interface input. The contents of the MIDI Message can be found by your routine at the *Mpacket* location in MIDI Synth's direct page (see *GetMSData* Tool call).

SeqEvent

This routine gets called every time the Sequencer processes a Seq Item. The MIDI Message contained by the Seq Item can be found at the *theSeqItem* location in MIDI Synth direct page (see *GetMSData* Tool call). The track number can be found at direct page offset *TrackSav*.

SysEx

Called while receiving a System Exclusive message. Every byte of the message, as it's received, will be passed to your routine in the accumulator. This includes the Status and EOX bytes (\$F0.....\$F7).

PacketOut

MIDI Synth will call this routine for every MIDI Message it sends out the MIDI port. The contents of the MIDI Message can be found by your routine at the *Mpacket* location in MIDI Synth's direct page (see *GetMSData* Tool call).

PgmChange

This routine gets called whenever a MIDI Program Change message is received at the MIDI port. The accumulator contains the Program value.

SMarker

The Sequencer calls this routine whenever it encounters a *SeqMarker* command. The X-reg has the bank address and the A-reg has the remaining 16-bit address of the Seq Item with the *Seq Marker* command.

Tool Call Reference

System routines

1123	<i>MSBoot</i>	Tool Locator init
2123	<i>MSStartUp</i>	Application start-up call
3123	<i>MSShutDown</i>	Application shut down call
4123	<i>MSVersion</i>	Returns the MIDI Synth version number
5123	<i>MSReset</i>	For compatibility only. Does nothing.
6123	<i>MSStatus</i>	Indicates whether MIDI Synth is active
7123	<i>GetMSData</i>	Get pointer to MS direct page
8123	<i>SetCallBack</i>	Set record of user 'Call-Back' addresses
9123	<i>MSSuspend</i>	Halt MIDI Synth update IRQ's
A123	<i>MSResume</i>	Continue MIDI Synth update IRQ's

MIDI

2723	<i>InitMIDIDriver</i>	Initialize the MIDI Driver
3823	<i>RemoveMIDIDriver</i>	Remove the MIDI Driver
4B23	<i>SetMIDIPort</i>	Select MIDI I/O state
6A23	<i>SetMIDIMode</i>	Select Omni, Poly or Multi mode
8823	<i>SetBasicChan</i>	Set the Basic MIDI channel
1223	<i>SetVelComp</i>	Sets the MIDI velocity compensation
1823	<i>SysExOut</i>	Send a System Exclusive Message out the MIDI port

Synthesizer

1423	<i>SetInstrument</i>	'Check-in' an instrument record
0823	<i>PlayNote</i>	Turn a note ON
6C23	<i>StopNote</i>	Turn a note OFF
4A23	<i>MIDIMessage</i>	Send a MIDI message to synth
5D23	<i>KillAllNotes</i>	All active notes turned OFF
	<i>SetMasterTuning</i>	Tune overall synth frequency
2423	<i>SetTuningTable</i>	Set a customized MIDI note tuning table
3523	<i>GetTuningTable</i>	Get current MIDI note tuning table

Sequencer Control

1623	<i>SetTempo</i>	Set seq tempo in Beats Per Measure
1023	<i>SetBeat</i>	Set number of clock ticks per beat

<i>SetRecTrack</i>	Select which track to record
<i>SetPlayTrack</i>	Select which tracks to play
<i>SeqPlayer</i>	Start seq play or record
<i>SetMetro</i>	Sets the metronome parameters
<i>TrackToChan</i>	Force all seq items in a track to a specified channel
<i>SetTrackOut</i>	Specify Sequencer output path

Sequencer Buffer

<i>Merge</i>	Merge two seq buffers together
<i>Locate</i>	Find 1st seq item in buffer with specified time-stamp
<i>LocateEnd</i>	Find end-of-sequence
<i>DeleteTrack</i>	Remove all items in specified track
<i>ConvertToTime</i>	Convert measures to clock ticks
<i>ConvertToMeasure</i>	Convert clock ticks to measures

\$0123 MSBoot

Initializes MIDI Synth. Called only by the Tool Locator.

Parameters None

Errors None

\$0223 MSStartUp

Starts up MIDI Synth for use by the application.

