Eventide®
the next step

Programming Manual for *Orville* and the DSP7000 family of Harmonizer® Brand Effects Processors.

(covering *Orville*™, DSP7000/7500™ and DSP4000B+™)

Part No: 141035 Manual Release 1.3 25 March, 2003

This page intentionally left blank

The Harmonizer® Programmer's Manual
Table of Contents

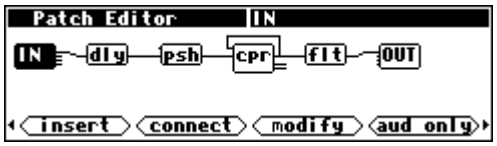
The Harmonizer® Programmer's Manual

This manual covers Orville™ as well as the DSP7000™ family of Harmonizer® Brand Effects Processors, as well as the DSP4000B+™. In the following text these will, for convenience, be referred to as 'Harmonizers'. Much of its contents also apply to the older 4000 family, but the reader is not advised to view this publication as an exhaustive reference for these models. This manual does not cover the Eclipse™ or the H3000 family of Harmonizer® Brand Effects Processors. In most cases, a reference to the DSP7000 also refers to the DSP7500 and DSP4000B+.

One of the reasons Eventide's effects units are so versatile is that their effects programs are "modular." A single program is composed of many smaller "modules." Modules might best be thought of as good old fashioned "guitar pedals" *(except, of course, that unlike guitar pedals, the modules in the Harmonizer are 24 bit, crystal clear, high-end audio processors!)*. Imagine you have a gym floor covered with guitar pedals and a coat rack draped with patch cords. You run around connecting pedals, a delay pedal to a pitchshifter pedal, the output of that pitchshifter pedal to a compressor, the output of that compressor into a filter, etc. The end result of all that patching is, to the Harmonizer, a *program*.

Although that picture is oversimplified, it does capture the essence of what's going on inside VSigfile (Eventide's PC-based Graphic Design Environment) or the Harmonizer's built-in Patch Editor. You're just connecting modules (guitar pedals) to each other to produce a desired overall program.

Without going into details, the example cited above, "a delay pedal into a pitchshifter into a compressor into a filter" is shown to the right *as seen in the Patch Editor*. The little boxes represent the modules and the lines between them represent "patch cords." IN represents the inputs to the DSP (Digital Signal Processor) running the program, dly represents the delay module, psh represents the pitchshifter module, cpr represents the compressor module, flt represents the filter module and OUT represents the outputs from the DSP running the program.



Of course, if things were going to remain this simple there would be no need for this separate Programmer's Manual. But *conceptually*, things are this simple! We'll muddy things up by implementing "control" features that will make your programs easier to use in the PARAMETER area. We'll further muddy them by making *large* programs that utilize many modules connected in ways that defy the "serial/parallel" paradigm. So the details may get a little complex, but the main idea should remain crystal clear: we're just connecting a bunch of 24 bit, full bandwidth guitar pedals!

The first chapter in this manual, ***General Principles***, will cover the underlying concepts involved in constructing programs either in VSigfile or the Patch Editor area. It is essential reading. The second chapter will discuss the mechanics of creating programs in the Patch Editor area. Vsigfile fans should refer to the online User Manual and Help files supplied with Vsigfile, but may wish to read this section for a different viewpoint.

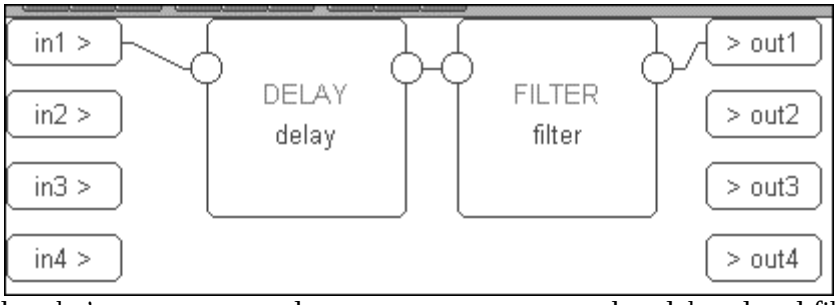
Information on the individual *modules* may be found either via Vsigfile's help system, or as a document on Eventide's Web Site.

GENERAL PRINCIPLES

OVERVIEW

This section will describe in general terms just what goes into constructing a program. Return to this section if you ever feel like you're being mired down in details later on.

First, the primary "stuff" of program construction is the "module." Modules are small, functional "chunks." Some modules may have names that will be familiar to you, such as `delay`, `reverb`, `filter`, `pitchshifter`, and `eq`. As you would expect, a `delay` module delays the signal at its input. A `reverb` module adds reverb to the signal at its input. A `filter` module filters the signal at its input. And so on.



Before going any further, let's say you wanted to construct a program that delayed and filtered a signal. You would begin either in VSigfile or the Patch Editor area with a "blank slate" that contained nothing but representations of the inputs and outputs of the DSP that would run the program. You would then add a `delay` module and a `filter` module. Lastly, you would connect one of the DSP's inputs to the `delay` module, the `delay` module's `output` to the `filter` module, and the `filter` module's `output` to one of the DSP's outputs. The result, as seen in VSigfile, is shown above.

Most modules, `delay` and `filter` included, have "control inputs" that allow you to change parameters associated with a given module. For example, a `delay` module has a control input that allows you to change the delay time for the module (will it delay the signal 20ms or 1000ms?). A `filter` module has three control inputs: one for the cutoff frequency, one for the resonance at the cutoff, and one to select the type of filtering done by the module (lowpass, highpass, notch, or band).

We normally construct programs so that parameters such as the ones described above can be altered in the PARAMETER area of the Harmonizer *(like the factory presets you've probably already played with)*. Some things called "*userobject* signals" are used in the construction of a program to create and organize menu pages of parameters in the PARAMETER area.

The three paragraphs above capture the three **cornerstones** of program construction in the Harmonizer.

1. We must connect appropriate modules to achieve a desired, overall audio effect.
2. We must control the parameters of the modules in a program so that the desired audio effect is achieved.
3. We must make some of the parameters available in the PARAMETER area so that the user can "tweak" the program to fit a particular situation.

Much complication will now be heaped upon the three **cornerstones**, but all of the complication is introduced in order to achieve the goals set out in the three **cornerstones**! Don't lose sight of the three **cornerstones**, as they motivate everything that follows! Get it - **cornerstones** !

To gain a greater appreciation for what we are doing when we construct an the Harmonizer program, consider the following analogy:

> Computer programs basically compute things. The computer user however, is not directly involved in actual computation (thank goodness). The user *does* direct the computer regarding *what* computations it should carry out and receives the results of those computations through a "user interface." The user interface on a computer is typically a monitor, a keyboard, and a mouse. The lucky individual who *designs* a computer program on the other hand, needs to consider both the actual computations that the computer performs *and* the way those computations will be controlled and displayed at the user interface.

By analogy, when you construct a program for the Harmonizer you must consider the actual audio manipulations carried out by the program *(cornerstones one and two)* ***and*** the way those manipulations will be controlled and displayed at the user interface *(cornerstones two and three)*. In this context, the user interface is the PARAMETER area in conjunction with the front panel keys and display. *Don't worry, constructing programs for the Harmonizer is decidedly easier than even the easiest computer programming!*

Unfortunately, we must discuss these two charges "bass ackwards." with user interface coming first and actual audio manipulations coming second. The latter can't be properly understood without the former. *(If you've ever learned a computer language, the first thing they teach you is how to print "Hello" on the monitor!)*

But first we'll take a brief detour and look at the different types of *signals* that interconnect modules in the Harmonizer. After that, we'll talk about the user interface.

Different Kinds of Signals

To achieve the goals set out by the three cornerstones, we must employ four signal types. Signals connect modules together. The four signal types are:

Audio Signals	Used to pass full bandwidth audio between modules in accordance with cornerstone one. Audio signals are represented numerically by a value between -1 and +1.
Control Signals	Typically used to pass parameter values between modules in accordance with cornerstone two. Control signals are low speed and are updated at a variable rate, depending on how busy the Harmonizer is. Control signals are represented numerically by a value between -32768.0 and +32767.999.
Mod Signals	Used to pass "modulation signals" between modules. A "modulation signal" is a 1/4 bandwidth audio signal. Mod and audio signals *may* be interconnected, but using mod signals to pass audio *will* result in a significant loss of signal quality. Although mod signals *look* like audio signals, they actually work to achieve cornerstone two (controlling the parameters of a module). In some cases, control signals are too slow to alter a parameter without "clicking" or "stuttering." For instance, if you wanted to alter a delay time quickly to produce a flange effect, a control signal might not be equal to the job. Thus certain modules (`moddelay` for instance) come equipped with a mod input. Other modules (such as the low frequency oscillator (`LFO`)) come equipped with a mod output. By interconnecting the two, fast, smooth parameter adjustment can be executed that would defy control signals.
Userobject Signals	Used to pass PARAMETER area menu page information in accordance with cornerstone three.

There exist module inputs and outputs for each of the four signal types. They are named (logically enough):

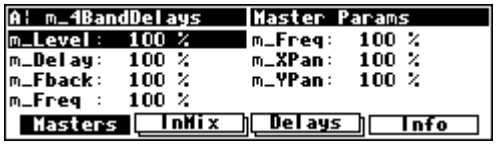
- audio inputs/outputs
- control inputs/outputs
- mod inputs/outputs
- *userobject* inputs/outputs

Only inputs and outputs of a similar type may be interconnected *(except for audio and mod inputs/outputs)*. A given module will only have those types of inputs/outputs that are pertinent to its function.

