converter

release 1.0

> > >

realtime midi performance system

user documentation

documentation version 1.5

contents

appendixes

foreword

This software was created as a prototype tool for direct midi datastream control and conversion of non-midi controller sources. As an experimental software system, certain parameters and functionality within the software has remained open in an attempt to avoid limitations in its potential applications; as a result, it is possible for it to be programmed in ways which do not conform to conventional midi specifications. A certain amount of knowledge and understanding of the underlying concepts of midi as well as its lower-level implementation is required in order to take full advantage of the type of control and power that converter provides. To learn about the midi data protocol specification and related concepts, refer to the midi format appendix. To truly understand converter and how to make it perform to its best abilities, **read this document carefully** – there are a lot of important pieces of information in various places of this document (many not so obvious).

Even though an experimental software program, **when configured properly converter provides extremely reliable and robust performance**. The nature of its user interface tends to avoid possibilities for crashes or other errors. Any functionality problems will almost entirely be a result of user programming errors or improperly configured interfaces. However, as the PC world represents denizens of manufactures of every type of hardware component, there are some potential caveats – refer to the troubleshooting appendix if you encounter technical difficulties.

Why on earth DOS? Why not Windows?

The DOS operating system was chosen for its combination of reliability, efficiency, and elegance. The low operating overhead of DOS enables real-time performance comparable to that of dedicated outboard hardware implementations, a requirement for practical applications in live performance and musical interactivity. DOS can provide this level of performance with **much lower hardware requirements** than an equivalent Windows application, and this provides an effective utility for older ‘obsolete’ computing hardware eclipsed by current trends in operating systems and technology. While consumer soundcards (and especially older soundcards such as the Soundblaster 16) provide audio quality which is substandard for professional applications in recording, converter provides an application in which these older soundcards and computers can find a new life within the context of current studio work – thus avoiding the garbage dumps. Considering a Pentium-233MHz machine can currently be purchased for about $150 CDN (late 2001), older computers make great cheap embedded systems or dedicated machines for a multitude of purposes.

During testing it was found that converter behaved unpredictably when run from within a DOS shell under Windows, due to the way it directly programs some of the computer’s hardware. Therefore, converter must be executed in “true” DOS mode (without Windows operating in the background).

A note about the graphics system

The graphics routines in converter are a **low priority** task, since handling the midi data stream and the related input conversion and processing is the most important task the computer has to do. As a result, the response time of the graphics system may create the visual appearance of a slight computational lag on slower systems. This doesn’t correspond to a lag associated with the actual midi stream. With regards to audio conversion using smaller buffers, often a couple buffers or more will pass by and be processed before the computer is able to update the screen, resulting in the visual effect of the peak meter “floaties” bouncing up from an audio peak that wasn’t reflected by the actual peak meter itself (the slower the computer, the more noticeable this effect). Keep in mind that updating the screen is at the bottom of the list in terms of computing importance, and by the time the screen reflects the ‘current’ input state, the processed or generated midi data has already been transmitted and received by the intended outboard device. If the system has a heavy processing load, some of the graphics functions (such as the midi text box or oscilloscopes) may automatically update slowly or cease to be updated; this is to ensure the best possible real-time performance.

A note about simultaneous input sources

Even though converter offers four sources of input – midi, audio, gameport and mouse – midi and audio input cannot be used simultaneously on one computer as a result of hardware interrupt handling issues at this time. The software provides two modes of operation: midi, joystick, and mouse input, and audio, joystick and mouse input.

Most importantly….

When development on converter was started, it was never intended for eventual release; making it available to the public is somewhat of an experiment to see if it generates any interest. Since this application is provided at no charge for any type of use (commercially or otherwise), **please help us help you by giving us feedback!** If you find it useful, please let us know what you think of it, what kinds of projects you've used it on, what types of computers / interface hardware you have, and any other information and/or suggestions. If we receive enough interest in converter and this approach to real-time music software, there are all sorts of other free software projects similar to this that could be launched, such as real-time multi-effects – but there's not much point if nobody cares.

Rest assured that any information we receive remains completely confidential, including personal email addresses.

Thanks for checking out converter, and I hope it at least meets if not exceeds your expectations! Feel free to send any comments, corrections or suggestions to me at **sonic@urr.ca**

Tom Roscoe

urr Sound Technologies Inc.
www.urr.ca

features and uses

Features

- extremely low-latency system response time similar to equivalent dedicated outboard hardware
- 2 channels of audio input, each with 2 additional filter channels for a total of 6 independent sources for audio to midi conversion, all with individual conversion modes, intelligent data thinning algorithms (for continuous controller conversion), and arithmetic processors
- audio can be converted to midi in both trigger/gate as well as continuous controller modes
- gameport and mouse to midi conversion providing 12 additional channels of variable and binary midi control, allowing for the design and implementation of custom midi control surfaces
- user-customizable midi data display panel
- programmable midi input processor allowing for 144 complete data operations in realtime, via a total of 432 conditional verifiers and 576 data transforms

Uses

- add additional controller functionality to midi equipment with limited controller programmability
- provides new possibilities in sampling, allowing a synthesized sound to adopt the rhythm or inflections of another sound in realtime (for instance, impose a rhythmic pulse to a filter on a sound on a synthesizer, based on a drum loop passed into the audio input in converter)
- enables sounds on an external midi device to be triggered from audio input peaks, for example a mic'd snare drum could trigger a sampled snare sound etc.
- adds midi control to non-midi devices such as turntables, microphones, etc.
- trigger sound or other events from objects on a stage in a live theatrical performance
- provides new and interesting uses for otherwise abandoned legacy (i.e. "obsolete") hardware
- provides a useful tool for debugging or testing midi systems / implementations

system requirements

Different aspects of the software require more processing power than others in order to deliver fast response time. Even though conceptually simple, converter must perform a tremendous number of computing instructions per second in order to provide the kind of programming flexibility it does. As a very rough guideline, a 486DX-33 system is recommended as a minimum for midi processing or 2 channel audio conversion (non-filter channels), and a Pentium-based PC at 100MHz or greater (233MHz ideal) for full audio support (including filter channels). If you are using a significantly slower machine and performance seems to be impacted, there are many configurable options described in this manual that will assist in maximizing the performance of converter. Note that a math coprocessor is required for 486-based computers. The machine used for development was a Pentium 233, with an ATI Rage IIC video card, Gravis Ultrasound 'Classic', Soundblaster AWE64, and a Roland MPU-IPC-T. Due to certain wait-states converter must use to properly communicate with the hardware, performance is not substantially increased by using PentiumII or faster CPUs.

- 486DX-33MHz based PC as a minimum, Pentium 100MHz to 233MHz recommended
- DOS (probably 4.0 or higher)
- SVGA video card with 512k of RAM
- about 540k of free conventional memory (the base 640k that DOS uses)
- one of the supported sound or midi interface cards or compatible alternative – see next page.

There are several standards for interface cards which converter supports. These standards are implemented on a variety of cards; however testing for each potentially compatible card is impossible due to the sheer number of interfaces on the market. The following guide outlines the interface standards implemented in converter, the specific interfaces which were used in the development of converter (and hence are directly known to work), and any information about cards which typically wouldn't work. **If you are successful using an interface card that is not listed here as specifically known to work, or you find a card that should work but doesn't for some reason, please contact me so it can be added to this list and assist other users in the future.**

Roland MPU-401

DEVELOPMENT INTERFACE: ROLAND MPU-IPC-T, MUSIC QUEST MPU-401 CLONE

Any 100% MPU-401 UART-mode hardware compatible midi interface card is supported by converter. This includes the Roland family of interfaces (MPU-401, MPU-IPC-T, LAPC-1, etc.), and theoretically should include various devices from MusicQuest, Turtle Beach, Terratec, and many others. Note that the 'intelligent' mode of the MPU-401 is **not** used – only the UART mode.

Users with a soundblaster 16 (or newer) card should use the soundblaster midi mode of converter, since it will automatically configure itself for your soundblaster's MPU-401 interface based on the SET BLASTER line in your autoexec.bat file.

Gravis Ultrasound

DEVELOPMENT INTERFACE: ULTRASOUND 'CLASSIC' (GF1)

There are several variations of the Gravis Ultrasound ("gus") card based on two different main processor chips. The first series of cards were based upon the GF1 processor, and as such are: Ultrasound ('classic'), Ultrasound Max, Ultrasound Ace (**not** recommended), and Ultrasound Extreme. The more recent Ultrasound card, called either the Ultrasound Plug & Play or Ultrasound Plug & Play Pro, is based upon the AMD Interwave chip. Either the Ultrasound 'classic' or Ultrasound Max will work with converter (with the SET ULTRASND line in your autoexec.bat file set correctly – the installation software should do this automatically). The Ultrasound Extreme and Ultrasound P&P series *may* work (we have no way of testing these cards), while the Ultrasound Ace will **–not-** work (it has no ability to record audio, and no hardware midi interface).

Soundblaster 16 (or newer)

DEVELOPMENT INTERFACE: AWE-64, AWE-32 (both ISA-bus)

There are a tremendous number of variations of the Soundblaster family of soundcards. Early Soundblaster cards (Soundblaster, Soundblaster 2.0, Soundblaster Pro) are not supported since they lack a proper midi interface, and cannot support the audio requirements of converter. Newer PCI-based Soundblaster cards (Soundblaster pci-128, Soundblaster pci-512, Soundblaster Live!) are not supported since they do not have complete soundblaster 16 compatibility – it is provided through a memory-resident driver that doesn't support midi or audio input. This may or may not apply to the newer SB16P&P-PCI. **In general, it is safest to go with an ISA-bus card for true DOS software compatibility**; some PCI-bus cards (perhaps Terratec?) may provide a memory-resident SB16 compatibility driver that *does* implement emulated hardware-level midi and audio input – let us know if you have success with one. Cards known to work are the Soundblaster 16, Soundblaster 32, Soundblaster AWE-32, Soundblaster AWE-64 and Soundblaster AWE-64 Gold.

Standard Analog Gameport Interface

DEVELOPMENT INTERFACE: GAMEPORT ON GRAVIS ULTRASOUND, SOUNDBLASTER AWE-64, AWE-32

The gameport interface that converter supports is the original IBM-spec standard analog gameport, which was originally available as an add-on card, and eventually became a standard component of soundcards as they started being manufactured. Therefore, if you have a Gravis Ultrasound or a Soundblaster 16 soundcard, you also have a standard analog joystick port. Newer types of gameport interfaces, such as advanced force-feedback digital interfaces, are not supported - however these newer advanced types are much less common.

Standard Microsoft or Logitech-compatible Mouse (with appropriate driver)

DEVELOPMENT INTERFACE: SERIAL-PORT CIRQUE GLIDEPOINT TRACKPAD, LOGITECH PS/2 WHEEL MOUSE, LOGITECH PS/2 OPTICAL WHEEL MOUSE, GENERIC SERIAL MOUSE

Any mouse or pointing device which is supported by an appropriate DOS mouse driver is supported by converter, as it interfaces with the DOS driver that either came with the device or is downloadable from the manufacturer's website. Since most manufacturers completely adhere to the Microsoft/Logitech mouse interface standard, most devices should work automatically. Make sure that you load the mouse device's driver each time you boot into DOS in order for converter to detect the mouse – a good idea is to place the command to run the driver (ie. `mouse`) in your `autoexec.bat` file, so the driver is automatically loaded each time DOS is booted.

To ‘install’ converter, simply create a directory (folder) somewhere on the computer’s hard drive, and simply copy all the files contained in the converter .zip archive to that folder. Or alternatively, copy the files to a blank 1.44MB 3½” floppy disk (see the appendix on creating a bootable floppy).

Before running converter, the file `hardware.cfg` should be examined to ensure the settings are appropriate for your system. This can be loaded using any text editor – in Windows you can use Notepad, or under DOS you could use Edit). Make sure not to change the layout of the file (don’t delete any lines, even blank ones).

If the default settings (listed below) suit your hardware, there is no need to edit this file.

The file `hardware.cfg` looks like this (default settings):

```
# HARDWARE SETTINGS CONFIG FILE
#
# NOTE! Do not add text or otherwise change the line spacing of this file.
#       Just edit the number for the parameter you need to change.
#       Altering the layout of this file may cause the software to not load.
#
# SOUNDCARD TYPE
# 0 = Roland MPU-401 ............. (( MIDI  INPUT ))
# 1 = Gravis Ultrasound .......... (( MIDI  INPUT ))
# 2 = Gravis Ultrasound .......... (( AUDIO INPUT ))
# 3 = Sound Blaster 16 or newer .. (( MIDI  INPUT ))
# 4 = Sound Blaster 16 or newer .. (( AUDIO INPUT ))
4

# AUDIO BUFFER SIZE (in bytes)
# should be no smaller than 12 and no greater than 512
128

# SETTINGS SPECIFIC TO MPU-401 ONLY
#
# MPU-401 IRQ
# possible settings: 2,4,5,7,9
2

# MPU-401 BASE ADDRESS
# possible settings: 300, 330, 360, etc
330

# END OF CONFIG FILE.
```

The first parameter allows you to select the soundcard and what input mode you want to use (audio input or midi input – converter cannot perform both midi and audio input simultaneously due to hardware interrupt-handling issues at this time).

The second parameter only affects audio mode, and allows you to define the size of the audio buffer. This determines both the response time for audio to midi conversion, as well as the quantity of midi data generated from the audio input (in continuous controller mode). Since each buffer of data is converted into a single amplitude value, the smaller the buffer size, the greater number of times it is converted per second – and conversely, the larger the buffer size, the fewer number of times it is converted. The smallest buffer size you can use is determined by your computer’s processing speed; smaller buffer sizes result in significantly more instructions per second and I/O functions that the computer must complete in order to avoid losing audio samples, and will have an impact on the speed of the graphics functions. The side effect of using a very small buffer is that the software will actually trace individual cycles of lower-frequency waveforms, resulting in midi continuous controller streams that oscillate at a very fast rate, much like an LFO on a synthesizer. This ability has been preserved for those wishing to experiment with these types of oscillations as control signals. To avoid this type of effect, try using an input buffer size of 128 bytes or greater – experimentation is key. For most applications, a buffer size below 96 (and certainly below 64) bytes is a bit overkill. **Typically, a buffer size between 96 and 192 bytes is recommended.** Note also that with a buffer size less than 72 bytes, gameport input is automatically disabled – refer to the gameport system overview for further details.

The last two parameters are only for those using a Roland MPU-401 or compatible for midi input. MPU-401 autodetection routines were not implemented, so in order for converter to locate your interface you must tell it which base address and IRQ the card is assigned to. The factory default for an MPU-401 is base address 330 at IRQ 2/9.

Once you have customized these settings to your specific needs, save the file and you're ready to run converter.

loading / running converter

Unfamiliar with DOS?

If you are using Windows9x on your computer, you can press [F8] just as Windows begins to load in order to bring up a boot menu, from which you can select "command prompt only" to boot into 'pure DOS'. Or, you can boot out of Windows9x by selecting 'Shut Down' from the Start menu, and selecting "Restart in MS-DOS Mode". In general, the best approach for running converter on a Windows9x machine is to create a self-booting floppy disk with converter installed on it; using this approach enables converter to be automatically run with its own autoexec.bat settings by simply inserting the floppy and booting (or re-booting) your computer. See the appendix in this manual that discusses this procedure.

If you installed converter on your hard disk: After booting into DOS, change to the directory (folder) you've installed converter by using the cd command, either `cd..` to move back one directory level, or `cd converter` to move into a directory (where converter is the name of the directory). Once in the appropriate directory, run converter by simply typing `c` and pressing enter.

Problems?

- If converter fails to load and function properly, make sure to check the troubleshooting appendix of this manual – there may be something about your computer's configuration that has been overlooked.