Parameters None

Errors

- \$2301 MIDI Synth already started.
- \$2303 Can't get Direct Page memory.
- \$2304 Can't get memory block.
- \$2305 Misc Tools not started.
- \$2306 Sound Tool Set not started.
- \$2307 Generator in use (MIDI Synth uses all 16 gens).

\$0323 MSShutDown

Shuts down MIDI Synth for the application.

Parameters None

Errors \$2302 MIDI Synth never started.

\$0423 MSVersion

Returns the version number of MIDI Synth.

Parameters

Stack before the call:

WORD Space for result
 ← SP

Stack after the call:

WORD Version number (\$xx.xx)
 ← SP

Errors None

\$0623 MSStatus

Indicates whether MIDI Synth is active.

Parameters

Stack before the call:

WORD Space for result
 ←—— SP

Stack after the call:

WORD Boolean TRUE if active
 ←—— SP

Errors

None

\$1F23 GetMSData

Returns MIDI Synth's direct page. This address is used primarily by call-backs to access data.

Parameters

Stack before the call:

LONG Result space.
LONG Result space
 ← SP

Stack after the call:

LONG Reserved
LONG Direct page address.
 ← SP

Errors \$2302 MIDI Synth never started.

Direct page offsets

Offset	Length	Name	Description
\$0C	WORD	<i>Mpacket</i>	MIDI input Status (low byte)
\$0E	WORD	<i>Mpacket</i>	MIDI input Data #1 (low byte)
\$10	WORD	<i>Mpacket</i>	MIDI input Data #2 (low byte)
\$EC	BYTE	<i>PacketBytes</i>	Number of data bytes (0-2)
\$12	BYTE	<i>SeqClock</i>	Seq clock fraction
\$13	LONG	<i>SeqClock</i>	Seq clock interger (high byte always zero)
\$31	BYTE	<i>SeqItem</i>	Curent Seq Item Status
\$32	BYTE	<i>SeqItem</i>	Curent Seq Item Data #1
\$33	BYTE	<i>SeqItem</i>	Curent Seq Item Data #2
\$EA	BYTE	<i>SeqItem</i>	Curent Seq Item track number
\$3F	BYTE	<i>MetroVol</i>	Metronome volume (0-\$FF)
\$E4	WORD	<i>MetroFreq</i>	Metronome frequency (0-\$7FFF)

\$1723 SetCallback

Sets the user 'call-back' vectors. Pointers with values of zero will disable the specific call-back function. See the section "Call-back Routines" for details.

Parameters

Stack before the call:

LONG Pointer to CallbackRec
 <— SP

Stack after the call:

<— SP

Errors \$2302 MIDI Synth never started.

CallbackRec

Offset	Length	Name	Description
0	LONG	<i>EndSeq</i>	Called at end-of-Sequence
4	LONG	<i>UserMeter</i>	Called on every beat
8	LONG	<i>Mstart</i>	Called when a MIDI 'Start' message is recieved
12	LONG	<i>Mstop</i>	Called when a MIDI 'Stop' message is recieved
16	LONG	<i>PacketIn</i>	Called when any complete MIDI message is recieved
20	LONG	<i>SeqEvent</i>	Called when playing a Seq Item
24	LONG	<i>SysEx</i>	Called while receiving a System Exclusive Message
28	LONG	<i>PacketOut</i>	Called when outputing a MIDI message
32	LONG	<i>PgmChange</i>	Called when 'Program Change' MIDI message received
36	LONG	<i>Mcontinue</i>	Called when 'Continue' MIDI message received
40	LONG	<i>SMarker</i>	Called when playing a SeqMarker'.
44	LONG	<i>RecBufFull</i>	Called when sequencer record buffer full.
48	LONG	_____	Reserved. Set to zero.
52	LONG	_____	Reserved. Set to zero.

\$2223 MSSuspend

Disable MIDI Synth update interrupts. See section “Interupts” for information.

Parameters

None

Errors

\$2302 MIDI Synth never started.



\$2723 InitMIDI Driver

Before MIDI Synth can use MIDI, your application must pass it a MIDI driver. Any driver that works with the MIDI Tool Set should work with MIDI Synth (See Tool Ref V3 - MIDI Tool Set for details on MIDI drivers).