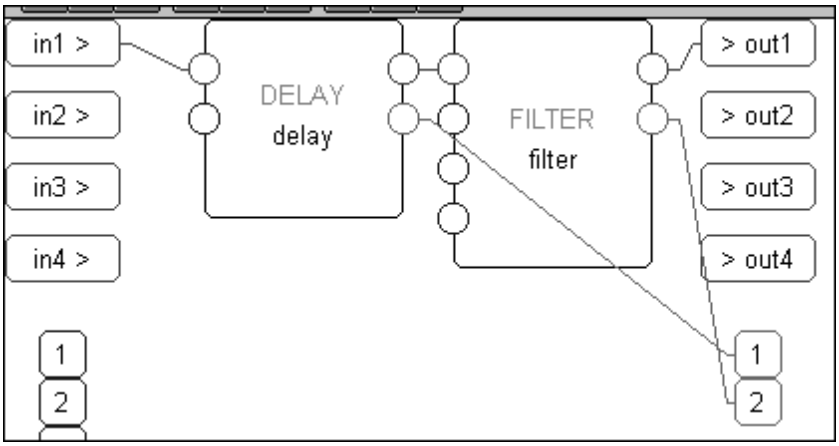
HOW A PROGRAM INTERFACES WITH THE PARAMETER AREA

Simple Interface

Because you really shouldn't be reading this manual if you haven't already read the ***User Guide***, we'll assume you've seen menu pages in the PARAMETER area.



A menu page, with an associated SOFT KEY, is created by connecting a module's *userobject* output to a *userobject* input on something called the "head" module. Every program has one (and only one) head module. The actual parameters that will appear on a menu page created this way depend on the module being connected. They will usually be the values of all unconnected *control inputs*..



For example, consider the simple delay module connected to a filter module we started this chapter out with *(again, as shown in VSigfile)*. Now that you've learned a little more, we've "unhidden" the control inputs for these modules *(the unconnected ones on the left)* and the *userobject* outputs *(the ones on the lower right)*. As you can see, we've connected their *userobject* outputs to the *userobject* inputs on the head module *(the disembodied "1" and "2" in the lower right corner. The observant user will spot that the head "module" doesn't really look like the other modules)*.

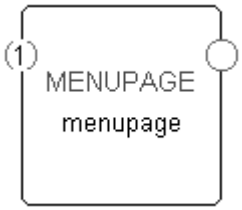
If we run this program and go to the PARAMETER area, we see the screen to the right. A menu page exists for each module that contains parameters pertinent to its functioning.

Note that the order of the connections to the head module's *userobject* inputs dictates the order of the SOFT KEYS.

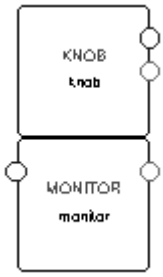
Constructing programs this way is fast and easy. Just concentrate on the audio connections and then connect every modules' *userobject* output to the head module. However, the user interface isn't very "slick" and may be cumbersome to use. That's where "custom" interface construction comes in. . .

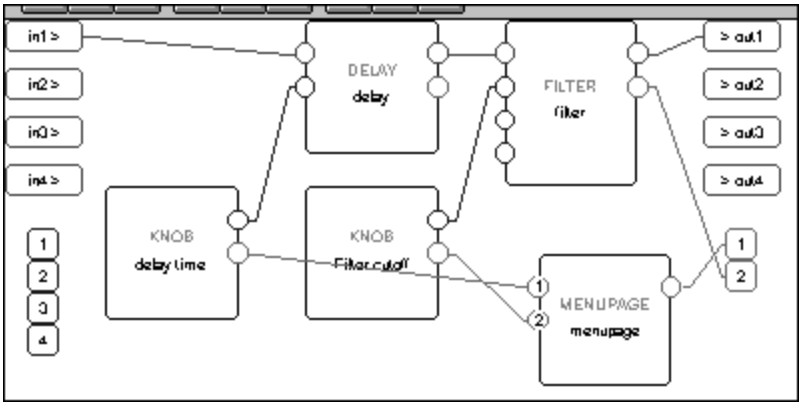
Custom Interface

There is one very special module that is used to create custom menu pages. It's called (appropriately) the `menupage` module. It has any number of *userobject* inputs *(the example shown to the right has only one)* and a single *userobject* output. The `menupage` module will create a menu page out of the *userobject* outputs that are connected to it.



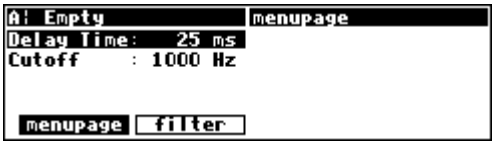
The `menupage` module is typically used with a special group of modules called the "interface" group. Most of the modules in the interface group have a *userobject* output and either a single control output or a single control input. The interface modules that have a control output *(like the "knob" module shown to the right)* are connected to the control input of another module. The interface module then "takes over" that control input. Similarly, interface modules that have a single control input *(like the "monitor" module shown to the right)* are connected to the control output of another module to display the value of that control output. *(It should now be clear that not all modules exist to deal with audio. Many, such as the interface modules, exist to create a user interface. Still others exist to manipulate control signals.)*





For example, we could utilize two `knob` modules and a `menupage` module in the program we've been working on. One `knob` module will take over the `delay time` control input on the `delay` module. The second `knob` module will take over the `frequency` control input on the `filter` module. The *userobject*s of both `knob` modules are connected to the `menupage` module, which is in turn connected to the `head` module.

When we run the program on the Harmonizer, the screen shown to the right appears in the PARAMETER area. Notice that the order that the `knob` modules' *userobject* outputs are connected to the `menupage` module dictate their order on the menu page in the PARAMETER area.



Also notice that because the second `knob` module "took over" the `filter` module's `frequency` control input, that parameter no longer appears on the `filter` module's menu page.

The menu pages found in the factory presets were almost exclusively made with interface modules and `menupage` modules.

Now that you have some understanding of audio signals, control signals, and *userobject* signals coupled with an understanding of how they all play a role in making a program both functional and accessible from the PARAMETER area, we can discuss modules in a little more depth.

MODULES

Modules are the magic that make the Harmonizer shine. They are signal processing "nuggets" that are interconnected (via the *signals* discussed above). Before we discuss aspects of the typical modules like the `delay` module, the `filter` module, the `pitch shifter` module, etc., we need to look at the more specialized `IN` and `OUT` "modules."

The IN and OUT "Modules"

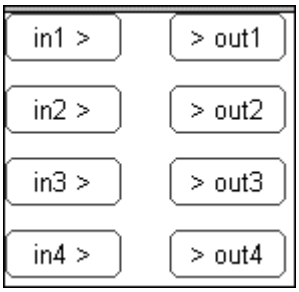
Orville

Orville's programs are loaded and run one at a time on a given DSP. The DSP running the program provides the program with four channels of input audio (where that input audio comes from is a function of the routing configuration, see the Harmonizer's ***User Manual***). The DSP running the program also takes the four channels of output audio from the program (where it is subsequently sent is again a function of the routing configuration).

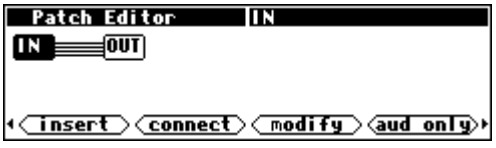
DSP7000

The DSP7000's programs are loaded and run on its single DSP. The DSP provides the program with two channels of input audio and takes two channels of output audio from the program. The remainder of this manual will show Orville-style four channel processing, but the idea is the same with the DSP7000's two channels. If you send a program that has more than two inputs or outputs to your DSP7000 from VSigfile, it will not be accepted.

The input audio and output audio connections to the program are handled through a pair of pre-defined modules called `IN` and `OUT`.
The IN module has up to four signals to send to the program, labeled 1, 2, 3, and 4. Since these signals are coming *from* the module, they are called `outputs` of the module. A small amount of confusion might result because the `IN` module has `outputs`. Similarly, the `OUT` module has `inputs` labeled 1, 2, 3, and 4. This difficulty is minor compared to the gain in consistency created by using the word `output` to refer to all signals that come *from* a module, and using the word `input` to refer to all signals that go *into* a module.



In the simplest of conceivable programs, the `IN` module's `outputs` are connected directly to the `OUT` module's `inputs` (this is the `Thru'` program in bank 0). Normally, other, optional modules are inserted in-between the `IN` and `OUT` modules.



The `IN` and `OUT` modules always remain as part of the program.

The Characteristics of Modules

There are several characteristics associated with any module. All modules have:
- a module type
- a module name

Modules use memory and processing resources that can be divided into the following groups:
- audio memory
- signal processing
- user interface and control signal memory
- control processing

Different types of modules use different amounts of these resources.

Modules that have audio inputs and outputs introduce a six-sample delay in the processed signal.

More complex modules have some or all of these items:
- *specifier*s
- audio inputs (and/or mod inputs)
- audio outputs (and/or mod outputs)
- control inputs
- control outputs
- *userobject* outputs
- *userobject* inputs

The following sections will discuss all of these attributes in depth. . .

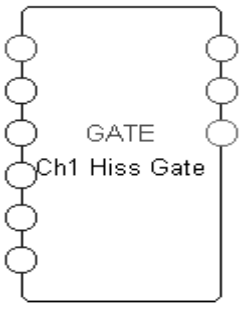
MODULE TYPE

There are many kinds of modules at our disposal. The "module type" simply defines a module as being a particular *kind* of module. When a module is added to a program, it is selected by module type. Once added, the module type cannot be changed. If a different module type is needed, the "offending" module must be deleted and then the correct module type must be added anew.

When a module is mentioned in this document, it is referred to by module type. For example, a module whose module type is "`samphold`" would be referred to as a `samphold` module.

MODULE NAME

The module name is a text string that is stored with a particular module. It is helpful to change the module name immediately after adding a module so that modules of the same type can be told apart. Choose a name that reflects both the purpose of the module within the patch, and the module type. The name may be up to 18 characters in length. To the right we see `gate` type module named "Ch1 Hiss Gate."