When you first run converter, you will notice a subdivision in the screen layout: a grey midi message display panel above a blue display panel that can be set to display several different pages of system and input information. By pressing the Escape key, a menu system ‘screen’ slides down over the lower display panel, giving access to the menu system which contains all the programmable parameters that define how converter operates. A convenient feature of this menu screen is that it can be retracted and restored (to check performance indicators on the display panel hidden behind the menu screen) without losing your place in the menu system hierarchy. To quickly return to the root menu page at any time, hold down [Alt] and press [Esc].

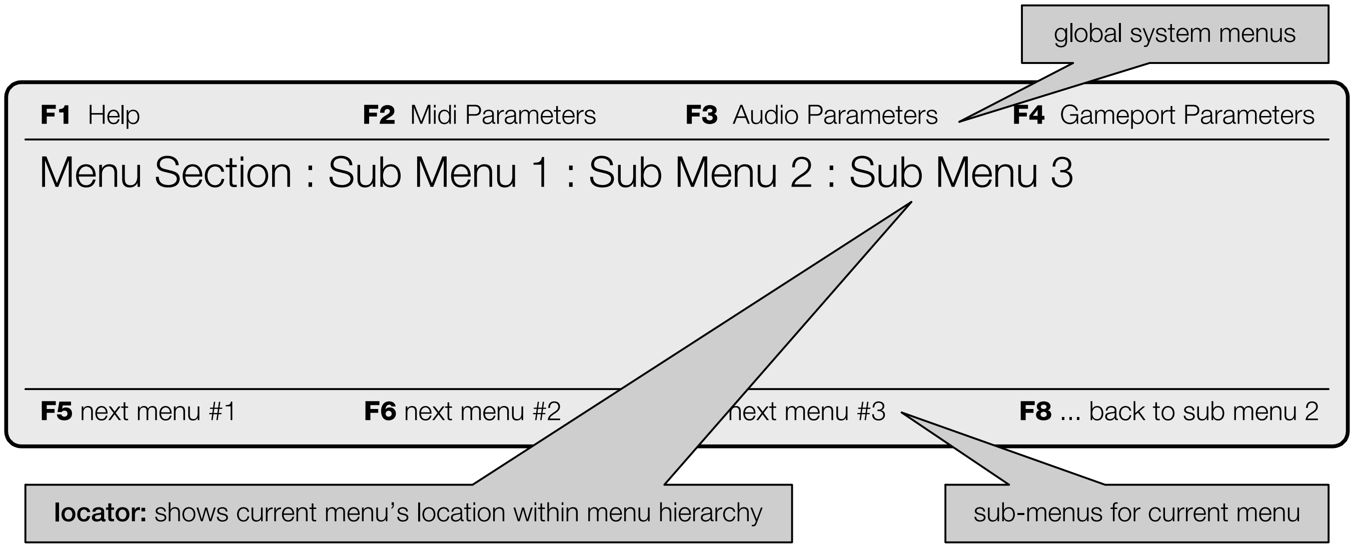
The upper display panel provides real-time display of midi data both generated (from audio and gameport inputs) and received (from a midi input interface). The upper left corner of this panel indicates midi activity on a particular input source, active channels in the output midi stream, and a processor buffer meter which roughly indicates system computing load (based on how heavily the computer is relying on its various processing buffers). Because of the prioritization of the conversion and midi output systems, a high processing buffer load doesn’t normally reflect any additional lag in system response time; however it does give an indication of how hard the computer is working to provide continuous output in addition to the real-time graphics system. For example, if converter is operating in audio input mode with all channels active, the processor buffer will indicate higher buffer usage when the lower display panel is in audio oscilloscope mode than one of the other display modes due to the increase in required graphics computing power.

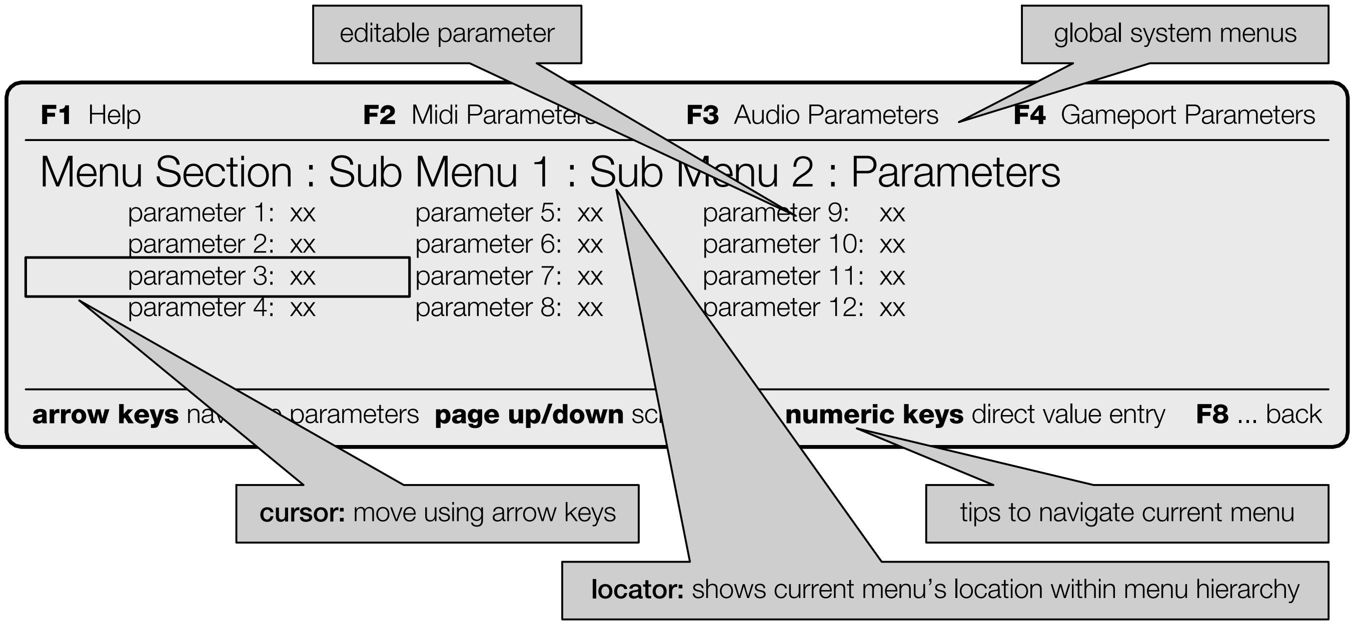
The lower display panel can display four different pages of system and input information, accessible by pressing function keys [F9] through [F12]. One or two of these pages may not display any active information depending on the mode in which converter is set to operate (midi or audio input mode).

- **midi note messages [F9]** : displays peak meters for all 128 notes, indicating note on and off messages with their note number and key velocity, and indicates whether top and bottom display panels reflect processed or unprocessed input midi data (relevant only in midi input mode).
- **audio input oscilloscopes [F10]** : displays waveform plotting (oscilloscopes) for each of the six audio channels, with associated peak meters. This display page takes the most graphics horsepower to maintain – if the redraw speed is slow, try setting the **oscilloscope redraw** parameter to intelligent (both parameters are in display settings, accessible from the root menu by pressing [F7]). More information on accellerating system performance can be found in an appendix towards the end of this manual.
- **hardware & system settings [F11]** : displays hardware settings for the selected audio or midi interface card, the data rate of midi transmission (useful to avoid overloading midi bandwidth), the filename for the current settings file and whether or not it has been edited since being saved last, input information from the gameport, and audio peak meters for each of the six audio input channels.
- **midi data format reference [F12]** : displays a quick reference card covering commonly used midi messages and their data format. For specific information on continuous controllers such as the modulation wheel, refer to the charts at the end of this manual.

The menu system is navigated by the other eight function keys, which serve as direct hot keys to the menu (even when the menu screen is retracted). The top and bottom of the menu screen indicates what menus the function keys access – the top four never change, and represent discrete sections of the menu system, while the bottom four reference sub-menus which allow you to navigate through the chosen section’s menu layers. In other words, the menus and parameters for midi input are access via [F2], for audio input [F3], and gameport input [F4]. Simple reference help is accessed by pressing [F1]. All four of these function keys can be pressed at any time from any location in a menu system, for quick access to another menu section.

The bottom four function key labels represent sub menus accessible from the current position in the menu hierarchy by pressing function keys [F5] through [F8] – their assignments change as you navigate your way around. When entering an option screen with a list of editable parameters, this bottom field changes to indicate the key presses which can be used to edit the displayed parameters. However, the function keys can still be pressed to access neighboring menus directly , without having to back up by pressing [F8] first. Note that you don't have to wait for animations to complete before pressing a key or entering data – try 'double-clicking' the function keys. The following diagrams explain the components which comprise the retractable menu screen.


global system menus
F1 Help
F2 Midi Parameters
F3 Audio Parameters
F4 Gameport Parameters
Menu Section : Sub Menu 1 : Sub Menu 2 : Sub Menu 3
F5 next menu #1
F6 next menu #2
next menu #3
F8 ... back to sub menu 2
locator: shows current menu's location within menu hierarchy
sub-menus for current menu



The Midi Message Textbox

To the right of the top portion of the screen is a scrolling textbox which, if enabled, shows the generated or converted midi messages. Within the textbox there is two columns of text, one which serves as a text label for the message type, with the particular midi message's relevant numerical value in the right column. Each message is formatted in the following way for greater intelligibility:

- Statusbyte messages (non-realtime, so in other words voice or channel messages as well as system exclusive) are displayed in bright blue, and the number associated with the message represents the midi channel the message specifies. For example, **CONTROLLER: 8** would represent a controller message on midi channel 8.
- Data bytes following a statusbyte message are displayed in medium-intensity blue, and the number associated with the message represents the value associated with the databyte.
- Realtime or system messages (such as clock) are displayed in dark blue, and the number to the right of the message is the specific statusbyte value of the message.

If midi input mode is used and the view processed data parameter in the realtime settings menu is set to off, the midi message textbox displays the incoming midi data values and not the values post-processor.

converter file types

There are three external files used by converter which serve different purposes – all three of these must be located in the same directory (folder) as the executable file (c.exe). The first file, `hardware.cfg`, has already been discussed as it is a text file which holds the individual computer-specific hardware settings for the soundcard used. The file `default.set` is the base settings file which is read from when converter is initially run, and updated (written to) when converter is quit. The final file, called `c.cfg`, is read from at run-time and written to on exit, and is used to store other computer-specific hardware settings adjustable from within converter. In this way, programmed converter configurations (`.set` files) can be interchanged with other users, while machine-specific settings remain tied to the particular machine on which converter is being used. At the same time, `c.cfg` and `default.set` provide something akin to a 'battery-backed memory' so that the last converter session, with all its settings, is automatically re-loaded and ready to be used the next time converter is run.

parameter settings saved exclusively in c.cfg

- display settings (vertical retrace, animation settings, oscilloscope settings, scrolling text window status, midi input processor view mode, etc.)
- input device settings (joystick axis channel calibrations, mouse axis speed)

file listing of the converter package

The following is a listing of all the files included in the converter source zip archive, and their relationship with the main program file. Files noted as required need to be in the same directory (folder) as the converter program file (`c.exe`) in order for it to operate properly. Files noted as optional are not needed for proper operation of converter.

file	description	dependancy
`c.exe`	converter program file (executable)	required
`c.cfg`	converter system configuration file	required
`hardware.cfg`	soundcard interface settings	required
`c1.cdt`	converter data file	required
`c2.cdt`	converter data file	required
`c3.cdt`	converter data file	required
`c4.cdt`	converter data file	required
`c5.cdt`	converter data file	required
`default.set`	default program file loaded when converter is initially run	required
`manual.pdf`	manual for converter (this document)	optional
`autoexec.bat` `(and reboot.com)`	a basic autoexec.bat file which can be used for a bootable floppy disk installation of converter	optional
`audiojoy.set`	preset program file	optional
`2chancc.set`	preset program file	optional
`6chancc.set`	preset program file	optional
`trigmain.set`	preset program file	optional
`trigflt.set`	preset program file	optional
`empty.set`	preset program file	optional

keyboard layout and hotkeys

accessible at any time

[Esc]	pop-down / pop-up menu system display (remembers current menu page)
[Alt] – [Esc]	direct hotkey to the root menu
[Alt] – [L]	direct hotkey to load program
[Alt] – [S]	direct hotkey to save program
[Alt] – [A]	direct hotkey to audio midi data settings
[Alt] – [G]	direct hotkey to gameport midi data settings
[Alt] – [M]	direct hotkey to mouse midi data settings
[Alt] – [X]	quit application
[Ctrl] – [Q]	alternative way to quit application
[Scroll Lock]	enter standby mode (halts all midi transmission – essentialy a ‘pause’ button)
[+]	increment current display midi channel
[-]	decrement current display midi channel
[Ctrl] – [+]	increment program change, transmit on current display midi channel
[Ctrl] – [-]	decrement program change, transmit on current display midi channel
[*]	toggle between view all channels and selected single midi channel
[/]	clear display
[z]	toggle midi message textbox on/off (unless menu cursor is on a label parameter)
[r]	reset midi output running status (re-transmit statusbyte)
[spacebar]	transmit midi panic (note off messages for all 16 channels and controller reset)
[F9]	midi note meter display panel
[F10]	audio oscilloscope display panel
[F11]	hardware and system info / joystick & audio input display panel
[F12]	midi data format reference display panel

accessible within the menu system display

arrow keys	move cursor
number keys	direct value entry
alphabetic keys	direct label entry
[pg up]	increment parameter value
[pg dn]	decrement parameter value
[Enter]	select program, operation or file / toggle parameter on or off
[F1] – [F4]	select global menu section
[F5] – [F8]	navigate current menu system

accessible within midi program & operation selection and settings menu pages

[Alt] – [c]	copy program / operation
[Alt] – [p]	paste copied program / operation

accessible when the menu system display is retracted

[Enter]	toggle between view processed midi data and raw input data display modes

note: When the [Alt] or [Ctrl] keys are referred to in a key combination such as [Alt] – [Esc], it is meant that the [Alt] key should be held down while the [Esc] key is pressed and released.

menu pages and parameter descriptions

The following pages of this manual are segmented in five sections, discussing the global settings for converter, followed by its audio, midi, gameport, and mouse input functionality. Each section devotes a page to each individual page in the menu system, and outlines the parameters found on it and describe their usage. At the top of each page, the menu page's title, location in the menu system heirarchy, as well as the key presses to access it from the root menu are listed.

The menu system employs animations to the text. Keys can be pressed at any time – there is no need to wait for any animations to complete before the next key is pressed. To speed up, slow down, or turn off the animations, refer to the page discussing the **display settings** menu page.

disk i/o

Root : disk i/o

[AltEscape – F5]

Program files store all the midi processor programs, audio, gameport and mouse settings, input source configurations, several of the global system settings as well as the custom viewpanel settings. These program files are stored in the same directory as the converter software; the program files are identified by their '.set' extension. A feature of converter is that the current program is automatically saved at exit into the file `default.set`, and immediately re-loaded upon re-launching the program – so if the same system configuration is always used, converter is always ready to operate immediately.

load program

Allows the selection of a program file (.set) to be loaded for operation. There is no support for directory hierarchies in the load function – all 'programs' have to be stored in the same directory as converter itself (not great but that feature is (hopefully) most likely not necessary)

This menu can be directly accessed at any time by pressing [alt]-[L].

save program

Allows you to save your programming work - useful feature! Just type in the filename (the filename input box doesn't allow you to use your own extension - .set is automatically added to the filename) and press enter.