Slot number

This tells the driver where the hardware for the MIDI interface is located. If you're using a SCC serial port interface, then Printer = slot 1 and Modem = slot 2. For a card interface, slot number is one of the seven physical card slots inside the GS.

Internal/External

If you're using an external MIDI interface that plugs into one of the serial ports, then set this to zero. Otherwise, if the interface is a card that plugs into an internal slot, set this to a value of one.

User ID

Memory ID used by the driver to allocate memory. You can use the same ID used to load the MIDI driver.

MIDI Driver

Your application must load the MIDI Driver from disk into memory. This is the memory location of where the driver was loaded.

Parameters

Stack before the call:

WORD	Slot number (1-7)
WORD	External= 0, Internal= 1
WORD	User ID for memory manager
LONG	Pointer to MIDI Driver
	← SP

Stack after the call:

← SP

Errors

- \$2302 MIDI Synth never started.
- \$2311 Driver already set. Call *RemoveMIDI Driver* before starting a new MIDI driver.
- \$238x MIDI driver error. See Tool Ref V3 - MIDI Tool Set for details on MIDI driver errors.

\$2823 RemoveMIDIriver

This call shuts-down the MIDI driver. Remember to dispose of any memory used by the driver after you make this call (memory was allocated with the ID passed in *InitMIDIriver*). You don't need to make this call if you used *MSShutDown*, but you still will need to dispose of the memory.

Parameters

None

Errors

\$2302 MIDI Synth never started.

\$2310 MIDI Driver was never started.

\$1323 SetMIDIPort

This call lets you enable or disable MIDI I/O. If input is enabled, all MIDI messages will now be sent to both the sequencer and synthesizer. If output is enabled, sequencer data will be sent out the specified port.

If you are not using MIDI output, make sure that it is disabled, since this will reduce CPU overhead.

MIDI Synth will not work with AppleTalk enabled. Your program must check for this at start-up time.

Parameters

Stack before the call:

WORD 0 = disable MIDI input, 1 = enable MIDI input
WORD 0 = disable MIDI output, 1 = enable MIDI output
 ←—— SP

Stack after the call:

 ←—— SP

Errors

\$2302 MIDI Synth never started.

\$2310 MIDI diver was never started.

\$0A23 SetMIDIMode

Controls how messages are sent from the MIDI port to the synthesizer. The three MIDI modes are:

0 = Omni mode

Accept messages on all MIDI channels and force them all to channel specified by the Basic Channel parameter. Messages from all channels will be played by only one Instrument since all channel numbers are changed to the Basic Channel value.

1 = Poly mode

Accept only those messages whose channels match the Basic Channel value. Ignore messages on all the remaining channels. Only one Instrument will be played since only one channel is active.

2 = Multi mode

Accept messages on all MIDI channels without modifying the channel number. Multiple Instruments can simultaneously play in Multi mode by sending MIDI messages through different channels. Each channel will play a different Instrument. The Basic Channel is ignored in Multi mode.

Parameters

Stack before the call:

WORD Omni, Poly or Multi mode (0-2)
 <— SP

Stack after the call:

<— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$0923 SetBasicChan

Sets the MIDI Basic channel. This is the channel used while in POLY or OMNI modes.

Parameters

Stack before the call:

WORD Channel number (0-15)
 ← SP

Stack after the call:

 ← SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$1223 SetVelComp

Increase the velocity value on all 'Note On' MIDI messages by the amount specified. This affects only those messages received through the MIDI port.

Parameters

Stack before the call:

WORD Velocity offset (0-127)
 ←— SP

Stack after the call:

 ←— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$1823 SysExOut

Send a MIDI System Exclusive message out the MIDI port. You must pass a pointer to a complete Sys Ex message starting with the Status byte and terminating with an EOX byte (\$F0 \$F7). All data bytes in between must have their MSB set to zero (\$00 - \$7F).

The Delay parameter specifies the number of 5ms tick delay between byte outputs. Since many MIDI devices will lose data if you send the message at full speed (Delay = 0), this parameter should usually be set to a minimum value of 1 tick delay.