RESOURCES

A resource is something that is needed for the operation of a Harmonizer program -there are several different kinds of resources. A program cannot run if it needs more of a particular resource than is available.

Audio memory

Modules that store audio for brief periods of time use audio (delay) memory. Modules that use audio memory include modules in the delay, filter, pitch shift, and reverb groups. Some modules contain "*specifier*s" *(see below)* that increase or decrease the amount of audio memory used by either varying the number of audio channels or by specifying the amount of delay explicitly.

Signal Processing

Modules that perform operations on audio use signal processing. The amount of processing performed by a module can only be changed via *specifier*s *(see below.)* This is important, as the amount of signal processing that can be done in any given period is finite. Modules that perform complex effects on audio use more processing than those that perform simple effects. For example, the `reverb_a` module uses more processing power than the `delay` module, even though the `delay` module might use more audio memory.

User Interface and Control Signal Memory

Interface memory includes memory used to store text, adjustable range limits, default values, control inputs, control outputs, and any data used by "control" modules. Modules that use text fields consume a large amount of this kind of memory. For instance, it is possible to use up all of the user interface memory with just two `textblock` modules if each contains enough lines of text.

Control Processing

Control processing is a resource that cannot be exhausted, though it can be strained. The Harmonizer will repetitively process everything that comes under the control process category as often as possible. Control operations will get slower as more operations are required. For instance, if a single menu page has eight values displayed that are all changing rapidly, the display may appear to update slowly. Typically, control values are updated about 100 times a second.

SPECIFIERS

A *specifier* is a control that affects a module's behavior. For example, a `delay` module might have a *specifier* that sets the maximum delay time a user can enter. A `pitchshifter` module might have a *specifier* that sets the number of pitchshifting voices used by the module. A module may have several *specifier*s.

*Specifier*s are only adjustable in the Patch Editor area or in VSigfile (i.e. *specifier*s can never be altered in the PARAMETER area). There is no input or output for *specifier*s; they reside "inside" a module (you'll learn how to access the "inside" of a module in the VSigfile or Patch Editor chapters).

*Specifier*s have the following features:

- they are extremely efficient in terms of resources. (A module with a *specifier* for a given characteristic is more efficient than a module with a control input for that characteristic.)
- they can change the amount of resources that a module needs.

- they can change the number of audio, mod, and control inputs and outputs, or even the number of other *specifier*s (!) for a module.
- they can be numerical, multiple choice, or text.
- They cannot be changed in real time.

AUDIO INPUTS

An audio input is used to pass high fidelity audio into a module. An audio input can be connected to at most one audio or mod output. Unconnected audio inputs are actually attached to a special "null signal" provided by the Harmonizer's operating system. The null signal simulates a zero voltage, noise-free audio source. Audio signals range if value from -1 to +1, or full negative to full positive. Audio inputs are always found on the left side of modules.

AUDIO OUTPUTS

An audio output is used to pass high fidelity audio out of a module. An audio output may be connected to any number of audio or mod inputs. Audio outputs are always found on the right side of modules.

CONTROL INPUTS

One module can control the parameter of a second module by connecting to the second module's control input *(as we saw the knob modules doing in the "Custom Interface" section)*. The range of values a control input can accept may be set by a *specifier*, by fixed internal programming, or even by another control input. A few notes concerning control inputs:

- Control inputs are always found on the left side of a module.
- The value of a control input cannot change the amount of resources used by a module.
- The existence of a control input takes up processing and memory resources. In modules with a variable number of control inputs (like the `c_switch` module), reducing the number of inputs reduces the amount of resources used. *(In such modules, specifiers control the number of control inputs.)*
- Control inputs can be connected to only one control output.

CONTROL OUTPUTS

A control output sends a numerical value to another module by connecting to one of the other module's control inputs. A single control output can connect to any number of control inputs. Control outputs are always found on the right side of a module.

MOD INPUTS

A mod input is used to pass a high performance modulation signal into a module. A mod input may be connected to at most one audio or mod output. Unconnected mod inputs are actually attached to a special "null signal" provided by the Harmonizer's operating system. The null signal simulates a zero voltage, noise-free audio source. Mod signals range if value from -1 to +1, or full negative to full positive. Mod inputs are always found on the left side of a module.

Although mod signals are high performance modulation signals, they might be said to stink at passing audio signals *(they were never really meant to! Remember, they act to achieve cornerstone two - to control the parameters of modules)*. An audio signal passed through a mod in/mod out on a module will lose fidelity. This is because the sampling rate used for mod signals is 1/4 that used for audio signals. *(Of course if you go for that retro, "aliasing." dawn-of-the-samplers kind of sound, mod signals might be right up your alley!)*

MOD OUTPUTS

A mod output is used to pass a high performance modulation signal from a module. A mod output may be connected to any number of audio or mod inputs. Mod outputs are always found on the right side of a module. See the comments made immediately above concerning the "low-fi" status of mod signals.

USEROBJECT OUTPUTS

Most modules have a *userobject* output. The *userobject* output can be connected to the *userobject* input on a menupage module, the head module, or a gang module. Such a connection will allow the module's parameters to be accessible in the PARAMETER area. The existence or use of a *userobject* does not affect system resources or memory. This means that menu pages can be created without using much in the way of resources or program memory.

In VSigfile, *userobject* outputs are always found on the right side of a module. In the Patch Editor area, *userobject* outputs are not explicitly shown.

USEROBJECT INPUTS

A handful of modules (gang, head, and menupage) have *userobject* inputs. This means that these modules can accept as inputs other modules' *userobject* outputs. For instance, a menupage module may be used to create a PARAMETER area menu page by accepting the *userobject*s of other modules.

In VSigfile, *userobject* inputs are always found on the left side of a module. In the Patch Editor area, *userobject* inputs are not explicitly shown.

WRAP UP

OK, so that completes our birds-eye view of the program construction process. Recall that all of our constructing is done to satisfy the three so-called "cornerstones":

1. *We must connect appropriate modules to achieve a desired, overall audio effect.*
   This is achieved by connecting audio-manipulating modules via audio signals. The "heart" of the program lies in its audio construction.
2. *We must control the parameters of the modules in a program so that the desired audio effect is achieved.*
   This is achieved by using mod signals and control signals to alter the parameters of the audio-manipulating modules.
3. *We must make some of the parameters available in the PARAMETER area so that the user can "tweak" the program to fit a particular situation.*
   This is achieved by connecting userobject outputs to the head module. Additionally, menupage modules may be used in conjunction with interface modules to create custom menu pages.

That completes the *theory* of program construction, but much remains in the way of *execution*. We'll cover that in the Patch Editor Chapter. Much of this also applies to Vsigfile, which is decidedly easier to use than the Patch Editor, especially for constructing large programs.

PATCH EDITOR

GET COMFORTABLE BY DOING

Let's make a patch, OK? That way you'll get a feel for how the Patch editor works, and you'll have a much better understanding of things when you read about the details later.

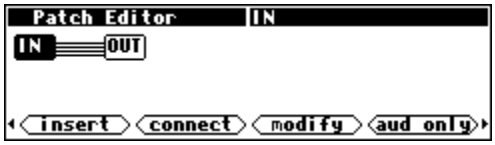
The patch we're going to make will be an audio compressor. A compressor reduces the audio gain when louder signals are input. The compressor we will build is constructed from a single "ducker" module.

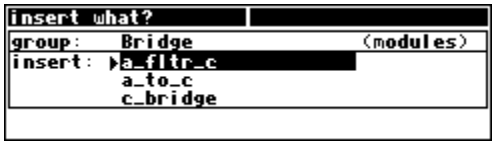
> The ducker module is the basic building block for most dynamics control patches. It is essentially a dynamic range compressor with separate inputs for the signal whose gain is to be processed and for the detection (*sidechain*) input.
>
> By connecting sidechain to the output, a basic compressor is built. By connecting a dry signal to the sidechain and a processed signal to the input, the processed signal can be ducked (have its gain reduced) during louder passages of audio.
>
> Ducking is often used by radio talk show hosts such that the host's audio overrides the guest or telephone caller. Each time the host talks the caller's audio is dropped down such that the host's audio is much louder. If the host talks loudly, the caller's audio disappears altogether.

Since we'll be using the ducker module as a compressor, we'll loop the output audio back to the sidechain input. Try the following tutorial out on your Harmonizer:

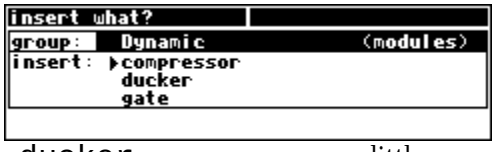
To start, go to the PROGRAM area and load the Thru' or Empty program from the Programming bank.

Next, go to the Patch Editor area by pressing and holding the PARAMETER key. We're looking at an empty slate. The only things we see are the IN and the OUT modules, which exist in every patch. The IN module is where audio signals come into your program, and the OUT module is where audio signals exit your program. In its current configuration, the DSP running Thru' should be passing audio unchanged, just as the patch display shows.



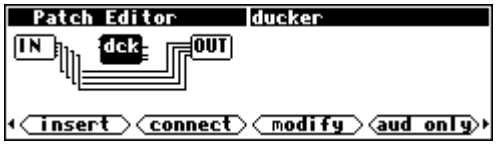


To start creating the compressor, insert a ducker module. Press the <insert> SOFT KEY. You will get a list of things to insert.

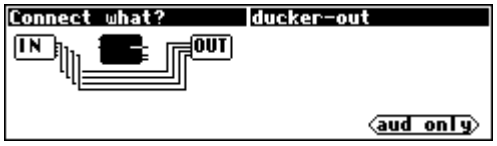


We want a ducker, which is in the "Dynamic" group of modules. You can either turn the KNOB until you see the little arrow pointing to ducker, or you can save a little

time by scrolling through groups first. To do the latter, press the LEFT CURSOR key so that the group name is highlighted, and then turn the KNOB until you see the "Dynamic" group.