This menu can be directly accessed at any time by pressing [alt]-[S].

turn midi logfile on/off

Enables or disables the continuous storage all input or generated midi data messages to a file called `logfile.txt` in the same directory as converter. This text file can be loaded in any text file viewer (such as Edit under DOS or Notepad under Windows), and its contents are formatted in the same way as in the scrolling message textbox on screen. This feature is best left off, and is merely provided for those using converter as a means to debug midi message streams generated by other midi devices, or some other type of technical work.

Note that this feature is unavailable when converter is being run from a floppy disk due to the inherent lack of speed of the floppy disk drive – converter must be run from a hard disk for this function to be visible.

realtime settings

root : realtime settings

[AltEscape – F6]

midi input processor
Enables or disables (bypasses) the midi processor, which is used only for incoming midi data. This parameter can be left at the 'on' position at all times, unless it is desirable to transmit all incoming midi data directly to the midi output.

view processed data
Selects whether the display panels show data after or before passing through the midi processor. By setting this parameter off, the raw input (unprocessed) midi data is displayed.

generate midi clock
Enables or disables the midi clock generation functionality of converter. This feature is provided as a quick and dirty clock source for applications like testing, or situations requiring a quick syncronization source, and is not meant to serve as or replace an exceptionally accurate master clock source for a midi system. The clock may provide less accurate timing when the system has a very heavy task load.

default clock tempo
defines the tempo of the midi clock generator, in BPM. Valid values are between 20 and 250.

audio input
Enables or disables all audio input (including the audio filter channels).

audio left filter / audio right filter
Enables or disables the left or right filter channels. By disabling one or both of these channels, system speed is increased by disabling the realtime IIR filter algorithms which require the cpu to perform a tremendous number of calculations per second. Unless the respective filter channel is being used (or your computer is fast enough that performance is not affected), it is recommended that the channel(s) be disabled.

Note: main audio input must be enabled in order to use the filter channels. If midi data conversion from the main channels is not desired, simply set their conversion mode to 'no conversion' in the **audio midi data settings** menu.

gameport input
Enables or disables input from the PC's gameport. By disabling this input, system speed is increased, due to the computing-cycle-wasting requirements of the analog gameport. Unless gameport input is being used, it is recommended that gameport input be disabled. Also note that gameport input is automatically disabled when converter is run in audio mode with a buffer size less than 72 bytes – otherwise audio interrupts occur so frequently that converter cannot accurately acquire a proper axis reading.

mouse input
Enables or disables input from the PC's mouseport. Mouse to midi conversion doesn't create a significant computing load; however, unless used, mouse input should be disabled to avoid any undesired midi messages transmitted by accidentally moving the mouse, etc.

display settings

root : display settings

[AltEscape – F7]

update screen
Selects one of three screen update settings: **on** (everything is refreshed), **minimal** (only the system activity panel is updated), and **off** (none of the graphical elements are updated in real-time). By selecting off or minimal, the computer completely focuses exclusively on performing midi processing or conversion operations. In general, converter’s graphics code is performed at a low priority respective to the time critical processing and/or conversion, so typically this parameter can be left set to “on”. However if the only machine you have to use is an old 386DX system or you otherwise feel the computer is severely bogged down and could perform better with a lighter load, and you don’t need to see what the computer’s doing, this option may improve system response times.

vertical retrace
Enables or disables screen redraw synchronized to the monitor’s vertical retrace interrupt. Enabling this parameter will result in less flicker from screen updates in certain situations; however it can slow down the speed of the graphics routines when there is a high processing load. A alternative way to reduce flicker is to set the oscilloscope redraw parameter to ‘intelligent’ (see below).

animations
Enables or disables user interface animations.

animation speed
Sets the speed at which the animations are performed.

midi message textbox
Enables or disables the scrolling midi message textbox at the right of the display panel. The updating of the text messagebox is only performed when free computing cycles are available (as is the case with all of the graphics functions), therefore having it enabled shouldn't affect performance; however it is sometimes benefical to disable this as it does add additional load to the graphics system. In many applications (such as continuous controller audio to midi conversion) the stream of generated midi data will scroll by too quickly to read, and so computing cycles are wasted on a flurry of illegible text.

oscilloscope redraw
Selects the mode in which the oscilloscopes are redrawn. In **intelligent** mode, the oscilloscopes are updated only when a midi message is generated from the current input(s). In **continuous** mode, the oscilloscopes are updated continuously, even when the entire system is idle and there is no input signal. Intelligent mode provides a way to reduce unnecessary flicker in the oscilloscope displays, however continuous mode is more suitable for situations such as trigger mode where little midi data is actually generated from the audio inputs.

translate numerics
Determines whether or not converter displays midi statusbytes and note numbers in their numerical format or as a statusbyte label with channel number or note letter with octave for the various parameters within the menu system which directly reference midi bytes. By enabling this option, for example, parameters will display a statusbyte as “poly aft. [ch. 12]” for a polyphonic aftertouch message on midi channel 12 rather than that same statusbyte’s actual numerical representation of 171, as well as displaying “C octave 4” rather than that note number’s numerical value of 60.

customize user panel

root : customize user panel

[AltEscape – F8]

panel meter (A – E)
Selects the midi controller message type to monitor. This provides a visual monitor for controller types not assigned to existing hardwired display elements on the upper portion of the screen.

Audio to midi conversion in converter is performed on the basis of amplitude, in one of two modes: a continuous time-varying conversion mode for continuous controller applications, and a trigger/gate conversion mode useful for using an audio source to trigger midi note on / note off messages. **Note that only the line-input is used on all the soundcards – the mic input is not supported.** On the Soundblaster 16 and compatibles (as well as the Gravis Ultrasound), converter automatically sets the level for the line input to its maximum position so that an external mixer program is not needed.

Each audio channel generates one value byte, based on the amplitude of the incoming audio, which can be inserted into any data byte in a particular midi message type. The associated midi message data which encapsulates this converted value byte is defined in the audio parameters menus.

As an example, the following steps would program the audio conversion engine to generate a continuous stream of modulation wheel data (continuous controller) from the left (non-filter) input channel. This is assuming that the main audio input is enabled and converter is operating in audio input mode.

Step 1: program the midi conversion settings

- go to the audio parameters menu section (F3), press [F7] for the midi conversion settings menu, then [F5] for midi data settings, and finally [F5] again for the main audio channels parameters.
- type 2 for the left conversion byte position parameter, 176 for the left statusbyte (or use the page up / page down keys to scroll until the parameter reads “cntrl [ch. 1]”), then type 1 for the left 1st databyte value, and leave the left 2nd databyte value at zero. This tells converter that the converted value byte should be used for the 2nd data byte in the midi message, and that the type of midi message to generate is a controller message on midi channel 1 (by using 176 as the status byte). By setting the 1st data byte in the message to a 1, we have specified the controller message type to be modulation wheel.
- move the cursor to the left convert mode parameter and choose continuous controller mode (“cont. ctrl”), then select “no conversion” for the right convert mode parameter.

Step 2: specify the midi data reduction settings (optional)

Since continuous-controller type of audio conversion can generate a tremendous number of midi messages per second, especially if more than one audio channel is being used, it can be useful to program the data thinning algorithms to intelligently reduce the density (messages per second) of the generated midi stream.

- press [F8] twice to back up in the menu heirarchy, then press [F6] for data reduction settings.
- there are two parameters for each of the six audio channels; the parameters for the main audio channels (non-filter channels) are the first four on the left of the menu screen. Try a value of 10 for the left continuous controller reject parameter (labelled “left cc reject” on the menu), and a value of 7 for the left channel reject threshold parameter (labelled “left reject thresh.”). These settings mean that out of every 10 audio byte conversions, only 1 is transmitted, unless there is a sudden change in the audio amplitude that exceeds the threshold (which we set to 7). If the latter happens, a midi message is transmitted instantly (data rejection does not occur) in order to preserve the musically important transients of the audio.

That’s all there is to it. If for some reason no data is being generated (and yet you can see audio input in the oscilloscope or system info display pages), check to make sure the input gates are not interfering by setting the left threshold parameter to 0 in the main audio settings menu. Additionally, you may want to experiment with the arithmetic processing that can be applied to the channel allowing such processing as amplitude compression or expansion (using a combination of division and addition or multiplication and subtraction operands respectively).

The following is an example of how to program converter in trigger mode, so that impulses on the left main audio channel channel will trigger a note with velocity on midi channel 1.

Step 1: program the midi conversion settings

- go to the audio parameters menu section (F3), press [F7] for the midi conversion settings menu, then [F5] for midi data settings, and finally [F5] again for the main audio channels parameters.
- type 2 for the left conversion byte position parameter, 144 for the left statusbyte (or use the page up / page down keys to scroll until the parameter reads “note on [ch. 1]”), then type 60 for the left 1st databyte value, and leave the left 2nd databyte value at zero. This tells converter that the left main audio channel converted value byte should be used for the 2nd data byte in the midi message (representing note velocity in this case), and that the type of midi message to generate is a note on message on midi channel 1 (by using 144 as the status byte). By setting the 1st data byte in the message to a 60, we have specified the note number (middle C in this case).
- move the cursor to the left convert mode parameter and choose trigger mode (“trigger”), then select “no conversion” for the right convert mode parameter.

Step 2: program the audio input settings

- press [F3] to quickly return to the beginning of the audio parameters menus, followed by [F5] for the main audio settings menu. The three parameters we need to check and / or modify are the “left trigger level”, “left release level”, and “left trig decay time”.
- the left **trigger level** parameter controls the input gate, in that the audio signal’s amplitude must exceed this level before a midi message is generated from it. This can be set to any value from 0 to 127; however with a value of 127, audio will never be converted because 127 is the max level. With a value of 0, audio at all amplitude levels is converted (the gate is effectively disabled). This feature is useful to separate wanted signals from unwanted signals (ie., separate signals from background noise, or eliminating accidental triggering from other instruments bleeding into the microphone such as a hi-hat bleeding into a snare drum mic). This can be set at 0 for now, and adjusted if the source being tracked requires a different setting.
- the left **release level** parameter determines the level below which the audio amplitude must fall before a zero-value midi message is generated (in this case, note on velocity zero which is equivalent to a note off message). This can be any value from 0 to 127; however if set to zero, a zero-value midi message will never be generated. Since the notes will be stuck on without a corresponding note off – type message, set this value to at least 1.
- The left **trigger decay time** parameter determines the minimum time window (in milliseconds) between a trigger value midi message (in this case note on) and its subsequent zero-value midi message. This parameter is useful to avoid rapid re-triggering of notes, especially if using a very small audio buffer size where lower frequency waveforms may be traced. Note however that this parameter can limit the minimum note length that converter will track at a given tempo / bpm for the particular channel. For example, the higher the number used, the fastest note length converter will accurately track will move down from 64th notes, down to 32nd notes, down to 16th notes, etc. Try a setting below 150 to start, and adjust as necessary.

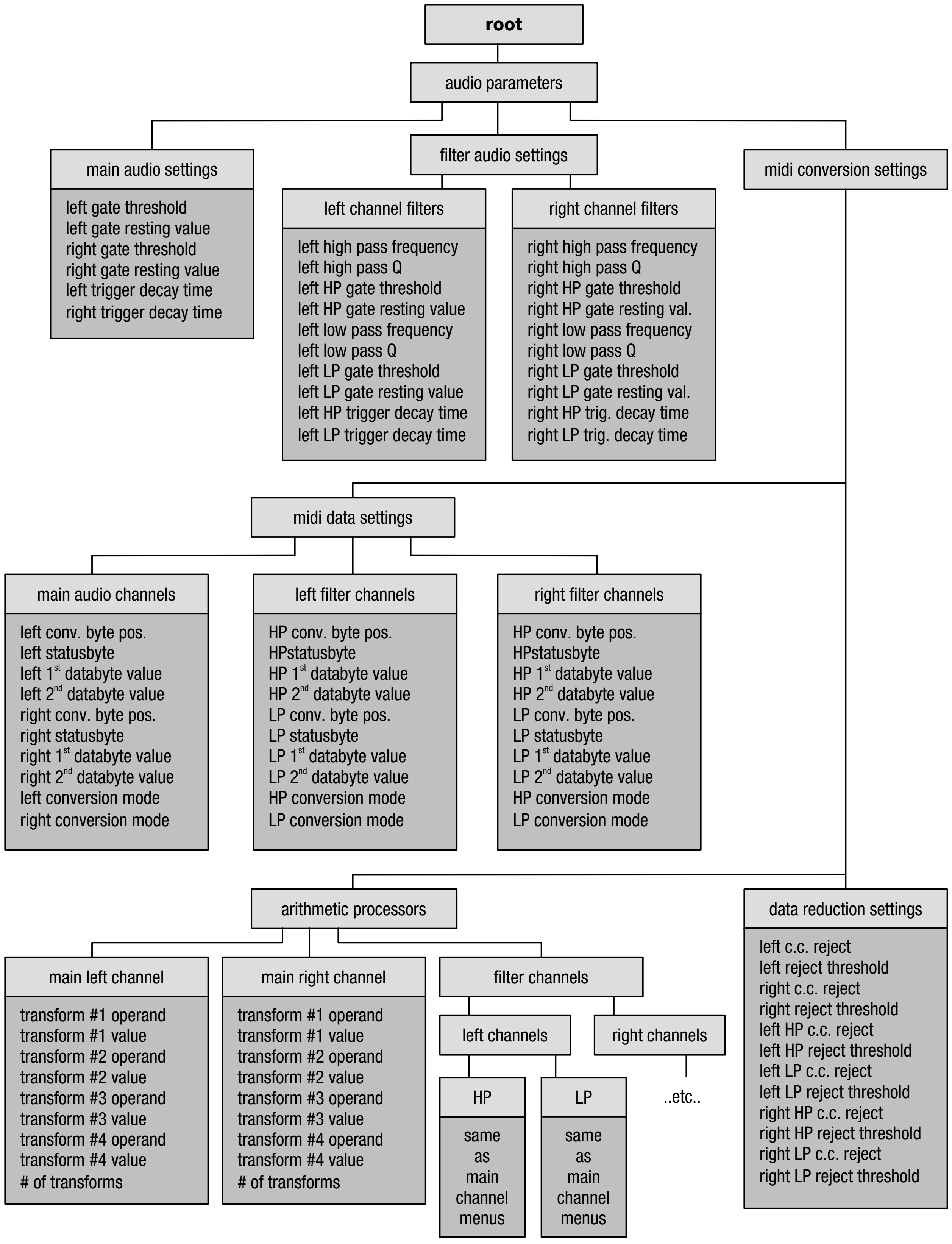
And that’s it – converter is programmed to perform triggering based on audio taken from the left audio input. This same methodology can be applied to the right channel, or any of the filter inputs. Note that sometimes it is beneficial to use the filter channels for conversion work rather than the main input channels, as the filters provide the ability to further separate wanted from unwanted trigger signals. If this approach is used, make sure to set the main audio channels to “no conversion” in the main audio midi conversion settings menu – since the filter channels are only active when the main audio inputs are enabled in the root realtime settings menu, disabling the main input conversions helps to avoid any unwanted midi messages from being generated.

Additionally, you may want to experiment with the arithmetic processing that can be applied to the trigger channel, through which one can create customized velocity response curves and the like.

Since the input audio is never heard, it is not necessary to avoid clipping the signal – in fact, it can be useful to overly increase the input level of the audio in certain applications. In trigger mode, it is usually better to set the signal level so as to avoid constant input overload: the signal level should be set to ensure the audio peaks rise and fall within the trigger gate thresholds, to avoid mis-firing.