When you make the SysExOut tool call, control won't return to you until the complete message is sent out. In the meantime, if you need to monitor the message output progress, you can pass the address of your monitor routine. This routine will get called after every single byte (except for the EOX) is sent out. Return in native mode via an RTL instruction.

Parameters

Stack before the call:

LONG	Pointer to complete Sys Ex message (\$F0 \$F7).
WORD	Delay. Number of 5ms intervals between byte output.
LONG	Address of user Monitor routine. Set to zero if unused.

← SP

Stack after the call:

← SP

Errors

\$2302	MIDI Synth never started.
\$230E	Message error
\$230B	MIDI output disabled.

\$1423 SetInstrument

Send an Instrument Record over to MIDI Synth. An internal copy of the Instrument Record is kept by MIDI Synth, so after this call is made, you can dispose of the memory used by the Instrument Record. Since MIDI Synth uses the internal copy to play an instrument, any changes made to your Instrument Record will not be heard at the output unless you make this call again with the new modified Instrument Record (once the Instrument is set by this call, MIDI Synth makes no further references to your original Instrument Record).

Parameters

Stack before the call:

LONG	Pointer to Instrument Record
WORD	Instrument number (0-15)
	<— SP

Stack after the call:

<— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error

\$0B23 PlayNote

Start playing a note with the specified instrument, pitch and volume. The channel number specifies which one of the 16 Instruments to play.

Parameters

Stack before the call:

WORD	Channel number (instrument 0-15)
WORD	MIDI note number (0-127)
WORD	Key velocity or volume value (0-127)
	<— SP

Stack after the call:

<— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error
\$230B Message queue full

\$0C23 StopNote

Stop playing the note with the specified instrument and pitch. This call will have no effect if the note is no longer active.

Parameters

Stack before the call:

WORD	Channel number (instrument 0-15)
WORD	MIDI note number (0-127)
	← SP

Stack after the call:

← SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error
\$230B Message queue full

\$1A23 MIDI Message

Send a MIDI Channel Message to the synthesizer, sequencer or MIDI port. The Destination parameter specifies where the message goes in MIDI Synth.

Destination:

- 0 - Message plays only the synthesizer.
- 1 - Message plays the synthesizer and will also get recorded if the sequencer is currently recording MIDI data.
- 2 - Send the message out the MIDI port.

Parameters

Stack before the call:

WORD	Destination (0-2)
WORD	Number of valid data bytes (0-2)
WORD	MIDI Status Message (\$8x - Ex)
WORD	Data byte #1 (0-127)
WORD	Data byte #2 (0-127)
	← SP

Stack after the call:

← SP

Errors

- \$2302 MIDI Synth never started.
- \$230A Parameter range error (does not check the MIDI message)
- \$230B Message queue full
- \$230D MIDI output disabled
- \$230F MIDI output buffer is full

\$0D23 KillAllNotes

Turns OFF all active synthesizer notes.

Parameters None

Errors \$2302 MIDI Synth never started.

\$2423 SetTuningTable

Set a customized MIDI note tuning table (see “Tuning Table” section for details). MIDI Synth makes an internal copy of this table, so after this call no further reference is made to the external table.

Parameters

Stack before the call:

LONG Pointer to custom note tuning table (128 WORDS)
 ←— SP

Stack after the call:

 ←— SP

Errors

\$2302 MIDI Synth never started.

\$2523 GetTuningTable

Get the current MIDI note tuning table (see “Tuning Table” section for details).

Parameters

Stack before the call:

LONG Pointer to 128 WORD buffer
 ←—— SP

Stack after the call:

 ←—— SP

Errors

\$2302 MIDI Synth never started.

\$1623 SetTempo

Set sequencer tempo. Tempo values are specified in Beats Per Minute (BPM) minus 10. In other words, a value of zero gives the minimum tempo of 10 BPM while a value of 255 results in the maximum tempo of 265 BPM.