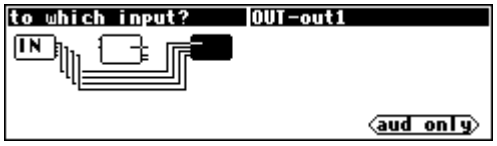


Press the RIGHT CURSOR key twice to highlight the `ducker` and press the SELECT key. A `ducker` will appear in your program. It's the little box marked `dck`.

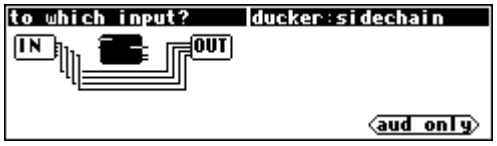


Note that the `IN` and `OUT` modules are still connected to each other, just as they were. Audio is still passing through the DSP running `Thru` unchanged!

Now we need to make a compressor from the `ducker`. Press the <connect> SOFT KEY.

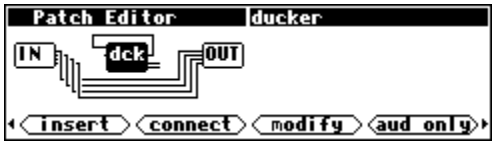


The upper left-hand side of the display now asks you what you want to connect. The upper right-hand side of the display gives the name of the currently selected output. The center of the display shows the currently selected output as a highlighted little line inside the module's box. You can choose to connect a different output instead by pressing the RIGHT or LEFT CURSOR key or by turning the KNOB. Right now we want to connect the `ducker`'s `output`, which *is* the currently selected output. So just press the SELECT key.



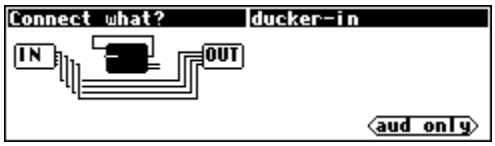
Notice that the output we selected is still identified by a little line inside its module box, but the box itself is not highlighted. The little line tells us what we're connecting *from*. The editor is now asking what input we would like our previously selected output connected *to*. The currently selected input is shown as a highlighted little line inside the `OUT` module. Rotate the KNOB (or use the LEFT or RIGHT CURSOR key) to select the `ducker`'s `sidechain input`. As you move the highlighted little line between available inputs, the upper right-hand side of the display will show the currently selected input and the name of the module it resides on.

The display to the right shows the ducker highlighted, with its output selected and its sidechain input *ready* to be selected. Press the SELECT key.

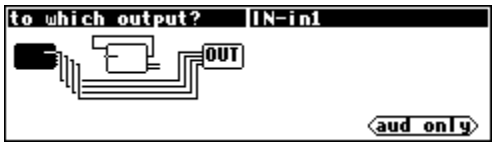


There you have it. There's a wire connecting the ducker's output to its sidechain input.

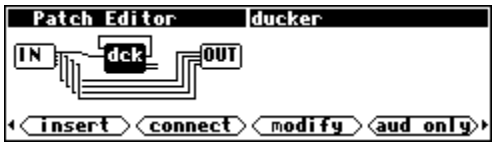
Now let's connect the ducker's main input to the IN module. Press the <connect> SOFT KEY and then the LEFT CURSOR key twice. The ducker's main input should be highlighted as shown to the right.



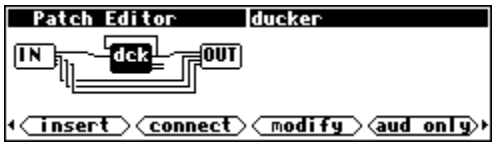
Press the SELECT key to actually select the ducker's main input. Now we need to select something to connect the ducker's main input *to*. As luck would have it, the IN module's input 1 is the currently selected candidate.



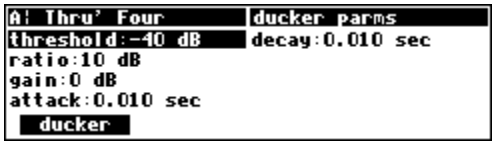
Press the SELECT key to complete the connection.



The last thing we need to do is connect the OUT module to the ducker so that we can hear what the compressor sounds like. Press the <connect> SOFT KEY and then the SELECT key to select the ducker's output. Press the SELECT key again to connect that to the OUT's output 1.



Now the compressor will be heard on the first "channel" of the DSP running the program, while the remaining three "channels" go uncompressed. Press and hold the PARAMETER key to see the ducker module's menu page.



Now that you're somewhat familiar with the mechanics of inserting and hooking modules up, let's move on to something a little more comprehensive.

The IN and OUT "Modules"

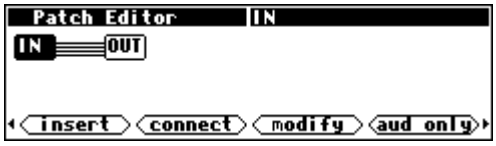
Orville

Orville's programs are loaded and run one at a time on a given DSP. The DSP running the program provides the program with four channels of input audio (where that input audio comes from is a function of the routing configuration, see the Harmonizer's ***User Manual***). The DSP running the program also takes the four channels of output audio from the program (where it is subsequently sent is again a function of the routing configuration).

DSP7000

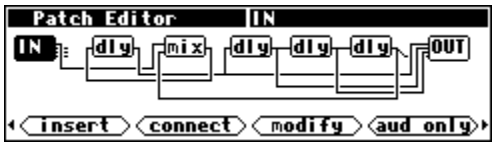
The DSP7000's programs are loaded and run on its single DSP. The DSP provides the program with two channels of input audio and takes two channels of output audio from the program. The remainder of this manual will show Orville-style four channel processing, but the idea is the same with the DSP7000's two channels. If you send a program that has more than two inputs or outputs to your DSP7000 from VSigfile, it will not be accepted.

In the simplest of conceivable programs, the `IN` module's `outputs` are connected directly to the `OUT` module's `inputs` (this is the `Thru'` program in bank 0). Normally, other, optional modules are inserted in-between the `IN` and `OUT` modules. The `IN` and `OUT` modules always remain as part of the program.

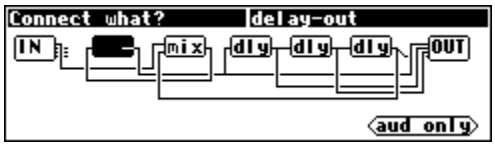


THE PATCH EDITOR AREA DISPLAY

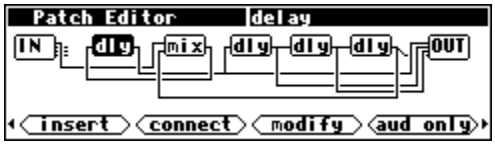
When the PARAMETER key is pressed and held, the Harmonizer presents a Patch Editor area display of the current program along with a selection of SOFT KEYS. This is the default Patch Editor area screen. Unlike the other areas in the Harmonizer, the top line of the screen is used for "special purposes."



The left half of the top line is used as a question field when the <connect> or <unplug> SOFT KEY is used.



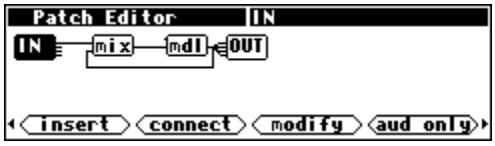
The right half of the top line shows the name of the currently selected module (except during <connect> or <unplug> operations when it shows the currently selected input or output).



A block diagram of the program takes up most of the display. As mentioned before, a program consists of a series of modules. Each module is shown on the display as a block with lines indicating its inputs and outputs. Inputs are on the left side of a module while outputs are on the right side. Each module is shown with a three-character (or less) abbreviation of its function name.

→ See the ***Modules Section*** for a list of all modules.

The example screen to the right shows four modules and is shown in the default "audio only" mode. This means that the only modules and signals shown are audio paths and modules that work with audio. The modules shown in the example are:



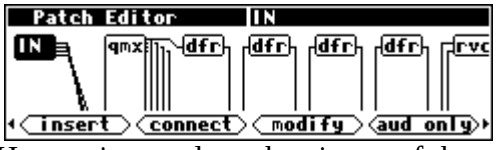
IN audio from the DSP's four inputs (only one is being used)
mix a two-input mixer
mdl "modulate-able" delay
OUT audio to the DSP's four outputs

As shown, the `IN` module's `output 1` is connected to one of the `inputs` of the `mixer`. The other `mixer input` comes from the `output` of the `"modulate-able" delay`. The `mixer` feeds the `input` of the `delay`. The `delay output` may be seen to drive five module inputs: the `mixer input` and all four of the `OUT` module's `inputs`.

Front Panel Controls

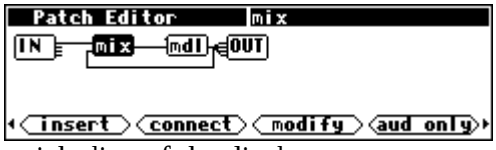
There are several controls used to manipulate the Patch display.

Knob



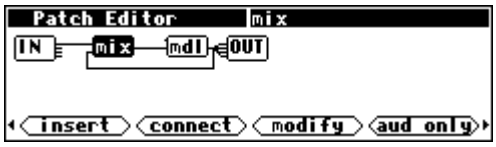
In many programs, the patch diagram will be larger than the screen. In such a case, the screen will display only part of the program. The KNOB may be used to shift the screen. A complicated program will move more slowly across the screen as the KNOB is rotated. This is due to the processing required for the Harmonizer to draw the picture of the program. If the KNOB is rotated faster than the screen moves, the screen will jump to catch up. If the KNOB is rotated very fast, the screen immediately jumps to the end of the program. The screen will not "wrap around" to the other end of the program.