Obviously, audio sources vary significantly in terms of bandwidth and amplitude, so the key is to experiment wth different source material and try different ways to program the conversion process. Take advantage of the gates on each audio channel to help separate signal from noise or other unwanted sounds. You may find that using an external equalizer to further filter out unnecessary frequency sections of an audio source or boost specific frequency areas will greatly improve the possibility for signal separation between the low and high pass filter channels. Keep in mind that trigger mode conversion works best and most reliably with a non-mixed audio source – for example, a single track consisting of a snare drum rather than a channel from a stereo mixdown with many layers and instruments. Trigger mode was designed more for applications such as triggering drum samples from a multi-tracked acoustic drum performance or a live mic'd drumkit than from a dynamically-compressed mixdown.

The following is a structural map of the menu hierarchy for the audio input menu section (F3 – audio parameters), which may help familiarize the location of the various parameters.



Depending on the audio conversion mode selected for a particular channel, one of two different sets of parameters will be visible for that channel in the audio input settings menu.

trigger decay time

When using a smaller input buffer for audio, or if converting a low-frequency waveform, the system may analyze the audio input quickly enough to generate a trigger pulse for each cycle of the input waveform. To avoid this rapid re-triggering, a minimum time constant (in milliseconds) can be established within which a subsequent re-analysis of the audio input will not be performed. This parameter can be ignored if trigger mode conversion is not being performed for the particular channel. As an example, settings under about 150ms are good for tracking percussion instruments.

WHEN CHANNEL IS SET TO CONTINUOUS CONTROLLER CONVERSION MODE:

threshold

This parameter determines the gate cutoff level for the respective channel (left / right), for which the audio signal's amplitude must cross before being converted, and below which audio will not be converted to midi information. As an example, this is useful for converting audio signals which have a higher residual noise component than usual. Possible values are in the range of 0 – 127.

resting value

Typically, when the audio falls below the gate cutoff value (threshold) for a particular channel, that channel's conversion value equals 0. The resting value parameter allows the zero-audio level to be any value from 0 – 127, useful for continuous controller applications where the controller should rest at a midpoint of 63 (such as the pitch bend controller), or default to a max value at 127.

WHEN CHANNEL IS SET TO TRIGGER CONVERSION MODE:

trigger level

This parameter specifies what amplitude the audio must cross (above) before a trigger event will be generated. Essentially the same as the threshold parameter in continuous controller conversion mode, the range of possible values are between 0 and 127.

release level

This parameter specifies the amplitude level below which the audio must fall before a release event (ie. note on velocity zero) will be generated. Possible values are between 0 and 127, however setting this parameter to zero will result in a release event never being generated (the audio amplitude cannot fall lower than zero!).

audio parameters : filter audio input settings : left channel/right channel

[F3 – F6 – F5/F6]

hp frequency
Selects the cutoff frequency in Hz for the high-pass filter for the particular audio input channel. Possible values are between 1000Hz and 10512Hz.

lp frequency
Selects the cutoff frequency in Hz for the low-pass filter for the particular audio input channel. Possible values are between 350Hz and 1000Hz.

hp Q / lp Q
Selects the Q (or steepness) of the filter cutoff. Possible values are between 1 and 15.

trigger decay time
When using a smaller input buffer for audio, or if converting a low-frequency waveform, the system may analyze the audio input quickly enough to generate a trigger pulse for each cycle of the input wave. To avoid this rapid re-triggering, a minimum time constant (in milliseconds) can be established within which a subsequent re-analysis of the audio input will not be performed. This parameter can be ignored if trigger mode conversion is not being performed for the particular channel.

WHEN CHANNEL IS SET TO CONTINUOUS CONTROLLER CONVERSION MODE:

threshold
This parameter determines the gate cutoff level for the respective filter audio channel (hp / lp). Possible values are in the range of 0 – 127. Refer to the previous page for more information.

resting value
The resting value parameter allows the zero-audio level to be any value from 0 – 127. Refer to the previous page for more information.

WHEN CHANNEL IS SET TO TRIGGER CONVERSION MODE:

trigger level
This parameter specifies what amplitude the audio must cross before a trigger event will be generated. Essentially the same as the threshold parameter in continuous controller conversion mode, the range of possible values are between 0 and 127.

release level
This parameter specifies the amplitude level below which the audio must fall before a release event (note on velocity zero) will be generated. Possible values are between 0 and 127, however setting this parameter to zero will result in a release event never being generated (the audio amplitude cannot fall lower than zero!).

audio parameters : midi conversion settings : midi data settings : main/filter channels

[F3 – F7 – F5 – F5/F6/F7]

conversion byte position

Selects which databyte of the chosen midi message the converted value is placed in. For example, to use a channel of audio input to control the pitch wheel, the generated message would consist of one statusbyte and 2 databytes as follows: the statusbyte (indicating a pitch wheel message on midi channel X), followed by the first databyte representing the LSB component of the pitch wheel message (unused in this case since we can only provide an 8-bit value, therefore should be set to a value of zero), followed by the second databyte representing the MSB value for pitch wheel. Hence you would use a value of 2 (to indicate that the value converted from the audio input signal should be placed in the 2nd databyte of the midi message). Since the statusbyte is not considered a data byte, this parameter should be read in terms of databyte number, not byte number in the cumulative message packet (i.e. 2 instead of 3). Possible byte position values are either 1or 2; if set to zero, no databytes are transmitted for the particular midi message.

Note that not all midi messages use both, or any, data bytes; channel aftertouch, program change, and song select only use the first databyte (for aftertouch amount, program change number and song number respectively), while system realtime messages such as midi clock simply use the statusbyte. As converter automatically enforces these midi definitions, setting this byte position parameter to 2 when the message type only supports one data byte results in the converted value not being a part of the generated midi message.

statusbyte

Determines the midi message type (and channel) to use for the audio conversion value. Possible values are between 128 and 255.

1st databyte value

Sets a fixed value for the first databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 1st databyte position.

2nd databyte value

Sets a fixed value for the second databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 2nd databyte position.

convert mode

Selects the mode in which to perform audio to midi conversion for the particular channel. In continuous controller mode, audio amplitude is tracked to provide time varying continuous controller values. In trigger mode, the audio is monitored to generate a midi message when the audio's amplitude exceeds the gate's threshold, with a respective counter-message generated when the audio falls back below the gate's threshold. Typically, trigger mode conversion is used to generate note on / note off messages from such sources as percussion sounds, etc.

audio data reduction settings

audio parameters : midi conversion settings : data reduction settings

[F3 – F7 – F6]

The parameters on this particular menu page only impact continuous controller audio to midi conversion; trigger mode audio to midi conversion does not employ (or require) data reduction.

continuous controller reject (cc reject)

Selects the ratio of converted values to transmitted midi messages. For instance, with a value of 0, every analysis value is converted into a midi message and transmitted (most often resulting in a very dense stream of midi data, especially if all six channels of audio are being converted with the same stream density). With a value of 4, only the fourth value is converted into a midi message and transmitted. Possible values are 0 to 99, with a value of zero disabling data reduction.

reject threshold

This parameter controls the amplitude change upon which data rejection is temporarily abandoned in order to allow fast peaks to be transmitted immediately. This allows for data to be thinned when there is little change in the amplitude of the input waveform, yet gives priority to sudden and musically important shifts in amplitude. Possible values fall between 0 and 127; with a value of zero or one, any change in amplitude is given priority which effectively disables data reduction.

Note that the arithmetic operations are performed sequentially in the order they appear in the transform list, from first to last – ‘order of operations’ is not followed. This allows for the input data to be processed in a more flexible manner.

transform operand (X-form)

Selects the type of data transform to be performed on the value converted from the audio input signal. Possibilities are: no xform, add, subtract, multiply, and divide.

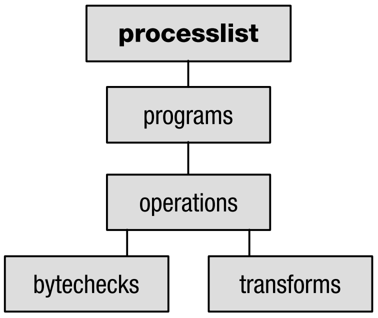
transform value

Selects the numeric value for the transform. For instance, if the chosen transform is divide, and the transform value is 3, all converted values will be divided by three. Possible values are between 0 and 127 (protection is implemented against division by zero).

# of transforms

Determines how many transforms are active, counting from the first transform. This makes it easy to bypass certain transforms and only activate the first 1, or 2, or 3 transforms. If set to zero, none of the transforms will be performed on the converted data.

The architecture of the midi processor is structured into 3 levels: the **process list**, which is the list of programs performed by the processor; a **program**, which represents a group of **operations;** and the **operations** which directly perform the midi data modification through a number of **bytechecks** (or conditional verifiers) and **transforms**. Upon receiving a complete midi message, the midi processor cycles through the process list to see if the midi message type should be modified by any of the operations in any of the programs.



The specifics of this architecture are as follows:

- the process list can contain up to 12 programs
- each program can contain up to 12 operations
- each operation consists of 1 statusbyte comparator, 3 data byte checks and 4 transforms
- each operation functions on one specific midi message type (status byte value)
- more than one program or operation can modify a given midi message (useful if the desired modification requires more than 4 transforms to complete, such as sysex)

To illustrate how to apply this system to a practical task, let's consider the design of a program to perform a keyboard split function at middle C for incoming midi data on channel 1 that will send all note messages below middle C out on channel 4.

There are two ways in which keyboards transmit note information; one involves the use of both note on and note off messages, and the other involves only note on messages which are considered note offs if their velocities are equal to zero. The program should be designed to support both methods of note transmission in order to function correctly with all keyboard controllers; we will design our program to actually convert note on messages with a velocity of zero to actual note off messages to illustrate one of the ways in which converter can be used to change the formatting of the midi stream.

Here's how to implement our keyboard split program:

- select an empty program slot in the process list (F2), in our case it should be slot #1 (if there are other programs already on the processor and you don't know what they are, you can start with a 'blank slate' by pressing [Alt]-[L] and loading EMPTY.SET)
- go into the program settings menu (F5), label the program as "keyboard split", and set the **number of operations** parameter to 3.

Now we have our program. The next step is building the three operations we need for this task:

- enter the operations list for this program (F6) and select the first slot.
- go to the operation parameters menu (F5), label the operation as "note on ch. 1", set the **message type** to 144 (note on channel 1), set the **# of byte checks** to 2 and the **# of transforms** to 1.
- go to the bytecheck parameters menu for this operation (F6). Set byte check #1 byte number (the **BC#1 byte number** parameter) to 1 – this tells the processor to check the first data byte in any note on message whose statusbyte matched our operation's message type (we chose 144). In this case, the 1st data byte of a note on message is the note number.
- set byte check #1's **check operand** parameter to the 'is in range' condition, the **low value** parameter to 0, and **high value** parameter to 59. This specifies that any note value below middle C will fit this condition.

- set byte check #2’s **byte number** parameter to 2 (the note velocity data byte), the **check operand** parameter to ‘is in range’, the **low value** parameter to 1 and the **high value** parameter to 127. By doing this, we have provided the additional condition that only note on messages with a velocity higher than zero will be accepted for transformation (in this case, sent for output on midi channel 4).
- go to the transform parameters menu (F7), set the transform #1 byte number (the **X#1 byte number** parameter) to zero (the status byte), set the **transform operand** parameter to the ‘set to’ function, and type 147 into the **transform value** parameter. This first transform changes the statusbyte of all midi messages that matched our two byte checks to 147 – a note on message on channel 4.

Now we have to create a second operation to convert note on messages with a velocity of zero under middle C into actual note off messages, and transmit them to channel 4 to avoid stuck notes.

- Press [alt] – [c] to copy the current operation.
- go back to the operations list by pressing F8 twice. Move the cursor over the next empty operation slot. Press [alt] – [p] to paste the operation we just created and copied into this new slot. Select our new operation, enter the operation parameters menu, and change the **operation name** to ‘note on v. zero’.
- go to the bytecheck parameters menu [F6], and change byte check #2’s **byte number**, **check operand**, and **low value** and **high value** parameters to 2, ‘is equal to’, 0, and 0 respectively. Leave the first byte check as is (since we are still working with notes below middle C). The second byte check singles out all notes with a velocity of zero. Note that the high value parameter is unused when the check operand is in equality mode (‘is equal to’).
- go to the transform parameters menu [F7], and ensure that transform #1’s **byte number**, **operand**, and **value** parameters are 0, ‘set to’, and 131 respectively. This transforms a note on velocity zero message from channel 1 into a note off message on channel 4.

Finally, we must implement an operation to send the appropriate incoming note off messages to channel 4 so as to avoid stuck notes that never receive a note off message.

- go back to the operations list by pressing F8 twice. Move the cursor over the next empty operation slot. Press [alt] – [p] to paste the note on operation still in memory. Select our new operation and enter the operation parameters menu. Change the **operation name** to ‘note off ch. 1’, the **message type** to 128 (note off channel 1), **# of byte checks** to 1 and **# of transforms** to 1. By setting the # of byte checks parameter to 1, we de-activate the second bytecheck which is not needed in this case. For clarity’s sake, you might want to set the parameters for byte check #2 to zero.
- go to the transform parameters menu, and change transform #1’s **byte number**, **operand**, and **value** to 0, ‘set to’, and 131 respectively. This transforms the note off message’s statusbyte to a note off message on channel 4.

The midi processor is now programmed to perform a midi split on all incoming note data on midi channel 1. If it doesn’t seem to be working, make sure the processor is turned on in the realtime settings menu (by pressing [alt-escape] followed by [F6], or by pressing [F8] to back up through the menu hierarchy until the root menu appears).

From this example, you can see how additional transformations on the notes below middle C could be performed. For instance, a transform could be set up to transpose those valid note messages down an octave (or any value), add an offset to their velocity to ensure a minimum volume, or change a keyboard’s velocity response curve by using the multiplication or division operands on the velocity data byte. Just remember that if you transform the note number or channel of note on messages, you must perform the same transformations to their respective note off messages - both actual note off messages and note on messages with a velocity of zero; otherwise, notes will get stuck on from never receiving a message telling them to turn off.

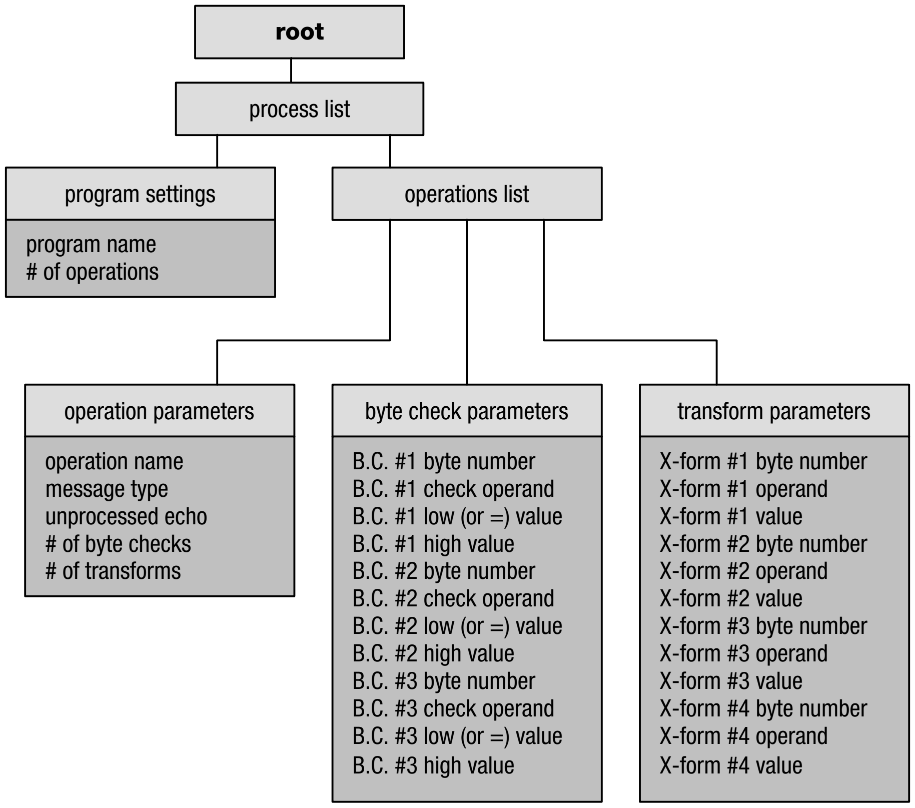
Both the process list and operations list should be programmed sequentially from slot 1 to slot 12; in other words, it's not crucial but it's a good idea not to leave empty slots between active programs or operations, and to set the **number of operations** parameter (in the program settings menu) to the number of operations actually used in the respective program. This avoids unnecessary additional processing time and reduces the potential for mistakes in operation programming.