Parameters

Stack before the call:

WORD Tempo in BPM-10 (0 - 255)
 ←—— SP

Stack after the call:

 ←—— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$1923 SetBeat

Set the number of Seq Clock ticks per beat. This value is normalized to 96 ticks to a quarter note. This value is used to determine when to call the user routine for the Beat Call-Back. Typical values are shown below:

o d ↓ ♪ ♫ ♮

w	h	q	e	x	r
384	192	96	48	24	12

Beat Duration Value

Parameters

Stack before the call:

WORD Beat duration in Seq Clock ticks (1-65535)
← SP

Stack after the call:

← SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$0E23 SetRecTrack

Sets which one of the 16 tracks is marked for recording. The Track field in all new Seq Items will be set to this value while recording.

Parameters

Stack before the call:

WORD Track number (0-15)
 ← SP

Stack after the call:

 ← SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$0F23 SetPlayTrack

Sets the state of the specified track for playback. A boolean TRUE ($\neq 0$) makes the track active, while a FALSE ($=0$) turns the track off during playback. Any number or combination of inactive/active tracks can exist at any given time.

Parameters

Stack before the call:

WORD	Track number (0-15)
WORD	Track Play state
	<— SP

Stack after the call:

<— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$1523 SeqPlayer

Starts and stops the sequencer play and record functions.

The Sequencer always assumes that the sequence starts on a beat. All functions related to the beat will be synchronized to the first item in the sequence.

Careful attention must be given to the *theClock* value when PLAYING or RECORDING. It is the caller's responsibility to align the *PbufStart* pointer correctly into the PLAY buffer so that it is pointing to a Seq Item which is time-stamped with a value equal or greater than *theClock* value. Use the result from *Locate* call before any PLAY or REC to properly set *PbufStart* if you wish to start at a specified Seq Clock time. See example in the section "Using MIDI Synth".

Parameters

Stack before the call:

LONG Pointer to SeqPlayerRec
 ←—— SP

Stack after the call:

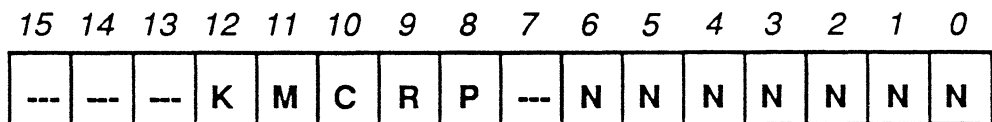
 ←—— SP

Errors \$2302 MIDI Synth never started.

SeqPlayRec

Offset	Length	Name	Description
0	LONG	<i>PbufStart</i>	Pointer to PLAY buffer
4	LONG	—	Reserved, set to zero
8	LONG	<i>RbufStart</i>	Pointer to REC buffer
12	LONG	<i>RbufEnd</i>	Pointer to last byte of REC buffer
16	WORD	<i>SeqFlags</i>	SeqPlayer Flags
18	LONG	<i>theClock</i>	The Seq Clock gets set to this value.

SeqFlags



- K** 1 = Key wait enabled 0 = Disabled
- M** 1 = Metronome On 0 = Off
- C** 1 = MIDI clock 0 = Internal clock
- R** 1 = RECORD On 0 = Off
- P** 1 = PLAY On 0 = Off
- N** Number of beats to count off (0-127)
- Reserved. Set to zero

Key Wait

The Sequencer does not begin a RECORD or PLAY request until after the MIDI Interface receives a MIDI Note Off message. This message does not get recorded.

Metronome

Sounds the internal metronome on every beat (see SetBeat call). The Sequencer must assume that the sequence always starts on a beat, so the metronome is heard immediately when the sequence starts.

Clock

When set to the *MIDI clock*, the Sequencer waits for MIDI Timing messages to advance the seq clock, synchronizing itself to an external MIDI device. The Sequencer automatically advances the seq clock itself when set for *internal*.

Count Off

The Sequencer will wait the specified number of beats before starting. The beat call-back (*Mupdate*) and metronome are still active during this period.

\$1E23 SetMetro

Sets the metronome parameters. Default parameter value will be used if parameter value set to zero. Metronome Wave must be 512 bytes in length.