Cursor Keys

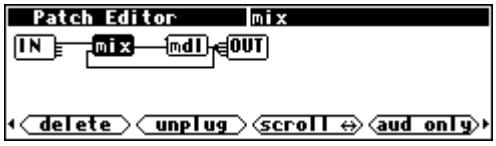


The LEFT and RIGHT CURSOR keys are used to highlight (select) modules. When you first enter the Patch Editor area, the `IN` module is highlighted. As shown on the example screen to the right, pushing the RIGHT CURSOR key causes the `mix` module to be highlighted. The name of the selected module is shown in the top right line of the display.

PARAMETER key



The PARAMETER key is used to toggle between sets of SOFT KEYS. Tapping the PARAMETER key on the screen to the right would give you access to. . .

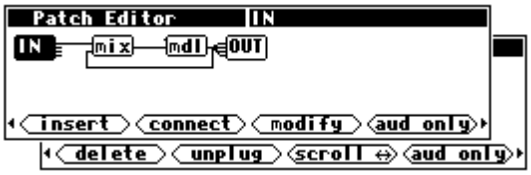


. . .these SOFT KEYS. Notice that you need only *tap* the PARAMETER key. If you hold it down, you will exit the Patch Editor area and return to the PARAMETER area.

→ To adjust the "hold time." key hold parameter on the [misc] menu page in the SETUP area.

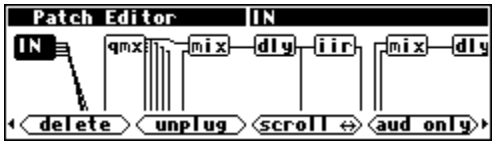
The Patch Editor Area SOFT KEY Functions

In the Patch Editor area, there are seven SOFT KEY functions. <insert> and <delete> add and remove modules from a program. <connect> and <unplug> add, remove, or change a signal connection. <scroll> changes the direction that the KNOB moves the display. <aud-only> changes the display mode to show audio and control lines instead of just audio lines or to show menupage modules. <modify> makes changes to internal module details and *userobject* information. Let's take a closer look at the functions of all these SOFT KEYS.

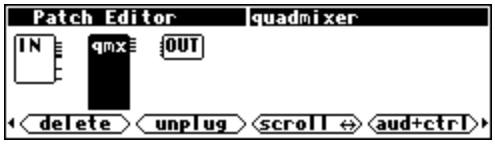


Scroll Direction *<scroll>*

The <scroll> SOFT KEY selects the direction of motion that the KNOB causes. This is useful if the program you are editing has so many signals that they dip below the level of the screen. . .

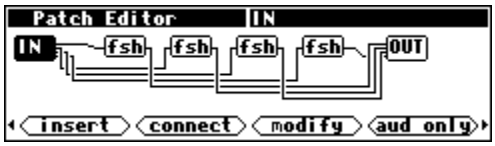


. . .or if one or more modules have enough inputs that they are taller than the screen. Here, the qmx module has inputs that exist "below" this screen.



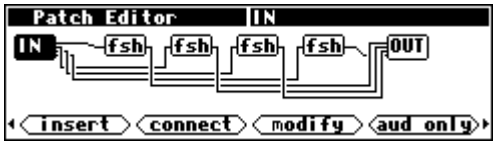
The <scroll> SOFT KEY changes to reflect current scroll mode, allowing you to scroll either horizontally or vertically.

The screen will not move if there is no off-screen information in the direction that you are spinning the KNOB. Thus, in the case of a simple program such as the one shown to the right, selecting scroll motion up and down and then rotating the KNOB will cause no change.



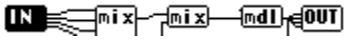
Display Mode *<aud only>*

It is quite possible to construct a complex program without using control signals. Eventually however, you'll want to create *custom* PARAMETER area menu pages. This is accomplished by using knob modules, fader modules, and other interface modules that use control signals. Most of the factory presets that come with the Harmonizer were created using these modules. One of the consequences of using interface modules is that there are usually more control signals than audio signals. As a result, what might have been a fairly "viewable" "patch" in terms of its audio signals becomes quite complex in terms of its control signals. To allow the patch to be viewed in a simplified manner, a feature exists that excludes control signals from the Patch Editor area display. Furthermore, modules that have no audio signals (this includes knobs, faders, etc.) are not shown in the aud only view. Note that mod signals are treated like audio signals by the editor; they will show up in the aud only view.

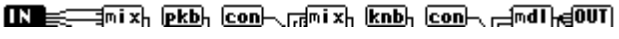


The right-most SOFT KEY provides display mode control. When this key is pressed, the display mode will change to the next mode in this order: aud only, aud+ctrl, ctrlonly, misc, aud only, aud+ctrl, etc. The right-most SOFT KEY's label will change to indicate the current mode. Upon entering the Patch Editor area, the right-most SOFT KEY is in the aud only mode, and only the audio path is visible. Control signals (and modules that contain only control inputs or outputs) are hidden. Note: most modules that have *audio* inputs or outputs also have *control* inputs or outputs.
The following images are composite pictures of a simple program in all 4 display modes.

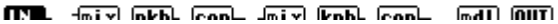
<aud only>



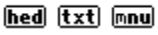
<aud+ctrl>



<ctrlonly>



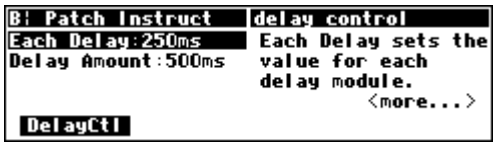
<misc>



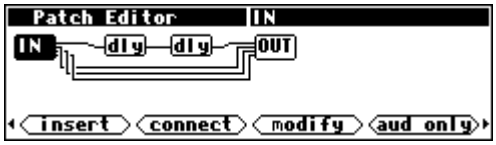
By comparing these different pictures, we can determine which of the wires in the aud+ctrl picture carry audio signals and which carry control signals. The misc display mode will be discussed later.

Connect Modules <connect>

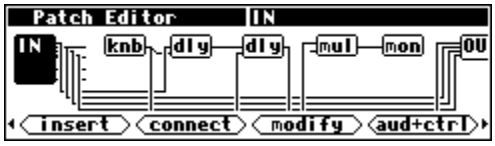
Pressing the <connect> SOFT KEY starts a process that will connect a module output to a suitable input. The Patch Editor will prompt for a starting input or output and then will prompt for a complementary destination. The Patch Editor automatically limits the available destinations to legal selections. For example, if a connection is started from a control input, only control outputs will be offered. Similarly, if a connection is started from a mod output, only audio/mod inputs will be offered. To abort a connect, press the PATCH key.



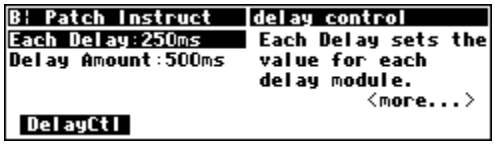
To illustrate, load the program Patch Instruct from the "Programming" bank. This program consists of a pair of delay modules, connected in series between DSP input 1 and DSP output 1. The remaining DSP inputs and outputs are "hardwired" one to the other. Press and hold the PARAMETER key to see what this "patch" looks like.



As you can see, there are two delay modules.



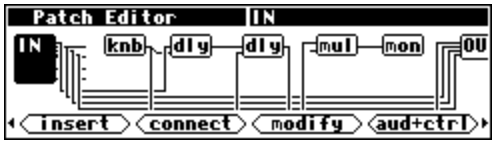
If you press the <aud only> SOFT KEY to go to the aud+ctrl display mode, you'll see the knobs and monitor that make the Each delay and Delay Amount parameters shown in the PARAMETER area work. Notice the mul module. That's a c_multiply module. It is multiplying the control signal from the knob (knb) module by a constant amount *(in this case 2 -but you can't see it in this display)* and feeding the result to the monitor (mon) module.



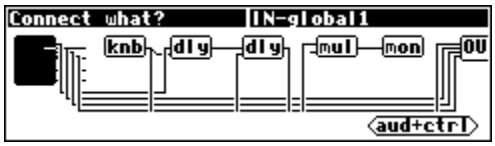
Go back to the PARAMETER area *(by pressing and holding the PARAMETER key)* to see how these modules and their connections in the Patch Editor area translate into parameters on a menu page. The monitor module creates the parameter Delay Amount that shows the actual delay, while the knob creates the parameter Each delay that sets the delay for each of the delay modules.

For the sake of demonstration, we'll use the <connect> SOFT KEY to rearrange the delay (dly) modules such that one is in "channel" 1 and the other is in "channel" 2, thus delaying each "channel" by up to 10 seconds (10,000mS). *(Hey! Ya gotta crawl before you can run, OK?)*

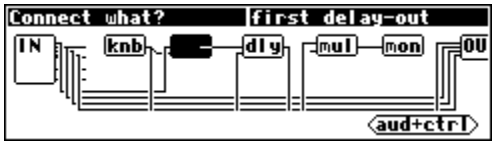
Press the PROGRAM key and reload Patch Instruct. After the program is loaded, press and hold the PARAMETER key to re-enter the Patch Editor area.



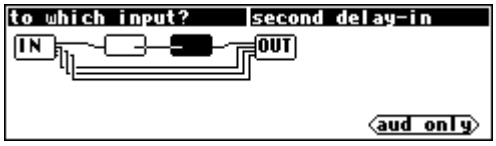
Press the <aud only> SOFT KEY to get the screen shown to the right.



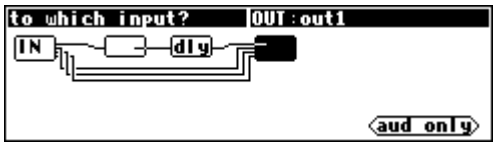
Now press the <connect> SOFT KEY. The Harmonizer will prompt for something to connect.