The copy and paste feature was demonstrated in the midi split example above. Both operations as well as entire programs can be copied and pasted into other slots to speed up process list programming. From within either the process list or related program-specific menus, the selected program (including all its operations) can be copied by pressing [alt] – [c], and pasted by pressing [alt] – [p]. The same feature exists for copying individual operations; from either the operations list or from within any of the operations parameter pages, an operation can be copied using [alt] – [c] and pasted using [alt] – [p].

Note that incoming system realtime messages, such as clock, active sensing, system reset, etc. are not currently modifiable by the midi processor, even though their statusbytes can be selected. However, a non-sytem realtime message can be *converted* into a system realtime message by the the midi input processor.

For additional information on the specifics of midi, as well as quick reference charts on the data formatting of various midi messages, refer to the midi appendix at the end of this manual.

The following is a structural map of the menu hierarchy for the midi input menu section (F2 – midi parameters), which may help familiarize the location of the various parameters.



midi processor program settings

midi parameters : program settings

[F2 – programselection – F5]

program name

Edits the name for the selected processor program.

# of operations

Selects the number of operations used in the program. All operations, byte checks, and transforms are performed sequentially starting with the first. This parameter allows you to disable the use of certain operations within the program; for instance if there are 6 operations and this parameter is set to 5, operation 6 will not be performed on the incoming midi data. This parameter should not be set to a value higher than the number of programmed operations, in order to both reduce processing time on slower computers and reduce the chance for possible user programming errors.

midi processor operation parameters

midi parameters : program settings : operation settings : operation parameters

[F2 – programselection – F6 – operationselection – F5]

operation name
Edits the name for the selected operation.

message type
Selects the statusbyte that identifies the messages to be processed by the current operation. Any midi message that begins with this statusbyte will pass through this operation (and be transformed if the additional ‘bytechecks’ (conditional verifications) are met). Possible values are between 128 and 255.

unprocessed echo
Enables or disables transmission of the original unprocessed version of the midi message (selected by the current operation and byte checks) to the midi output accompanied by the new processed message. This feature, as an example, would permit a layered effect if used in a midi split program such as described previously in the programming tutorial – simply enable unprocessed echo for all three of the programmed split operations, and the lower half of the keyboard will transmit both on channel one as well as four.

# of byte checks
Defines the number of conditional verifications to perform on the incoming midi message. This parameter can be used to disable certain conditional verifications in the same way as the **# of operations** parameter in the midi program settings menu.

# of transforms
Defines the number of transforms to perform on the incoming midi message. This parameter can be used to disable certain conditional verifications in the same way as the **# of operations** parameter in the midi program settings menu.

midi processor byte check parameters

midi parameters : program settings : operation settings : byte check parameters

[F2 – programselection – F6 – operationselection – F6]

Each operation has three independent byte checks (conditional verifiers), each of which can perform a comparison on any databyte of a midi message (even the same databyte as one of the other comparators). Because each operation has a dedicated byte check for the statusbyte (through the message type parameter in the operation parameters menu), these byte checks only reference the data bytes of a midi message. Each of the conditional verifiers have the following parameters:

byte check byte number (BC byte number)

Selects which databyte in the midi message to perform the conditional verification on. Possible values are between 1 and 16.

byte check operand

Selects what type of comparison to perform on the selected data byte. The possible selections are a range comparison, an equality comparison, and no comparison. The range comparison could be used to select note on messages of a certain range of the keyboard, or within a certain velocity range, for processing. The equality comparator could be used to select an individual note.

byte check low (or = to) value

Selects the low cutoff for the range comparator, or the value for the equality comparator. Possible values are between 0 and 127.

byte check high value

Selects the high cutoff for the range comparator; this parameter is unused for an equality comparison. Possible values are between 0 and 127.

midi parameters : program settings : operation settings : transform parameters

[F2 – programselection – F6 – operationselection – F7]

Each operation has four independent transforms which are performed sequentially if the midi message meets the parameters of all of the active conditional verifiers (byte checks). The transforms can manipulate any byte of a midi message including the statusbyte. Each of the four transforms have the following parameters:

transform byte number (X byte number)

Selects which byte in the midi message to perform the transform on. Possible values are between 0 and 16, with 0 referencing the message's statusbyte and 1 –16 referencing data bytes.

transform operand

Selects what type of transform to perform on the selected byte. The possible selections are: no xform, add, subtract, multiply, divide, set to, move to, xtend msg to, crop msg after, and do not transmit. The **do not transmit** transform allows the filtration of certain midi messages in the stream (note that this operator does not affect an individual byte of a message, but the entire message itself – therefore the other parameters (transform value and transform byte number), as well as any other of the four transforms for the current operation, are unused when this operand is selected).

transform value

Selects the value for the transform. Refer to the table below for the relationships between the transform operand and transform value. Possible values for this parameter are between 0 and 255. Note that if the **transform byte number** parameter is set to zero (or "statusbyte") for the particular transform, values of 128 and above will be translated into text labels such as "note off [ch. 3]" instead of their numerical equivalent, assuming that the **translate statusbytes** option in the display settings menu has been enabled.

operand	explanation of operand and transform value
no xform	no action is performed on any of the data, so transform value is irrelevant.
add / subtract multiply / divide	the transform value is added to or subtracted from the selected midi message byte. the selected midi message byte is multiplied or divided by the transform value.
set to	the selected midi message byte is set to the transform value.
move to	the selected midi message byte is copied to a new midi message byte number, specified by the transform value.
xtend msg to (extend message to)	a data byte is added to lengthen the number of bytes of a midi message being processed, where the transform value determines the midi message byte position of the generated byte. The generated byte value is set to 0, and any undefined bytes between the end of the unprocessed midi message and the generated byte position of the new (processed) message are automatically filled with a value of 0. The number of bytes that defines the message type is also increased, so further transforms can be performed on the new bytes. The **transform byte number** is unused directly; however the parameter must be set to a valid byte number for the unprocessed message – typically use statusbyte (0).
crop msg after	shortens the byte length of the entire processed midi message to the data byte number defined with the transform value. The **transform byte number** is unused; however the parameter must be set to a valid byte number for the unprocessed message as above.
do not xmit (do not transmit)	filters (does not pass to midi out) the entire midi message, processed or unprocessed – see the above description of the **transform operand** parameter for further explanation. This operand does not use a **transform value.** Set **transform byte number** to statusbyte.
reserved	reserved for future use; currently does nothing.

gameport input

Gameport to midi conversion allows for a cheap and easy source of additional realtime controllers, and the simplicity of the interface makes it easy to design and build customized control surfaces (for instance, controllers for mixing or a machine control interface with data entry buttons or sliders, etc). The nature of the analog gameport is such that not just conventional controls such as knobs can be easily interfaced to it, but also such things as photoresistors (which could act as a light-oriented expressoin controller) and other variably-resistant devices. The standard analog PC gameport is designed to support two joysticks, both with 2 axis (channels which can be used for variable controllers) and two buttons (which can be used for on/off type controllers), for a total of 4 variable and 4 I/O controller devices. Sounds could be triggered from any of the gameport channels by programming them to transmit note-on messages with the conversion byte in the 2nd databyte of the message; according to the midi specification a note on message with a velocity of zero is the same as a note off message, therefore this application would work.

Due to the (somewhat flawed) way in which the standard IBM-PC joystick interface was designed (back in the very early 1980's), there can be substantial differences in the circuitry between various interfaces. For this reason it is important to perform the calibration functions for each axis channel of your particular setup (joystick/controller and interface port). More information on this procedure is found in the 'Gameport Input Settings' page.

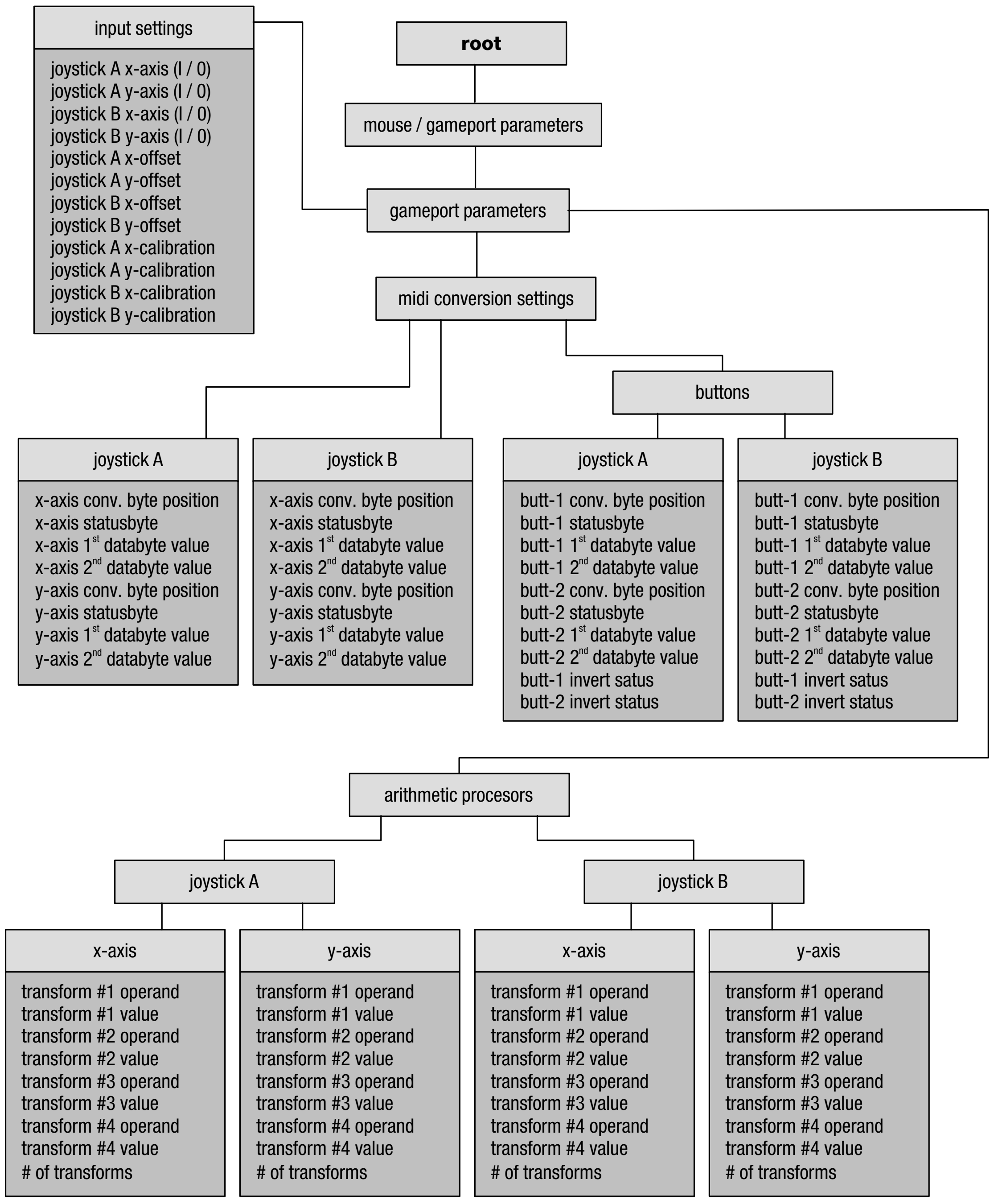
Note that if an audio buffer size less than 72 bytes is used when converter is run in audio mode, gameport input is disabled until converter is quit and re-run with a larger audio buffer size (selectable in the file hardware.cfg – see the section detailing this file in the beginning of this manual). This is due to the fact that the smaller audio buffer sizes generate so many audio interrupts that converter is unable to complete the timing-based gameport input routines properly in order to acquire an accurate reading.

Programming gameport to midi conversion is simple – here's how to program joystick A to generate a stream of pitch bend and modulation wheel data on midi channel 1:

- Press [F4] for the game/mouse parameters menu, then [F5] for gameport settings, followed by [F5] again for gameport input settings. Check to make sure that the **joystick A x-axis** and **joystick A y-axis** parameters are set to "ON".
- Press [F6] for the gameport midi conversion settings menu, followed by [F5] for joystick A's settings page.
- For the **x-ax conv. byte pos.** parameter (x-axis conversion byte position) type 2 to place the converted byte value in the 2nd databyte position; for the **x-axis statusbyte** parameter type 224 (for pitch bend channel 1); followed by 0 for the **x-ax 1st databyte val** parameter (x-axis 1st databyte value) and perhaps for clarity's sake place a 0 in the **x-ax 2nd databyte val** parameter (which is going to be filled in by the byte taken from the x-axis channel of joystick A).
- Next to program modulation wheel data, move the cursor to the **y-ax conv. byte pos** parameter and type 2, then type 176 into the **y-axis statusbyte** parameter to specifiy a controller message on channel 1, type 1 into the **y-ax 1st databyte val** parameter to specify modulation as the controller message type, and optionally place a 0 in the **y-ax 2nd databyte val** parameter.

And that's it – the gameport is programmed to transmit pitch bend and modulation data on midi channel 1. If it doesn't appear to be working, check the root realtime settings menu to ensure that the gameport input is activated. If it says "gameport disabled" you will have to exit converter and choose an audio input buffer size of 72 bytes or larger. Incidentally, due to the somewhat inaccurate nature of gameports, pitch control may not be the most musically appropriate use for them – refer to the midi appendix for a list of possible controllers the gameports could be assigned to.

The following is a structural map of the menu hierarchy for the gameport input menu section (F4 – gameport settings), which may help familiarize the location of the various parameters.



gameport input settings

game/mouse parameters : gameport parameters : input settings

[F4 – F5 – F5]

joystick axis

Enables or disables input from the specific axis (X or Y) of the specific joystick (A or B). By individually enabling only the channels needed, and disabling unused channels, overall system speed increases.

joystick offset

Allows for an offset adjustment of the input value from the specified axis of the specific joystick. This value is added to the input value. Possible values are between –127 and 127 – typically this can be left at a setting of zero.

calibrate joy axis

As there is some electrical variance between different joystick interfaces, these functions permit machine-specific timing calibration for the joystick port. The acquired calibration settings are stored in converter's main system configuration file `c.cfg`, so that once properly calibrated, these functions will never have be performed again unless a different gameport device is used. Notice that the 'calibrate joy axis' selections are not parameters themselves; the calibration functions are accessed by pressing [enter] when one of the four channel's calibration routines is selected with the cursor.

By entering into one of the calibration functions, the normal operation of converter is temporarily halted (converter enters standby mode) until calibration is completed. There are three steps for calibrating each axis channel of the gameport: values must be acquired for both the minimum, center, and maximum positions for the axis channel. Detailed on-screen explanations will guide you through all three easy calibration steps.