Defaults: Volume = \$FF
 Frequency = \$0180 = 384
 Wave = Wood Block

Parameters

Stack before the call:

WORD	Metronome volume (0-\$FF)
WORD	Metronome frequency (0-\$FFFF)
LONG	Pointer to Metronome Wave
	<— SP

Stack after the call:

<— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$1023 TrackToChan

Forces specified sequencer track to play on specific channel when playing to the synthesizer. Since each channel is assigned a different instrument, this essentially forces all Seq Items in a track to play the specified instrument. A Channel number with a value of \$FFFF will put the track in a “thru” mode. This means that no channel translation will occur. Seq Items will be sent as messages to the synthesizer with their original channel numbers unaltered.

Parameters

Stack before the call:

WORD	Track number (0-15)
WORD	Channel number (0 - 15, \$FFFF)
	<— SP

Stack after the call:

<— SP

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$2623 SetTrackOut

Sets the Sequencer output path for the given track.

Parameters

Stack before the call:

WORD Track number (0-15)
WORD PathValue (0 - 2)
 ← SP

Stack after the call:

 ← SP

PathValue

- 0 *Send Seq output to both the Synthesizer and the MIDI port.*
- 1 *Send Seq output to only the MIDI port.*
- 2 *Send Seq output to only the Synthesizer.*

Errors

\$2302 MIDI Synth never started.
\$230A Parameter range error.

\$1C23 Merge

Merge two Seq Buffers together. Buffer 1 is merged with Buffer 2, with the combined result left in Buffer 2. Make sure that enough space is available beyond Buffer 2 to hold the added Items from Buffer 1. The buffers are merged so that all Seq Items from both buffers are organized with increasing time-stamp order. This call can be used at the end of a **SeqPlayer RECORD** function to merge the REC buffer with the existing PLAY buffer to form a new PLAY buffer that includes all the newly acquired Seq Items in the REC buffer.

Parameters

Stack before the call:

LONG Pointer to Buffer 1
LONG Pointer to Buffer 2

←— SP

Stack after the call:

←— SP

Errors \$2302 MIDI Synth never started.

\$1123 Locate

Finds the address of the 1st Seq Item in buffer with with specified time-stamp. If it can't find a match, then it will return with the address of 1st item with a value greater than the specified time-stamp. If all Seq Items have time-stamps less than the given time, a pointer to the EOS (end-of-seq) will be returned. This call can be used before a **SeqPlayer** call to find a starting location in the PLAY buffer.

Parameters

Stack before the call:

LONG	Result space
LONG	Time-stamp to match (Seq Clock ticks)
LONG	Pointer to Seq Buffer
	<— SP

Stack after the call:

LONG	Pointer to Seq Item
	<— SP

Errors

\$2302 MIDI Synth tool never started.
\$230A Parameter range error.

\$1B23 LocateEnd

Finds the end of a sequence. This call will return a pointer to the byte following the EOS (end-of-seq) marker.

Parameters

Stack before the call:

LONG	Result space
LONG	Pointer to sequence
	←— SP

Stack after the call:

LONG	Pointer to location following the EOS marker
	←— SP

Errors

\$2302 MIDI Synth never started.

\$1D23 DeleteTrack

Delete all Seq Items from specified track.

Parameters

Stack before the call:

WORD Track number (0-15)
LONG Pointer to sequence

← SP

Stack after the call:

← SP

Errors

\$2302 MIDI Synth never started.

\$2023 ConvertToTime

Converts measures to Seq time (Seq Clock ticks).

Parameters

Stack before the call:

LONG	Result space
WORD	Ticks per beat (0-\$FFFF)
WORD	Beats per measure (0-99)
WORD	Beat number (0-99)
WORD	Measure number (0-999)

← SP

Stack after the call:

LONG	Seq Clock ticks
------	-----------------

← SP

Errors

\$2302 MIDI Synth never started.

\$230A Parameter range error.

\$2123 ConvertToMeasure

Converts Seq time (Seq Clock ticks) to measures.

Parameters

Stack before the call:

WORD	Result space
WORD	Result space
WORD	Result space
WORD	Ticks per beat (0-\$FFFF)
WORD	Beats per measure (0-99)
LONG	Seq Clock ticks

<— SP

Stack after the call:

WORD	Measure number (0-999)
WORD	Beat number (0-99)
WORD	Remainder Seq Clock ticks

<— SP

Errors

\$2302 MIDI Synth never started.

\$230A Parameter range error.