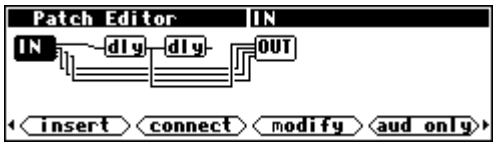
Using the CURSOR keys, select the output of the leftmost dly module. You'll know you have the correct output when the upper right of the display reads first delay-out. When the screen looks like the one shown to the right, press the SELECT key.



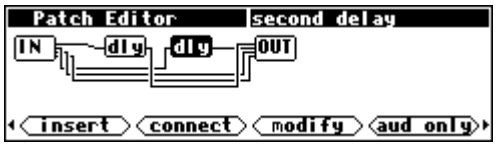
As this is an *audio* output, the screen mode will automatically change to `aud only` mode. Because you have selected an *output*, the screen will now prompt you for an *input* to connect to. The Patch Editor will automatically choose a valid audio input for you to connect to as shown to the right. Note that the Patch Editor's choice may not be your choice! Use the LEFT and RIGHT CURSOR keys and the KNOB to experiment with what exactly can be selected.



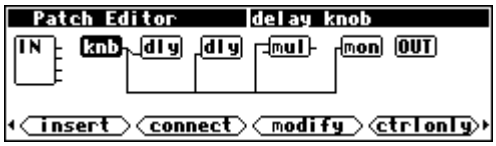
Although the <aud only> SOFT KEY is presented, pressing it in this case will *not* allow connections to anything that *isn't* an audio or mod input. This is because we've already selected an *audio* output, and you can't connect an *audio* output to anything but an *audio* or *mod* input. Feel free to press the <aud only> SOFT KEY, and you'll see what we mean.



After you've experimented, set the screen mode back to `aud only`, select `out1` on the `OUT` module, and press the SELECT key.



The current program now has a single delay module between `IN`:`in1` and `OUT`:`out1`, a `delay` module that has an input but no output, and straight paths between the remaining DSP "channels." Note that the signal that was *previously* connected to `OUT`:`out1` has been automatically disconnected. See Notes below.



You should now be able to connect `IN`:`in2` to `second delay`:`in`. Then connect `second delay`:`out` to `OUT`:`out2`. The screen to the right is what you should end up with. This "patch" has a delay module in each "channel" of audio.

As an exercise, you could go to `ctrlonly` screen mode and bypass the `mul` module by connecting the `knob` (`knb`) to `mon-in` *(as shown to the right)*. That would make the `Delay Amount` parameter in the PARAMETER area show the correct delay value.

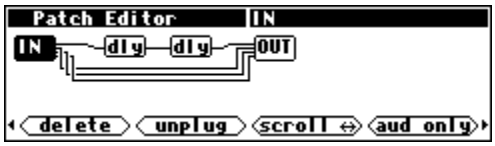
Notes on <connect>

Although it is possible to connect a single output to multiple inputs, it is not possible to connect two signals to a single input. If an attempt is made to connect a signal to an input that is already in use, the new signal will replace the old. To connect multiple audio signals to one input, a `mixer` or `adder` module could be used to combine the audio signals. For control signals, a `c_adder` module could be used.

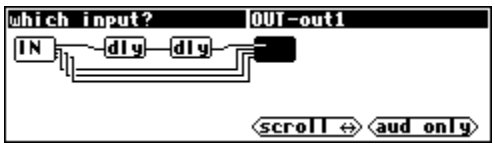
Breaking a Connection <unplug>

The <unplug> SOFT KEY removes a single connection between two modules. To break a connection, press the <unplug> SOFT KEY, then use the CURSOR keys and the KNOB to choose which input to disconnect. The <unplug> SOFT KEY will not allow a disconnect to be specified by output because outputs may be connected to more than one input.

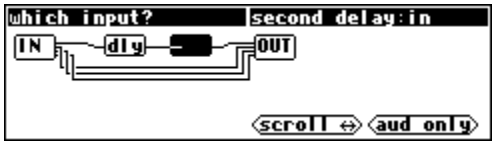
Example: To break the connection between the two `delay` modules in the program `Patch Instruct` from the "`Programming`" bank, first load the program. Then press and hold the PARAMETER key to enter the Patch Editor area.



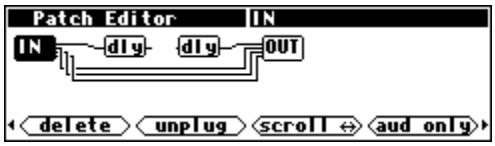
After the display updates, briefly press the PARAMETER key again to reveal the alternative set of SOFT KEYS.



Press the <unplug> SOFT KEY



Use the LEFT CURSOR key to select the input for the second `delay` module.



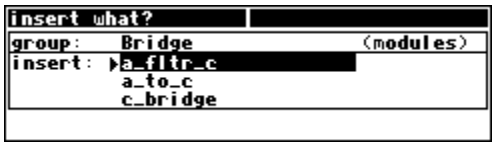
Press the SELECT key.

That's it. You've "unplugged" the input to the `delay` module.

Note that the `aud only/aud+ctrl/ctrlonly/misc` SOFT KEY is active to aid in selecting a module and input to be unplugged. Changing the display mode does not deselect the currently selected input. The upper right corner of the screen will indicate the currently selected input, regardless of the display mode.

Inserting Modules <insert>

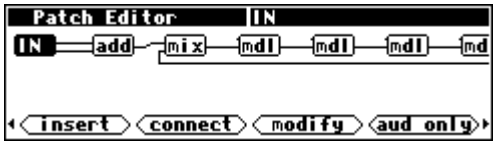
The <insert> SOFT KEY adds a new module to the "patch." The new module will be inserted to the right of the currently selected module. Use the RIGHT and LEFT CURSOR keys to select the insertion point.



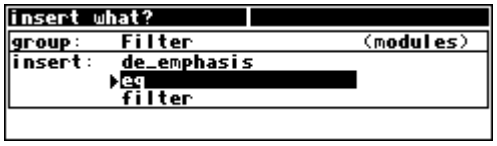
Note that modules do not, strictly speaking, have to be in any particular order because connections can run in either direction. However, the programmer should be aware that every instance of reverse signal flow will add a four sample delay to the process. In some casess, such as where a preset has multiple signal paths, such delays can cause objectionable "phasing," or other artefacts.

After you've chosen where to place a new module, press the <insert> SOFT KEY. This display is a lot like the one given in the PROGRAM area. Just as in the PROGRAM area, the top area shows the name of a "group" (bank), and the field below lists the contents of that group. The difference is that here a group "contains" modules instead of programs. *If you're unfamiliar with how to load a program, you probably shouldn't be reading this manual. Please read the separate User's Manual before proceeding!*

Most modules will create a menu page in the PARAMETER area when they are inserted in the Patch Editor area. To demonstrate this, first load the program `Long Mono Delay` from Bank 2. Then press and hold the PARAMETER key to enter the Patch Editor area.



Insert an `eq` module by pressing the `<insert>` SOFT KEY, scrolling to the Filter group, and selecting `eq`. Press the SELECT key to actually insert the `eq` module.



Press and hold the PARAMETER key to return to the PARAMETER area. Note that now there is a menu page and an associated SOFT KEY for `eq` parameters.

Most, but not all, modules get their own PARAMETER area menu keys automatically. If the inserted module comes from any of the following groups, it will *not* show up automatically:

- `Bridge`
- `Control Math`
- `Control Process`
- `Interface`
- `Math`
- `Miscellaneous`

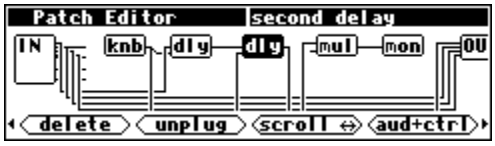
If a module does not create an "automatic" menu page in the PARAMETER area upon insertion but does *have parameters, a menu page can still be created for it in the PARAMETER area. This is a more advanced operation and is discussed in Chapter 3.*

Notes on `<insert>`

During the insert process, if you change your mind and decide not to insert anything yet, press the PARAMETER key to abort. That will put you back at the main edit menu without changing the patch.

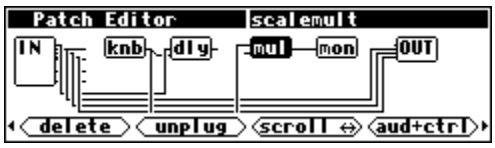
Removing a module *`<delete>`*

The `<delete>` SOFT KEY removes a module from a program. Any signals connected to the deleted module are disconnected. Select a module to be deleted by highlighting it using the LEFT and RIGHT CURSOR keys. Here we're choosing the `second delay` `delay` module.



Next, press the `<delete>` SOFT KEY. The Harmonizer prompts to be sure that the `<delete>` is intentional. If it is, press the DOWN CURSOR key (choosing OK) and then press the SELECT key.

In the example to the right, the `OUT` module now has an unconnected `input` and the first `dly` module has an unconnected `output`.



Modifying a module *`<modify>`*

The `<modify>` SOFT KEY is described in the next section.

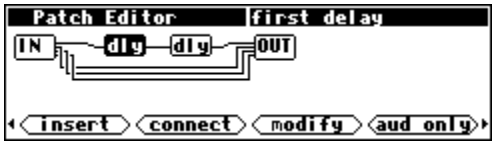
THE <MODIFY> SOFT KEY

The <modify> SOFT KEY gives you the ability to directly change the "internals" of a module. This is necessary to create complex, highly customized programs. The <modify> SOFT KEY works on one module at a time and is needed to change the following:

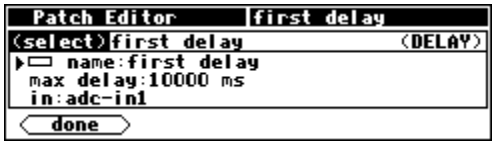
- the module name
- *specifier*s
- connections between *userobject*s and *userobject* inputs
- control inputs that are not "patched" and that are not controlled via their *userobject*

Modifying a delay module

To use the <modify> SOFT KEY in the Patch Editor display, simply select the desired module using the LEFT and RIGHT CURSOR control keys. . .