Once the calibration steps are completed, test the range of your joystick or gameport device to see if the calibration settings are satisfactory. If not, simply repeat the calibration process for that axis or channel until the acquired calibration settings are as ideal as possible.

axis conversion byte position

Selects which databyte of the chosen midi message the converted value is placed in. For example, to convert x-axis joystick input to control the pitch wheel, the generated message would consist of one statusbyte and 2 databytes as follows: the statusbyte (indicating a pitch wheel message on midi channel X), followed by the first databyte representing the LSB component of the pitch wheel message (unused in this case since we can only provide an 8-bit value, therefore should be set to a value of zero), followed by the second databyte representing the MSB value for pitch wheel. Hence you would use a value of 2 (to indicate that the value taken from the gameport should be placed in the 2nd databyte of the midi message). Since the statusbyte is not considered a data byte, this parameter should be read in terms of databyte number, not byte number in the cumulative message packet (i.e. 2 instead of 3). Possible byte position values are either 1or 2; if set to zero, no databytes are transmitted for the particular midi message.

Note that not all midi messages use both, or any, data bytes; channel aftertouch, program change, and song select only use the first databyte (for aftertouch amount, program change number and song number respectively), while system realtime messages such as midi clock simply use the statusbyte. As converter automatically enforces these midi definitions, setting this byte position parameter to 2 when the message type only supports one data byte results in the converted value not being a part of the generated midi message.

axis statusbyte

Determines the midi message type (and channel) to use for the joystick conversion value. Possible values are between 128 and 255.

axis 1st databyte value

Sets a fixed value for the first databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 1st databyte position.

axis 2nd databyte value

Sets a fixed value for the second databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 2nd databyte position.

gameport button midi conversion settings

g/m parameters : gameport parameters : midi conversion settings : buttons : joystick A/B

[F4 – F5 – F6 – F7 – F5/F6]

button conversion byte position
Selects which databyte of the chosen midi message the converted value is placed in. For more information, refer to the midi conversion settings documentation for the joystick axis inputs.

button statusbyte
Determines the midi message type (and channel) to use for the joystick conversion value. Possible values are between 128 and 255.

button 1st databyte value
Sets a fixed value for the first databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 1st databyte position.

button 2nd databyte value
Sets a fixed value for the second databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 2nd databyte position.

button invert status
Enables or disables inversion of the button's input value. This allows the button to generate its direct on / off status as is, or to provide an off status when physically in the on position and an on status when physically in the off position.

g/m parameters : gameport parameters : arithmetic processors : joystick A/B – Xaxis/Yaxis

[F4 – F5 – F7 – F5/F6 – F5/F6]

Note that the arithmetic operations are performed sequentially in the order they appear in the transform list, from first to last – ‘order of operations’ is not followed. This allows for the input data to be processed in a more flexible manner.

transform operand (X-form)
Selects the type of data transform to be performed on the value converted from the respective joystick axis input. Possibilities are: no xform, add, subtract, multiply, and divide.

transform value
Selects the numeric value for the transform. For instance, if the chosen transform is divide, and the transform value is 3, all converted values will be divided by three. Possible values are between 0 and 127 (protection is implemented against division by zero).

# of transforms
Determines how many transforms are active, counting from the first transform. This makes it easy to bypass certain transforms and only activate the first 1, or 2, or 3 transforms. If set to zero, none of the transforms will be performed on the converted data.

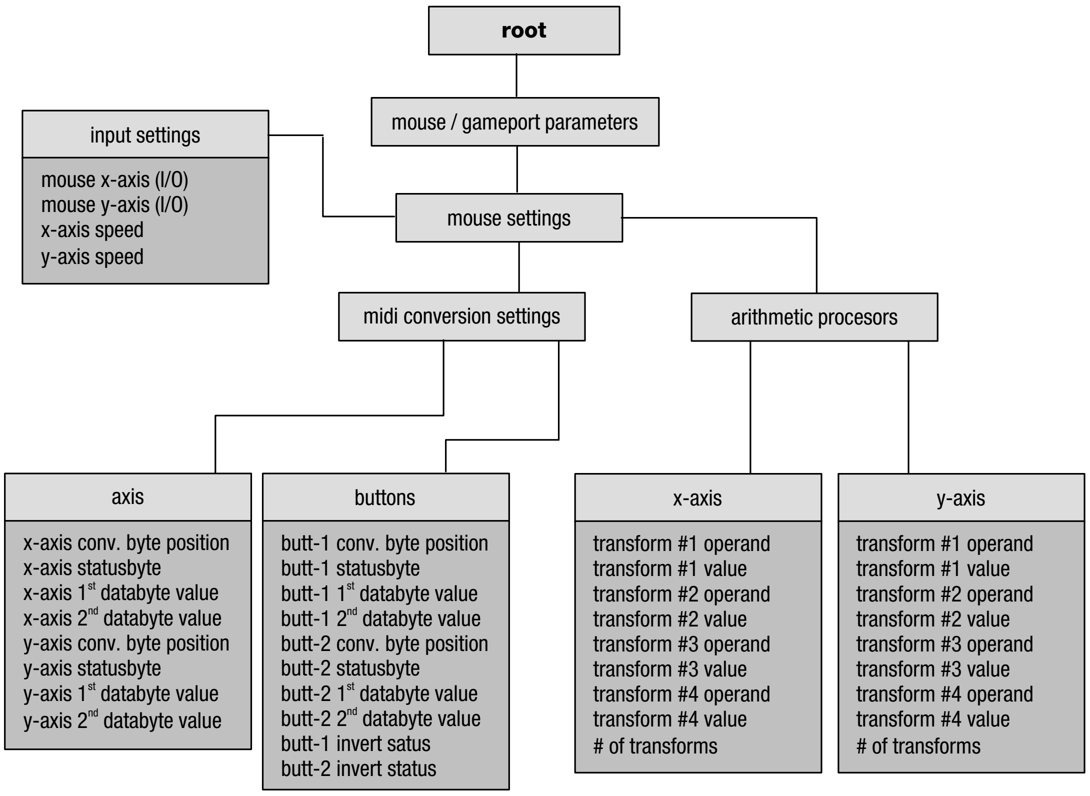
mouse input

Mouse input conversion is programmed in the same way as the gameport input, so additional explanation would be redundant. Mouse to midi conversion was implemented to allow the use of certain special control surfaces, such as a touchpad, to be used as a controller source – in much the same way as a ribbon controller on some synthesizers. Since a touchpad device has 2 axis (X and Y), it can be more expressive than an ordinary one-dimensional ribbon controller, as two different controller types can be generated from the one pad.

If the mouse tracking speed is too slow or too fast, try adjusting the x-axis speed or y-axis speed parameters in the mouse input settings menu.

Note that any DOS-mouse-interface compatible device can be used, connected to either a serial or PS/2 mouse port (assuming the DOS mouse driver supports both the port and the device). A serial-interface Cirque Glidepoint Trackpad was used for development. This device was detected and supported by the Logitech DOS mouse driver (version 7.2) on the development system with no problems.

The following is a structural map of the menu hierarchy for the mouse input menu section (F4 – mouse settings), which may help familiarize the location of the various parameters.



mouse input settings

game/mouse parameters : mouse parameters : input settings

[F4 – F5 – F5]

mouse x-axis
mouse y-axis
Enables or disables input from the specific axis (X or Y) of the mouse. This is useful if only one axis is needed for a particular application.

x-axis speed
y-axis speed
Sets the tracking speed for the individual mouse input axis, allowing for small movements to cover a wide range of values, or large movements to cover a small range of values for finer control. Possible values are between 0 (slow tracking) and 127 (fast tracking).

mouse axis midi conversion settings

game/mouse parameters : mouse parameters : midi conversion settings : joystick A/B

[F4 – F5 – F6 – F5/F6]

axis conversion byte position

Selects which databyte of the chosen midi message the converted value is placed in. For example, to convert x-axis mouse input to control the pitch wheel, the generated message would consist of one statusbyte and 2 databytes as follows: the statusbyte (indicating a pitch wheel message on midi channel X), followed by the first databyte representing the LSB component of the pitch wheel message (unused in this case since we can only provide an 8-bit value, therefore should be set to a value of zero), followed by the second databyte representing the MSB value for pitch wheel. Hence you would use a value of 2 (to indicate that the value taken from the mouse should be placed in the 2nd databyte of the midi message). Since the statusbyte is not considered a data byte, this parameter should be read in terms of databyte number, not byte number in the cumulative message packet (i.e. 2 instead of 3). Possible byte position values are either 1or 2; if set to zero, no databytes are transmitted for the particular midi message.

Note that not all midi messages use both, or any, data bytes; channel aftertouch, program change, and song select only use the first databyte (for aftertouch amount, program change number and song number respectively), while system realtime messages such as midi clock simply use the statusbyte. As converter automatically enforces these midi definitions, setting this byte position parameter to 2 when the message type only supports one data byte results in the converted value not being a part of the generated midi message.

axis statusbyte

Determines the midi message type (and channel) to use for the mouse conversion value. Possible values are between 128 and 255. Note that certain midi message types are unallowed, simply because they are either undefined (such as 253 or 245), or are complex multi-byte words such as 240 (system exclusive) or 241 (midi time code quarter frame).

axis 1st databyte value

Sets a fixed value for the first databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 1st databyte position.

axis 2nd databyte value

Sets a fixed value for the second databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 2nd databyte position.

mouse button midi conversion settings

game/mouse parameters : mouse parameters : midi conversion settings : buttons

[F4 – F5 – F6 – F7 – F5/F6]

button conversion byte position

Selects which databyte of the chosen midi message the converted value is placed in. For more information, refer to the midi conversion settings documentation for the mouse axis inputs.

button statusbyte

Determines the midi message type (and channel) to use for the mouse button conversion value. Possible values are between 128 and 255.

button 1st databyte value

Sets a fixed value for the first databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 1st databyte position.

button 2nd databyte value

Sets a fixed value for the second databyte of a midi message. The **conversion byte position** parameter overrides this value if it is set to place the conversion result in the 2nd databyte position.

button invert status

Enables or disables inversion of the button's input value. This allows the button to generate its direct on / off status as is, or to provide an off status when physically in the on position and an on status when physically in the off position.

Note that the arithmetic operations are performed sequentially in the order they appear in the transform list, from first to last – ‘order of operations’ is not followed. This allows for the input data to be processed in a more flexible manner.

transform operand (X-form)

Selects the type of data transform to be performed on the value converted from the respective mouse axis input. Possibilities are: no xform, add, subtract, multiply, and divide.

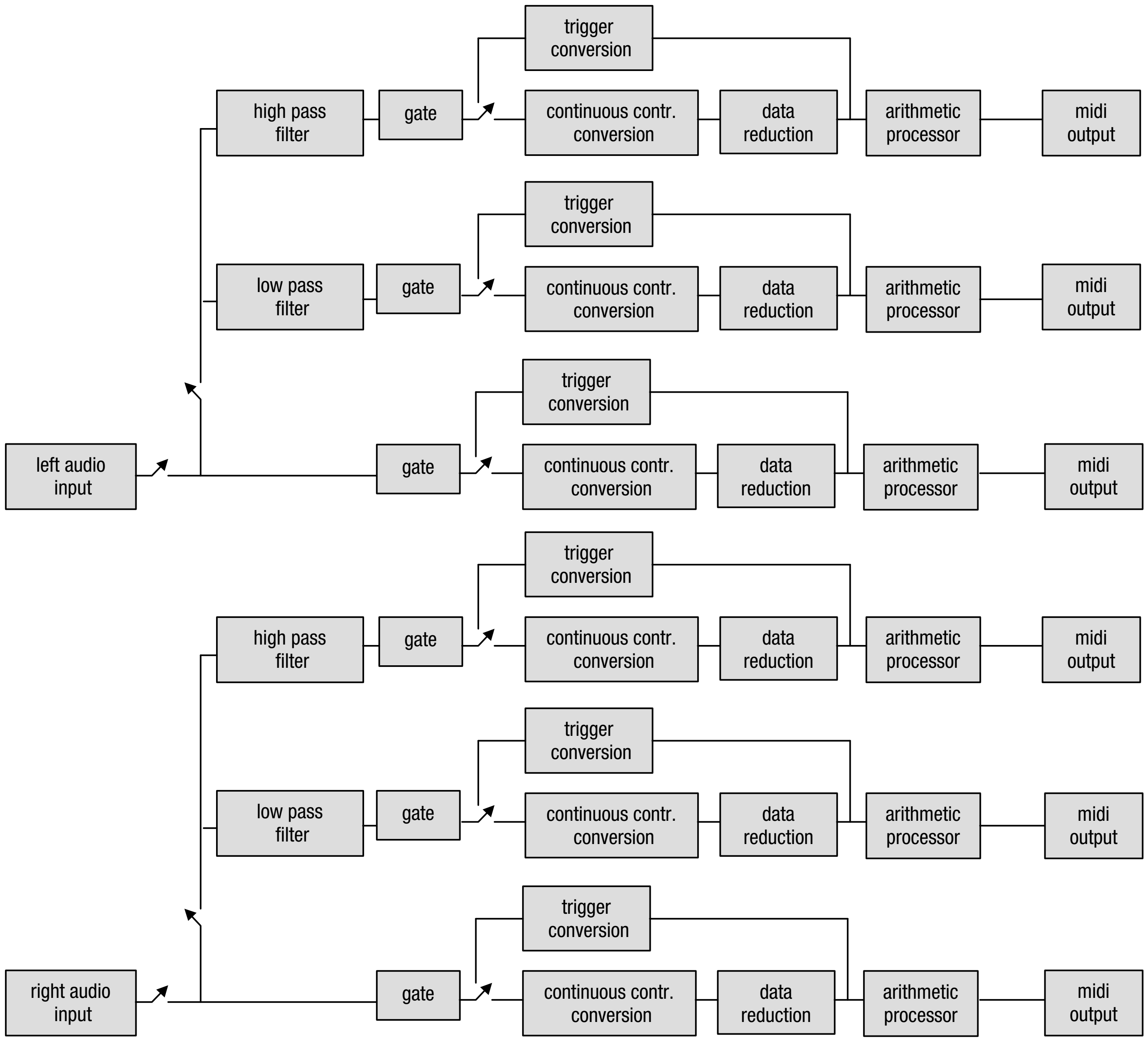
transform value

Selects the numeric value for the transform. For instance, if the chosen transform is divide, and the transform value is 3, all converted values will be divided by three. Possible values are between 0 and 127 (protection is implemented against division by zero).

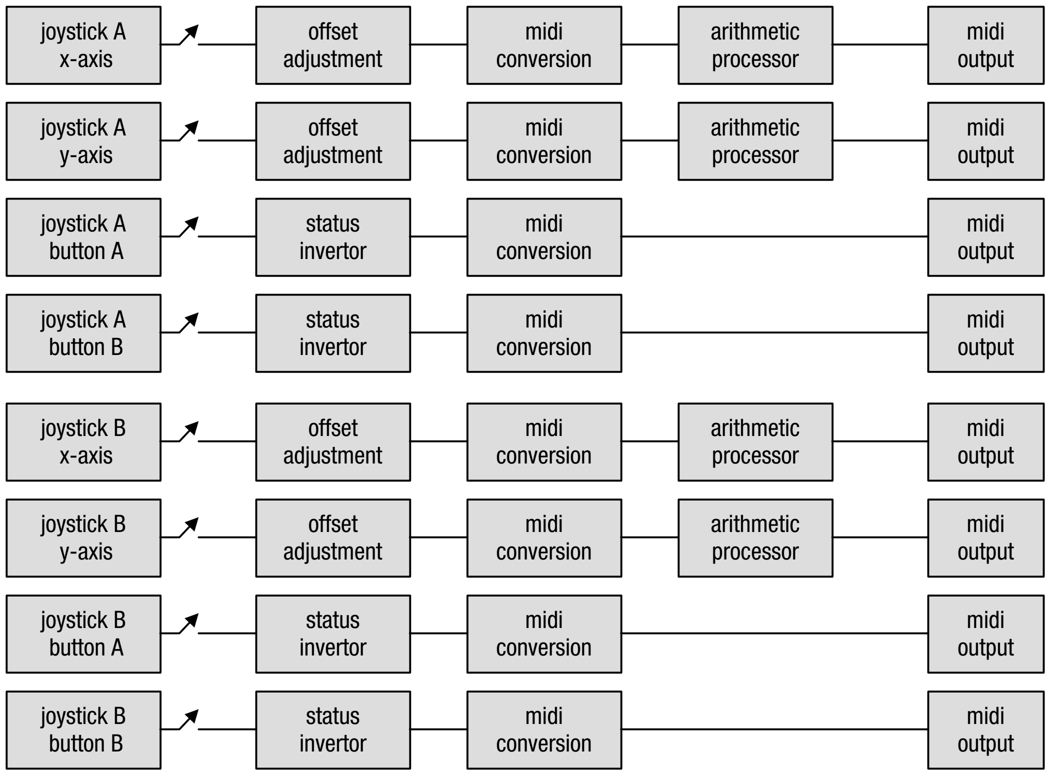
# of transforms

Determines how many transforms are active, counting from the first transform. This makes it easy to bypass certain transforms and only activate the first 1, or 2, or 3 transforms. If set to zero, none of the transforms will be performed on the converted data.

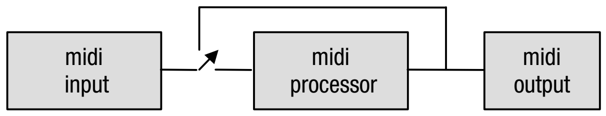
audio input signal paths



joystick input signal paths



midi input signal path



Midi is a simple asynchronous serial interface able to transmit data at a rate of 31.25Kbps, which allows a maximum transfer rate of about 3000 bytes per second. Midi data is formatted into data ‘packets’ called messages. Each message begins with a status byte, and may be followed by any number of additional data bytes as determined by the message type. Some messages only consist of a statusbyte (notably the system realtime messages such as the midi timing clock), while others are followed by one, two, or more data bytes. For most applications using converter, only channel-oriented midi messages (called voice messages) will be of use (so don’t worry about the variable multi-byte system exclusive messages, time code quarter frames, and the like).

Midi data is always transmitted sequentially from statusbyte to last databyte. To reduce the transmission of redundant data, a type of run-length-encoding called running status exists, where messages of the same status byte are sent without re-sending the status byte. For instance, if three notes were pressed on a keyboard transmitting on midi channel one, there would be one status byte for a note on message on channel 1 (status byte 144), followed by the first note number and note velocity, then the second note number and velocity, and then the third note number and velocity. It is assumed that any data following that original status byte is the data bytes for an additional note on, and this condition remains until a new status byte is received which cancels this running status.

For a much more in-depth explanation of midi, do some research on the web, or visit:
`http://www.harmony-central.com/MIDI/Doc/doc.html`

voice messages (channel specific)

message type	status byte	data byte 1	data byte 2
note off	128 – 143	note number	note velocity
note on	144 – 159	note number	note velocity
polyphonic aftertouch	160 – 175	note number	aftertouch amount
controller / mode change*	176 – 191	controller number*	controller value*
program change	192 – 207	program number	not used
channel aftertouch	208 – 223	aftertouch amount	not used
pitch wheel control	224 – 239	pitch wheel LSB	pitch wheel MSB

* see the table of controller (or ‘continuous controller’) definitions on the next page.

realtime messages (not channel-specific)

message type	status byte	data following statusbyte
system exclusive	240	machine – specific variable – length bytestream
midi time code ¼ frame	241	1 data byte, many bytes comb. for time position
song position pointer	242	byte #1: LSB byte #2: MSB
song select	243	1 data byte: song number
tune request	246	none
end of system exclusive (EOX)	247	none
timing clock	248	none
midi tick	249	none
start	250	none
continue	251	none
stop	252	none
active sensing	254	none
system reset	255	none

controllers / mode changes (MSB)

controller	controller number	data byte 2
bank select	0	value (MSB 0-127)
modulation wheel	1	value (MSB 0-127)
breath control	2	value (MSB 0-127)
foot controller	4	value (MSB 0-127)
portamento time	5	value (MSB 0-127)
data entry	6	value (MSB 0-127)
volume	7	value (MSB 0-127)
balance	8	value (MSB 0-127)
pan position	10	value (MSB 0-127)
expression	11	value (MSB 0-127)
effect control 1	12	value (MSB 0-127)
effect control 2	13	value (MSB 0-127)
general purpose controller 1	16	value (MSB 0-127)
general purpose controller 2	17	value (MSB 0-127)
general purpose controller 3	18	value (MSB 0-127)
general purpose controller 4	19	value (MSB 0-127)
damper pedal (on/off)	64	0 – 63 = off, 64-127 = on
portamento (on/off)	65	0 – 63 = off, 64-127 = on
sustenuto (on/off)	66	0 – 63 = off, 64-127 = on
soft pedal (on/off)	67	0 – 63 = off, 64-127 = on
legato footswitch (on/off)	68	0 – 63 = off, 64-127 = on
hold 2 (on/off)	69	0 – 63 = off, 64-127 = on
sound controller 1 (sound variation)	70	value (0-127)
sound controller 2 (timbre)	71	value (0-127)
sound controller 3 (release time)	72	value (0-127)
sound controller 4 (attack time)	73	value (0-127)
sound controller 5 (brightness)	74	value (0-127)
sound controller 6	75	value (0-127)
sound controller 7	76	value (0-127)
sound controller 8	77	value (0-127)
sound controller 9	78	value (0-127)
sound controller 10	79	value (0-127)
general purpose controller 5	80	value (0-127)
general purpose controller 6	81	value (0-127)
general purpose controller 7	82	value (0-127)
general purpose controller 8	83	value (0-127)
portamento control	84	source note
effects 1 depth	91	value (0-127)
effects 2 depth (tremulo)	92	value (0-127)
effects 3 depth (chorus)	93	value (0-127)
effects 4 depth (celeste/detune)	94	value (0-127)
effects 5 depth (phaser)	95	value (0-127)
data entry +1	96	none
data entry –1	97	none
non-registered parameter number LSB	98	value (LSB 0-127)
non-registered parameter number MSB	99	value (MSB0-127)
registered parameter number LSB	100	value (LSB 0-127)
registered parameter number MSB	101	value (MSB 0-127)

Note that the above list represents **defined controllers** according to the Midi Specification. Some of the gaps in the numbering represent numbers for which controllers have not yet been defined (hence in converter these values are referred to as **undefined controllers**) – they may be used, however most (or possibly all) midi gear will not respond to undefined midi controller messages.

high-resolution component (LSB) for controllers (unused by converter)

controller	controller number	data byte 2
bank select	32	fine value (LSB 0-127)
modulation wheel	33	fine value (LSB 0-127)
breath control	34	fine value (LSB 0-127)
foot controller	36	fine value (LSB 0-127)
portamento time	37	fine value (LSB 0-127)
data entry	38	fine value (LSB 0-127)
channel volume	39	fine value (LSB 0-127)
balance	40	fine value (LSB 0-127)
pan	42	fine value (LSB 0-127)
expression controller	43	fine value (LSB 0-127)
effect control 1	44	fine value (LSB 0-127)
effect control 2	45	fine value (LSB 0-127)
general purpose controller #1	48	fine value (LSB 0-127)
general purpose controller #2	49	fine value (LSB 0-127)
general purpose controller #3	50	fine value (LSB 0-127)
general purpose controller #4	51	fine value (LSB 0-127)

special controllers

controller	controller number	data byte 2
all sound off	120	0
reset all controllers	121	0
local control (on/off)	122	0 = off, 127 = on
all notes off	123	0
omni mode off & all notes off	124	0
omni mode on & all notes off	125	0
poly mode (on/off) all notes off	126	number of channels
poly mode on & all notes off	127	0

midi note numbers

octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

preset files

Several 'preset' settings files (.set) accompany the converter distribution. These presets are pre-programmed for some of the typical applications of converter, and serve as templates which can be edited or tweaked to accommodate the specifics needed for a particular setup.

Note that the file `default.set` is the work file converter uses to automatically 'remember' its settings between sessions. This means that converter can be quit, and then re-run, and all the settings will be restored from the last session – kind of like a hardware battery backup.

file	description/routing	
audiojoy	audio input from the main (non-filter) channels, gameport input from joystick A	
	audio left channel:	channel volume continuous controller, midi channel 2
	audio right channel:	channel aftertouch, midi channel 2
	joystick A X-axis:	pitch wheel, midi channel 1
	joystick A Y-axis:	modulation wheel, midi channel 1
	joystick A button 1:	damper pedal, midi channel 1
	joystick A button 2:	sustenuto pedal, midi channel 1
	joystick B X-axis:	expression controller, midi channel 1
	joystick B Y-axis:	foot controller, midi channel 1
	joystick B button 1:	portamento pedal, midi channel 1
	joystick B button 2:	legato pedal, midi channel 1
2chancc	continuous controller audio conversion from the main (non-filter) channels	
	audio left channel:	modulation wheel continuous controller, midi channel 1
	audio right channel:	pitch wheel continuous controller, midi channel 2
6chancc	continuous controller audio conversion from all audio channels (main & filters)	
	audio left channel:	channel volume, midi channel 1
	audio right channel:	modulation wheel, midi channel 2
	audio left LP channel:	expression controller, midi channel 4
	audio right LP channel:	fx control 1 continuous controller, midi channel 5
	audio left HP channel:	foot controller, midi channel 7
	audio right HP channel:	channel aftertouch, midi channel 8
trigmain	note on/off with velocity trigger audio conversion from the main (non-filter) channels (programmed for general midi percussion map on midi channel 10)	
	audio left channel:	note #36 (general midi bass drum), midi channel 10
	audio right channel:	note #42 (general midi snare drum), midi channel 10
	note: use the note message display panel (F9) to graphically see triggered note messages.	
empty	a blank template with an empty midi processor and all inputs set to default settings.	

As a general troubleshooting tip, if converter is having trouble functioning properly, or it is difficult to isolate what is interfering with its successful operation, try making a bootable floppy disk with converter installed on it – this may help determine if the problem is a hardware or software issue.

My soundcard isn't found by converter! or **converter hangs on loading!**

Make sure that the appropriate DOS environment variable for your soundcard is included in your `autoexec.bat` file – converter references the hardware information included in this string of text in order to locate your soundcard. See the next page for specific information. The best approach is to use the installation software that came with your soundcard (or that is available on your soundcard manufacturer's website) to properly install and configure your soundcard on your PC.

If the environment variable is in your `autoexec.bat` file, it appears to be correct, but you are using a soundcard that implements emulation of a Soundblaster 16, it may be that your soundcard isn't 100% compatible with the Soundblaster 16 standard; an example of this is the Soundblaster Live! which only implements Soundblaster Pro digital audio output under DOS – no midi or audio input is available, which converter requires (obviously). The only solution is to use a different soundcard.

Another possibility is that there is not enough free memory available under DOS for converter - if there are a lot of memory-resident programs loaded at bootup (ie. from the `autoexec.bat` file), these may be taking up memory converter requires to function properly. If this is the case, try making a bootable floppy with converter and its own autoexec.bat file – see the section in this manual which describes this procedure for further information.

Finally, check to make sure that the settings in the `hardware.cfg` file in converter's folder are correct and valid, specifically the buffer size setting.

When I exit converter, I no longer can use my computer's keyboard at the DOS prompt!

The DOS keyboard interrupt handler is replaced within converter so as to provide better and more extensive keyboard support. When converter is quit, the original keyboard handler is 're-connected' to the keyboard. However, on certain motherboards, it appears this does not happen properly, which may be a quirk of the keyboard controller on that particular motherboard. Note that this in no way affects normal converter operation. This issue may or may not be fixed in a later release (the code in converter seems to be completely correct). For now, just reboot.

My Soundblaster card is found and initialized, but audio input stops a few buffer frames after a signal is fed into the card!

This was an issue on the development machine, where the AWE64 was configured to use DMA channel 0 as its 8-bit DMA channel. Channel 0 is not typically recommended for use by peripherals, as historically this channel has been used for motherboard memory refresh cycles, and while that may no longer be the case, and the Soundblaster configuration utility may claim that the channel is free to be used, it may in fact be used by another device on the motherboard. In short, the DMA controller is re-programmed by that other device, taking control from converter and the Soundblaster card, and stopping audio data input. During development, it was possible to 'force' this situation to work by re-programming the DMA controller chip each time an interrupt occurred from a complete buffer transfer, but this created significantly decreased system performance and also resulted in lost samples. The solution is to configure the card to use another DMA channel that is actually free – try channel 1 or 3.

My Roland MPU-401 (or compatible) isn't found by converter; instead the computer hangs after the software says "Initializing MPU-401…"!

This means you have specified the incorrect base address for the card in the hardware.cfg file. Check your card's jumper settings and/or try another address. If the base address setting in hardware.cfg is indeed correct, you may have another device installed in the computer which is using the same address as your MPU-401 – try to identify what that piece of hardware would be, or change the base address of your MPU-401 card to a different setting such as 300.

My Roland MPU-401 (or compatible) is detected and initialized by converter, however no midi data is received by the software!

This means you have specified the incorrect interrupt (IRQ) for the card in the hardware.cfg file. If the IRQ setting in hardware.cfg is known to be correct, another hardware device installed in your computer may be either using the same IRQ number. Try configuring your MPU-401 to use a different IRQ number, or locate the conflicting piece of hardware and re-configure it.

Why won't converter run in a DOS shell under Windows?

During testing, it was found that converter didn't function predictably when run in a DOS shell (DOS prompt) under Windows9x. This is likely due to the fact that many of the computer's built in devices are controlled directly from converter itself, which may be interfering with Window's controlling nature in regards to hardware. Solution? Boot your machine into 'pure' DOS mode in one of several ways:

- Re-boot your computer, press F8 as Windows9x begins to boot to bring up a 'boot menu', and select 'command prompt only'
- create a bootable floppy and run converter from it (easy approach)
- reboot your machine into pure DOS mode by clicking on the start menu and selecting shutdown, and 'Reboot into MS-DOS'. Note that this approach may not automatically run your `autoexec.bat` file, which may be needed if you are using a Soundblaster or Gravis Ultrasound card. If so, you can run the file yourself from the dos prompt by simply moving to your hard disk's root directory (ie. `c:\`) and typing `autoexec` followed by return.

Running converter in a DOS shell from within Windows9x would in effect reduce the benefits of it being designed for DOS in the first place, as the additional weight of Windows running in the background would significantly reduce converter's realtime performance. For this reason, making it function properly from within Windows is not a priority.

When I try to run converter under Windows Millenium Edition, WindowsXP, or various flavours of Windows NT (including Windows2000), it doesn't work.

Although converter hasn't been tested under these platforms, it is automatically assumed that it will not function at all, due to the fact that these versions of Windows use a DOS emulator which does not permit any program to communicate directly with the computer's hardware. Since converter is almost entirely based on direct hardware communication specifically for performance benefits, it obviously will not get along with any of these Windows platforms. However, all is not lost – simply create a bootable floppy disk on a computer with Windows98 or earlier, copy the converter files onto the disk, and configure it for your computer's soundcard. By using a bootable floppy disk you don't need to worry about incompatibilities with your version of Windows since DOS is loaded right from the floppy. For more information see the appendix on creating a bootable DOS floppy disk.

When I try to run converter, it crashes, and I know my soundcard isn't the problem!