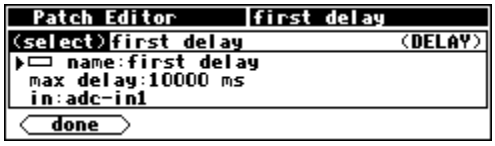


. . .and then press the <modify> SOFT KEY. On entry to the <modify> menu, the display shows the current KNOB mode (select or adjust), the name of the module, the module type, and the first three lines of module information.

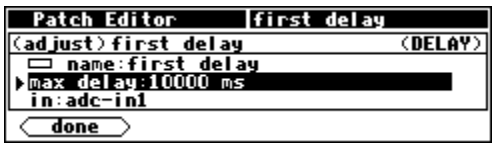


The <modify> menu scrolls and behaves much like things in the PROGRAM area do.

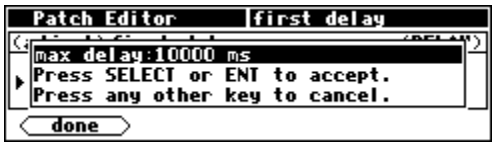
- To scroll through the data for the module use the CURSOR keys. If the KNOB mode reads (select) *(as shown to the right)* you can also use the KNOB to scroll through the data for the module.



- To change any of the data in the module, first ensure that the line of data you want to change is highlighted and that the KNOB mode reads (adjust) *(these two requirements are actually one in the same)*. You highlight a line by either pressing the RIGHT or DOWN CURSOR key OR by pressing the SELECT key.



- Once a line of data is highlighted, spin the KNOB or use the numeric keypad to adjust the data. A pop-up menu appears that prompts you to either press the SELECT key or the ENT key when you are satisfied with the change.



Below is a composite of the information for the first delay module in the Patch Instruct program.

Composite illustration of <modify> on a delay module

The above example includes several details that are familiar and a few that aren't. The following is a breakdown of each line.

KNOB Mode

If the KNOB mode reads (select), spinning the KNOB will scroll through the menu. If the KNOB mode reads (adjust), spinning the KNOB will adjust the data on the current line *(the line with the little triangle next to it!)*.

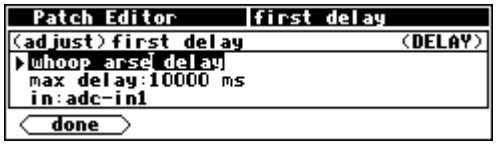
Module Name

The name of the module we are <modify>ing!

Module Type

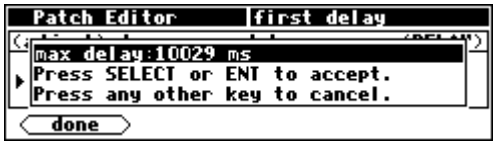
The type of module we are <modify>ing!

Edit Module Name



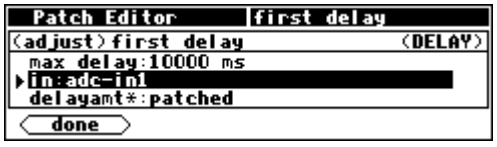
This is the current name of the selected module. Selecting this line and pressing the SELECT key will allow you to change the name. After the name is changed, press the SELECT key to make it "stick" . The Harmonizer will display a message indicating that "Modifying…" and then "Loading new patch…" is taking place. If the <done> SOFT KEY is pressed *while* the name is being edited, the name change will be lost and the display will return to the basic Patch Editor area display.

Specifier

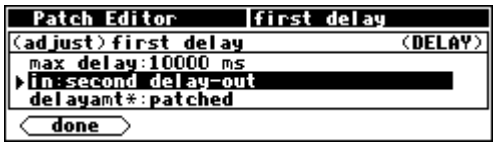


This example (the delay module) has only one *specifier*. Some modules have many *specifier*s. To change the *specifier*, choose it and press the SELECT key. This will bring up a menu. Change the value with the numeric keypad or the KNOB. Press the SELECT key or the ENT key to save your alteration. The Harmonizer will display a message indicating that "Modifying…" and then "Loading new patch…" is taking place. If the change in *specifier*s makes the program take up too much of any resource, the Harmonizer will display the "Patch too big" error message and will reverse the change. To return to the <modify> menu *without* changing the value of the *specifier*, press any key other than SELECT or ENT.

Audio input



The delay module has one audio input. In this program, the first delay module's input is connected to adc-in1. That's "techie speak" for in1 on the IN module.

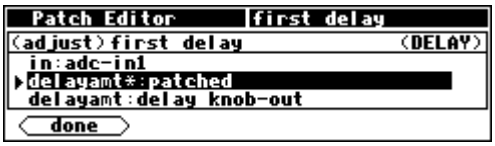


You can change the output that connects to the current module's input in the <modify> menu if you so desire. Here we've changed it to second delay-out.

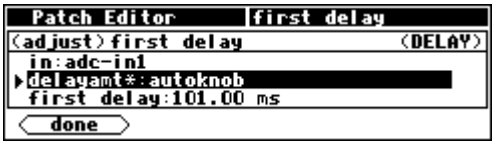
Returning to the basic Patch Editor area display (*by pressing the <done>* SOFT KEY), we can see that the second `delay` module's `output` is indeed connected to the first `delay` module's `input`.

"Fine," you say, "but it seems simpler to do stuff like that with the <connect> SOFT KEY." A wise pupil are you. . .

Mode of 'Delayamt' Control Input



Every control input has two possible modes, "patched" and "autoknob." If patched is selected *(as in the example to the right)* the next line of the menu will show the module and output connected to *this* input *(`delay knob-out` in the example to the right)*. More on this below under "Output Connected to 'delayamt' Control Input." . .



If "autoknob" is chosen *(as in the example to the right)*, the next line of the menu will show the "autoknob's" PARAMETER area menu statement along with the current value of the control input *(`first delay: 101.00 ms` in the example to the right)*. More on this below under "Autoknob." . .

Output connected to 'delayamt' control input

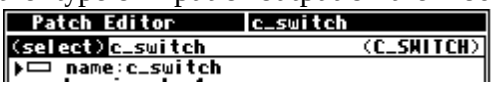
Since the mode of the `delayamt` control input is set to "patched" in the example shown to the right, the *next* line will show the module and output connected to *this* input. By selecting this line *(as shown to the right)* and pressing the SELECT key, the chosen module and output may be changed. But of course the standard method of re-patching control inputs is to use the <connect> SOFT KEY in the basic Patch Editor area display.

Autoknob

If this particular module's `delayamt` input is set to "autoknob." the control input's value is adjustable. The prompt offered (in this case "`first delay`") is the same prompt that would be offered if this module's *userobject* was displayed on a menu page in the PARAMETER area *(by connecting this module's userobject to the head module)*. The prompt, also called a "menu statement." is, in this case, the same as the module name. The text that is displayed is determined by the inherent properties of a particular module type (i.e. `delay` module) and may be different for other module types.

Modifying Complex Modules

Some modules have *specifier*s that change the number of remaining *specifier*s in the module or the number of some other type of input or output on the module. Consider these two composite screen images:

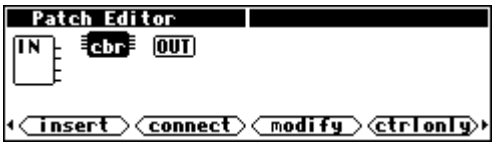


The major difference between these two examples of the `c_switch` module is that the module on the left has its `number inputs`: *specifier* set to `1` whereas the module on the right has its `number inputs`: *specifier* set to `4`. Since the `c_switch` module will always have its `number inputs`: *specifier* set to `1` when it is first inserted, the `<modify>` menu must be used to enable more inputs. Note that since the number of control inputs in the `c_switch` module has changed, basic Patch Editor area display will show a different icon for the module:

1 input `c_switch`: 

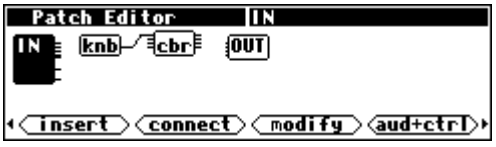
4 input `c_switch`: 

INTER-DSP COMMUNICATION FOR ORVILLE

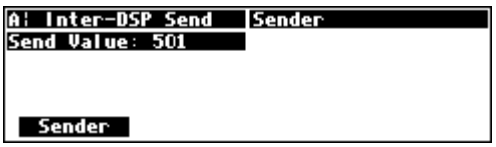


Control signals can be sent from one DSP to the other in Orville. *7000 family users should note that they only have a single DSP, so will probably want to skip this section.*

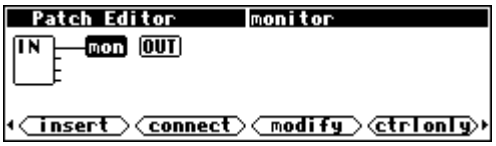
The `c_bridge` module accepts four control signal inputs. Control signal outputs that are connected to these inputs appear at the *other* DSP's "global control outputs" and at the control *outputs* of a `c_bridge` module in the *other* DSP. A DSP's global control outputs are located on the IN module as seen on the screen to the right *(notice that we're in `ctrlonly` display mode).*



For example, load the program `Inter-DSP Send` from the "`Programming`" bank into DSP A. A `knob` module is connected to the first input of a `c_bridge` module in as shown to the right.

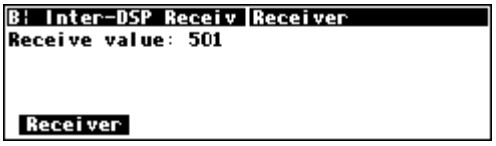


Additionally, the `knob` module's *userobject* output is connected to the `head` module so that the `knob` module's parameter shows up in the PARAMETER area as shown to the right.