Make sure you have enough free memory available in the base 640k that DOS uses – you can check by typing 'mem' at the DOS prompt. Memory above the base 640k is not directly available to converter, so even if a computer has been expanded to 128MB of RAM, converter doesn't make use of any of it (ordinarily it hasn't any need to). Since converter (and many other DOS-based applications) requires at least 500k of free ram , it's a good idea not to have a huge number of memory-resident software running at the same time if they are taking up a significant amount of base memory, such as DOS cd-rom drivers, EMM386, etc. A good workaround is to make a bootable floppy disk from which to run converter.

This section is provided for users configuring an older soundcard within DOS, for which there may be no accompanying documation or installation software. **If you are experienced with DOS and setting environment variable hardware settings, or you have all the necessary software and documentation to properly configure the card under DOS, or your soundcard already works under DOS with other software (and especially if it works fine with converter), you will likely have no need to read this section**. If you have difficulties getting converter to locate your soundcard, this reference might be able to give you some pointers in the right direction. Note that the details of complete hardware installation for soundcards is far beyond the scope of what can be covered here; specific hardware installation information for soundcards may be placed on the website if there is such a need.

In order for converter to make use of your soundcard, it must be able to find out its particular hardware settings which tend to differ from computer to computer based on the other peripherals one has installed in the machine. Both the Gravis Ultrasound and Soundblaster series make use of what's called a 'DOS environment variable', or in other words a line of text placed in the **autoexec.bat** file (editable using Windows' notepad or DOS edit) which lists the specific information a software application needs in order to 'connect' with the interface. Both the Gravis Ultrasound and Soundblaster cards were packaged with DOS-based configuration or installation utilities which allowed the setting (and testing) of correct hardware DMA, IRQ, and port address settings for the card. However, due to the fact that the soundcards have been discontinued, and the DOS-specific software for these 'obsolete' soundcards may not remain available on the manufacturer's website indefinitely, the following information is provided to assist in the successful configuration of these cards.

Note that this guide does not necessarily replace the need for the appropriate installation software and documentation for your particular card.

Note for users running with Windows9x: Even if you have the appropriate drivers installed for the card under Windows, you might want to verify that an environment string has been placed in the autoexec.bat file for DOS usage. Often it seems that the Windows-based installation software for soundcards doesn't automatically perform this task.

Gravis Ultrasound

The Gravis Ultrasound uses a jumper on the card itself to select the base address for the card (either 210, 220, 230, 240 etc), while the rest of the parameters (IRQ, DMA) are configured by software.

Environment variable: `SET ULTRASND=240,1,5,7,5`
The numbers after the = sign in the line represent:

```
    base address, 8-bit DMA, 16-bit DMA, Audio (GF1) IRQ, Midi port IRQ.
```

Typically, the DOS software installation for this soundcard will automatically update the autoexec.bat file with the correct parameters. However, a caveat:

- if a Windows9x (specific) driver was installed as well, the ULTRASND string might instead look like this: `SET ULTRASND=XXX,X,X,X,X` (literally). To function under DOS, this line should be 'disabled' by typing the letters 'rem' at the beginning of the line (before the word 'SET'), and a new line with actual numerical information should be typed in to look something like this:

```
rem SET ULTRASND=XXX,X,X,X,X
SET ULTRASND=240,1,5,7,5
```

Note that the numbers above are for a specific configuration – they aren't 'one size fits all'. You must use the numbers specific to your system's configuration. **A bootable floppy could be created to run converter with its own autoexec.bat, thus avoiding this issue of modifying the autoexec.bat on your hard drive (see the next appendix).**

Assuming you have installed a Windows driver for the card (since you are reading this paragraph), you can find the specific numbers from the Windows driver by going to your control panel, selecting system, choosing the 'device manager' tab, expanding the 'sound, video and game controllers' droplist, and double-clicking the Gravis Ultrasound item. This should give you a window with several tabbed entries – click on 'resources'. Under the 'resource settings' listbox, read the following numbers and write them down on paper to be used for your environment string:

- the first number of the first entry (input/output range – this is your card's base address)
- the fourth and fifth entries (interrupt requests - these are the cards audio and midi IRQs)
- the fifth and sixth entries (direct memory access – these are the card's 8 and 16-bit DMA channels)

Place these numbers in their respective places on the DOS environment string (line of text) in your autoexec.bat file. The IRQ numbers can be interchanged, as well as the numbers for the dma channels – if they don't work for some reason, try switching them around.

Soundblaster 16, Soundblaster 32, AWE32, AWE64, and 100% compatible

The Soundblaster series of soundcards have a variety of installation requirements – some are 'plug and play', others are not. The specifics of each soundcard's hardware configuration is beyond the scope of what can be covered here; however, often the card will be configured appropriately under Windows (if used) and the numbers for the environment variable can be sourced from it.

Environment variable: `SET BLASTER=A220 I10 D0 H7 P300 E620 T6`
The important numbers after the = sign in the line represent:

```
    base address, IRQ, 8-bit DMA, 16-bit DMA, Midi port address
```

The E and T parameters are not important in this case, and can be left alone.

Typically, the DOS software installation for this soundcard (either ctcm.exe or diagnose.exe or the like) will automatically update the autoexec.bat file with the correct parameters. However, if this software isn't available, and the card has been configured sucessfully in Windows9x, the relevant information can be retrieved from the Windows driver.

Open the Windows9x control panel, double-click on 'system', choose the 'device manager' tab, expand the 'sound, video and game controllers' droplist, and double-click the appropriate Creative driver for *audio* (in our case, 'Creative AWE64 16-bit Audio (SB16 compatible)'). This should give you a window with several tabbed entries – click on 'resources'. Under the 'resource settings' listbox, write the following numbers down on paper to be used for your environment string:

- the first entry (interrupt request – this is your card's IRQ)
- the second and third entries (direct memory access – the one with a value of 3 or below is the Soundblaster's 8-bit DMA channel, and the one with a value above 4 is the Soundblaster's 16-bit DMA channel)
- the first number of the fourth entry (input/output rage – this should be the Soundblaster's base address and have a value of something like 220, 240, 260, etc)
- the first number of the fifth entry (input/output range – this should be the Soundblaster's MPU-401 midi port base address and should have a value of 300, 330, etc).

Place these numbers in their respective places on the DOS environment string (line of text) in your autoexec.bat file.

Roland MPU-401 and compatible

These interface cards (generally) do not use a DOS environment variable to specify their base address and IRQ; therefore, the card's base address and IRQ are typed into the file hardware.cfg that converter uses for initialization (see the discussion on configuring this file earlier in this manual). Many of these interface cards (specifically the Roland cards, and many compatibles such as ones from Music Quest) used jumpers on the card to specify the base address and IRQ to be used. To determine the settings for your particular interface, either look at the interface card you have installed (and/or refer to the manual or documentation that came with the card), or if the card has been successfully installed under Windows9x refer to the card's settings under Windows as follows:

Open the Windows9x control panel, double-click on 'system', choose the 'device manager' tab, expand the 'sound, video and game controllers' droplist, and double-click the appropriate driver for your MPU-401 (in our case, 'MPU-401 Compatible' for our Roland MPU-IPC-T). This should give you a window with several tabbed entries – click on 'resources'. Under the 'resource settings' listbox, write the following numbers down on paper to be used in converter's **hardware.cfg** file:

- the first entry (input/output range – this is the base port address of the MPU card)
- the second entry (Interrupt Request – this is the IRQ your card uses)

Open Notepad (under Windows, or Edit under DOS) and place these numbers in their respective locations in the hardware.cfg file.

Soundcard manufacturer websites/webpages for manuals and drivers

Roland MPU-401, MPU-IPC-T, SCC-1, LAPC-1, etc.

Documentation available as HTML or PDF – scroll down until you find the section for MPU cards.

`http://www.rolandus.com/SUPPORT/DOCS/SUPNOTES.HTM`

Gravis Ultrasound Classic and Plug & Play series

Documentation, software installation, and utilities are available on this page. The soundcard sections are located towards the bottom of the list – try the following files:

- GUS411.ZIP (for Ultrasound Classic/Max series)
- P20DISK2.ZIP (for Ultrasound Plug&Play – may also require P20DISK1.ZIP but possibly not)

`http://www.gravis.com/support/sup_1059.html`

Creative Labs Soundblaster Series – DOS/Legacy drivers

This particular Creative Labs website has various DOS-oriented utilities and installation disks available for download. Note that these downloads are not designed for non-Creative Labs hardware (ie. clones). Specific downloads to try are:

- "Basic DOS-level utilties for use in Windows95 MS-DOS mode or a Windows 95 Command Prompt only boot" (towards the top of the list – DOS software such as Diagnose.exe which can be used to configure your soundcard)
- "Sound Blaster 16/SB32/AWE32 Basic Disk for DOS/Windows 3.1 Installation"
- "Plug and Play Configuration Manager" (2 disks, towards the bottom of the list, for the Plug and Play versions of the soundblaster cards such as AWE-64, etc)

`http://www.europe.soundblaster.com` (under technical support -> download drivers…)

creating a bootable DOS floppy to auto-run converter

Since converter and its associated files take up relatively little disk space, the application can be run from a single 1.44MB 3.5” floppy disk. This fact enables converter to become a ‘portable’ application that can be run ‘out of the box’ so to speak by creating a bootable DOS disk that boots your computer into ‘true’ DOS mode and auto-loads converter without intervention. Using converter becomes as easy as placing the floppy in your disk drive and rebooting – very useful for Windows9x users who do not want to modify the autoexec.bat file on their computer’s hard drive.

Here’s the steps:

- Find a blank 1.44MB 3.5” floppy disk, and format it as a bootable disk in one of the following ways:
    - **Under Windows9x Explorer shell:** Insert the blank disk in the disk drive, double-click “my computer”, select your floppy drive (typically A:), click on the File menu and select format. In the dialog box that appears, make sure “1.44Mb (3.5”)” is selected under the “Capacity” heading, then select “Full” under “Format type”, and select “Copy system files” under “Other options”. If desired, a label for the disk can be entered in the “Label” textbox (completely optional). Click on “Start” to begin the process, which will take a minute or so.
    - **Under DOS Command Prompt / ‘True’ DOS mode:** Insert the blank disk in the disk drive, type `format a: /s` (3.5” floppy drives are typically a: under DOS) and press enter, and then enter again at the “..press enter when ready..” prompt that appears. The disk will be formatted after a minute or so, and once completed, a prompt asking for a disk label will appear. Either type in a label and press enter, or just press enter (the label is optional). When prompted to “Format another (Y/N)”, type n and press enter.
- Copy all the files included in the converter package to the floppy disk. To conserve space on the floppy disk, the file `cmanual.pdf` can be omitted (see the converter file listing in an earlier section of this manual for specific information on files are not required for normal operation of converter)
- Check the `autoexec.bat` file on the disk and edit it (using Windows’ Notepad or Edit under DOS) in such a way that the “SET BLASTER” line reflects the specific settings of your soundcard (or if using a Gravis Ultrasound, replace the “SET BLASTER” line with the appropriate “SET ULTRASND” line for your card’s configuration), and save the file.
  (For more information on determining the hardware settings to use for your particular card, see the previous appendix section.)
- If using a mouse or other pointing device for midi conversion, place its driver on this floppy and put the command to load it (ie. `mouse`) on the next line in the `autoexec.bat` file (before the line with ‘`c`’, which loads converter – the mouse driver must be loaded before converter).

With this newly-created bootable disk, you just put it in your disk drive, boot up your computer (or reboot it), and converter automatically loads and runs without intervention. It also means converter can easily be run on other machines – all that is needed are (possibly) different numbers in the “SET BLASTER” or “SET ULTRASND” string in the `autoexec.bat` file if the other machine(s) have different configuration settings.

Note that this feature assumes your computer is configured to check the floppy drive for a disk when it is turned on – if it does not, check the manual that came with your motherboard or computer for a reference on the bootup BIOS settings which typically allow you to enable this feature.

tips to ensure best speed / performance

As with any complex system open to many different configurations and applications, there are certain factors which can influence converter's performance. As a summary of issues discussed in this manual which potentially affect the speed or response time of converter in live performance, the following tips and suggestions are designed as a general guideline for those wishing to improve performance, especially if a slower machine is being used.

- Figure out what your particular machine's speed is capable of with converter; avoid overloading the computer by trying to do too much at the same time. Specifically, if speed is an issue:
  - Disable the audio filter channels if they are unused – these take significant cpu power
  - Disable the gameport channels if they aren't being used – again, proper gameport interface routines take a consistently generous amount of cpu time, mostly in terms of a wait-state (due to the inherently cheap, ancient, and poor design of the IBM analog gameport interface).
  - Set the screen update option (root -> display settings) to minimal.
- Avoid using the vertical refresh interrupt option
- Use the 'Hardware & System Settings' display panel [F11] instead of the oscilloscopes display panel [F10]
- Monitor the midi bandwidth generated by converter by checking the midi data rate parameter in the hardware & system settings view screen (F11). Remember that midi's bandwidth (number of bytes it can transmit per second) is about 3000 bytes – typically keeping it under about 1500 or so is usually a good idea. This can be done by making greater use of the data reduction algorithms if many channels of continuous controller audio to midi conversion is being used.
- Monitor the computer's processing load by checking the processor buffer peak meter at the upper left of the screen – it should never be 'maxing out' constantly (even though the actual 'max' is much higher than the peak meter's max position).
- As a last resort, disable screen updating and see if that improves the situation by entirely disabling the real-time graphics.

gameport interface pinout

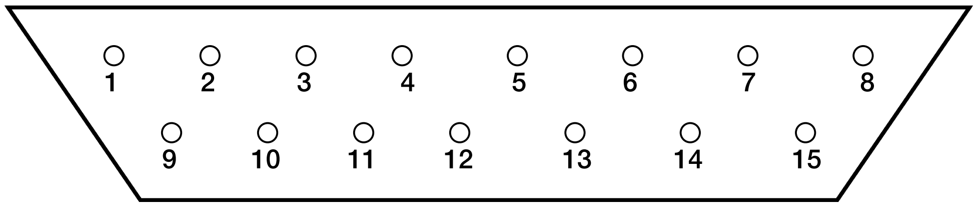
Warning! The following information is provided for users with concrete electronics knowledge and experience. There are 5-volt pins on the joystick interface that can provide a surprising amount of current if short circuited, causing potentially expensive and/or irreparable damage to components in the computer, as well as information loss, and any other number of disasters. By providing the following information, urr Sound Technologies Inc. in no way encourages general users of converter to attempt the construction of devices for interface to the PC gameport, and is in no way liable for any damages or loss of data as a result of the construction and/or use of any device based on the following information.

End of disclaimer ☺

Note that there are two differences on the joystick port found on a soundcard, such as the Gravis Ultrasound or Soundblaster, versus the original IBM-spec PC gameport interface – pins 12 and 15 are assigned to midi functionality instead of the original ground and +5v designations.

Use 100kΩ potentiometers on the axis inputs.

joystick D-sub connector pin numbering



Standard Gameport Pinout:

pin	connection
1	+5 volts DC
2	joystick A button 1
3	joystick A x-axis
4	ground
5	ground
6	joystick A y-axis
7	joystick A button 2
8	+5 volts DC
9	+5 volts DC
10	joystick B button 1
11	joystick B x-axis
12	ground
13	joystick B y-axis
14	joystick B button 2
15	+5 volts DC

Soundcard Gameport Pinout:

pin	connection
1	+5 volts DC
2	joystick A button 1
3	joystick A x-axis
4	ground
5	ground
6	joystick A y-axis
7	joystick A button 2
8	+5 volts DC
9	+5 volts DC
10	joystick B button 1
11	joystick B x-axis
12	MIDI TXD (Transmit)
13	joystick B y-axis
14	joystick B button 2
15	MIDI RXD (Receive)