Now load the program `Inter-DSP Receive` from the "`Programming`" bank into DSP B. A `monitor` module is connected to global control output 1.

Additionally, the `monitor` module's *userobject* output is connected to the `head` module so that it shows up in the PARAMETER area as shown to the right.

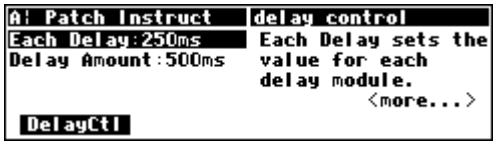


You can see for yourself that changing the `Send Value` in DSP A alters the `Receive value` in DSP B. Of course, you could <insert> a `c_bridge` module in DSP B to send control signals to DSP A *at the same time that DSP A is sending control signals to DSP B!*

CREATING THE USER INTERFACE

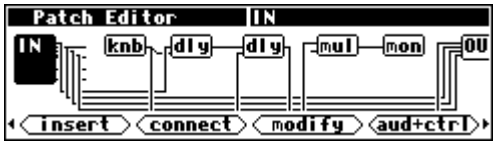
The Patch Editor automatically creates PARAMETER area menu pages for most modules when they are inserted by connecting their *userobject*s to the head module. You can create a wealth of programs this way. In the PARAMETER area, parameters will automatically be grouped by module and SOFT KEYS will appear - one per module. A program created this way will be fully functional and have all of the audio characteristics of a factory preset. Audio, however, is where the similarities end. A program created using the "automatic menu system" will not look as slick, nor be as easy to use, as the factory presets are. Factory presets are created by hand-connecting the *userobject*s of knob modules to menupage modules and then hand-connecting the *userobject*s of those menupage modules to the head module. This chapter details how this is done.

Viewing Menupages and Menupage Modules

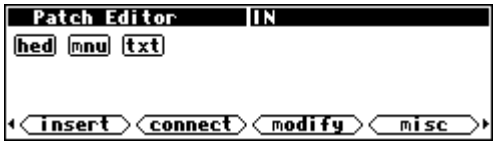


A menupage module has a single *userobject* and any number of *userobject* inputs. Normally a menupage module is connected to the head module. If so, the menupage module shows up in the PARAMETER area as one or more pages of parameters, a title line, and a SOFT KEY. The information for the title line and SOFT KEY and the list of connected *userobject*s that comprise the parameters seen in the PARAMETER area are accessible by using the <modify> SOFT KEY on the menupage module. For example, load the program Patch Instruct from the "Programming" bank.

The menu page shown above is visible in the PARAMETER area. It is created with a menupage module, a knob module, a monitor module, and a textblock module *(we'll see how in a minute)*.

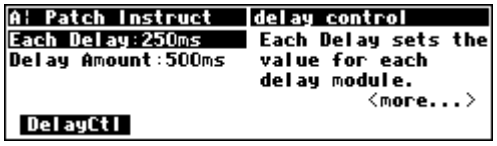


Pressing and holding the PARAMETER key will access the Patch Editor area. Do so and then press the <aud only> SOFT KEY to get the screen shown to the right. From this display the knob (knb) module and monitor (mon) module are visible. Both are visible in this display mode because they have one or more control inputs or outputs. The menupage and textblock modules have neither so they will only be visible in the misc display mode. Press the <aud+ctrl> SOFT KEY twice to view the program in the misc display mode.



The three modules, head (hed), menupage (mnu), and textblock (txt) can now be seen. Pressing the LEFT or RIGHT CURSOR key will select one of the modules. Select the menupage module and then press the <modify> SOFT KEY.

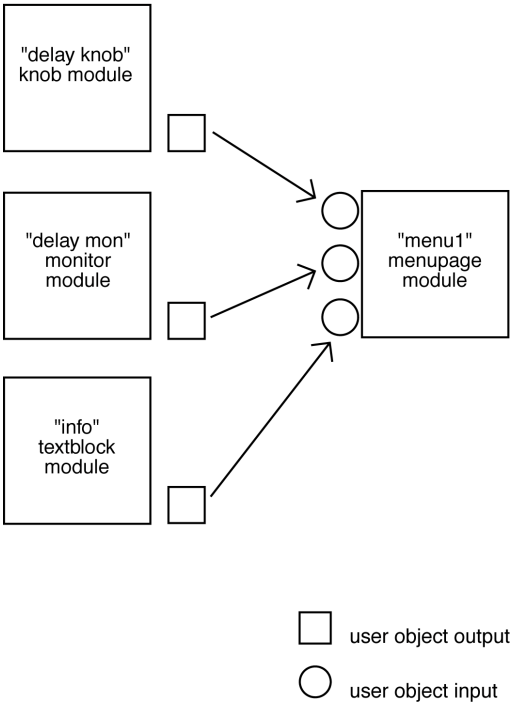
From here, we can see the description "delay control" and the 8 char name "DelayCtl."



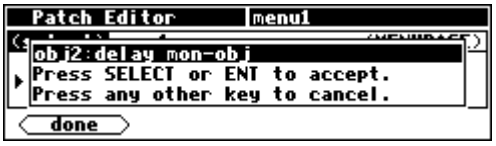
Note that these equate to the title and SOFT KEY when the menu page is viewed in the PARAMETER area.

To the right is a composite of the <modify> menu data that would be seen by rotating the KNOB. The obj data lines indicate which *userobject* outputs are connected to this menupage module's *userobject* inputs. In this case, the *userobject*s of the module named "delay knob." the module named "delay mon." and the module named "info" are all connected to this menupage module's *userobject* inputs. are included *userobject*s.

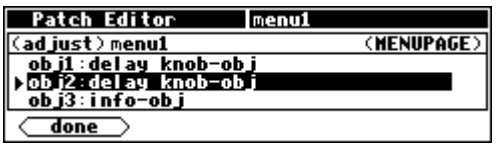
To the right is a diagram of what's actually going on. As was mentioned before, the connections made between *userobject* outputs and *userobject* inputs are *not* shown as little lines in the Patch Editor area. The connections are implicit, much like the connections made between inputs and outputs when routing signal flow on the [analog], [dsp A], [dsp B], and [digital] menu pages in the SETUP area of Orville.



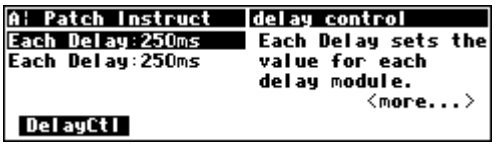
As an exercise, use the KNOB and RIGHT CURSOR key to select obj2. Press the SELECT key. The screen should look like the one to the right. Rotate the KNOB left and right to view available *userobject*s that can be connected to this *userobject* input. The possibilities are: adc-nullobj, delay knob-obj, scalemult-obj, delay mon-obj, menu1-obj, and info-obj.



Just for the fun of it *(and what fun it is!)* select delay knob-obj. The screen should look like the one to the right, with delay knob-obj connected to both *userobject* input 1 and *userobject* input 2.



Now return to the PARAMETER area by pressing the <done> SOFT KEY and then pressing and holding the PARAMETER key. You should see that indeed, the Each Delay parameter is now doubled on the menu page. Change one version and then highlight the second version; you'll see that they are the same even if they do exhibit some peculiar behavior. You won't normally have any reason to put the same parameter on the same menu page more than once! *(But you may want to put the same parameter on* different *menu pages in the same program so that it will be accessible from more than one "place" in the program.)*

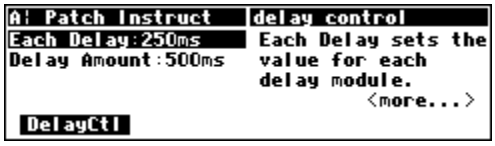


Interface Modules

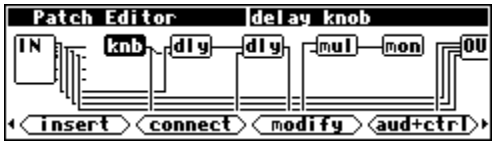
Control inputs are used to send a parameter value into a module. The parameter value is generated by another module, perhaps a knob module. One common use for this capability is the creation of custom "parameter adjusters" to adjust the parameters for the modules in a program. The custom "parameter adjusters" are special purpose modules from the "interface" module group. This group includes the common text/numerical parameter adjuster that is generated by the knob module, as well as several graphical "parameter adjusters" (hfader module, vfader module, and rfader module).

PARAMETER ADJUSTERS

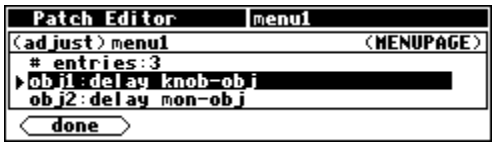
"Parameter adjuster" modules have a single control output and a *userobject* output. If connected to a menupage module, a "parameter adjuster" will show up on the menu page in the PARAMETER area as a parameter.



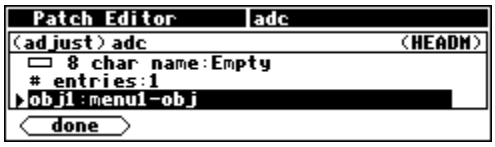
For example, in the now infamous program Patch Instruct shown to the right, the module named "delay knob" is a "parameter adjuster."



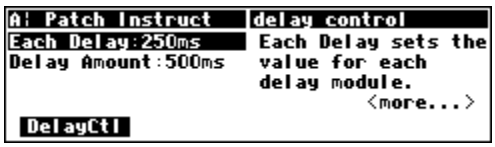
Its *userobject* is connected to a menupage module. . .



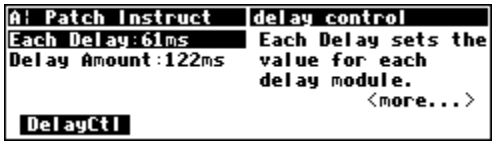
. . .which is in turn connected to the head module.



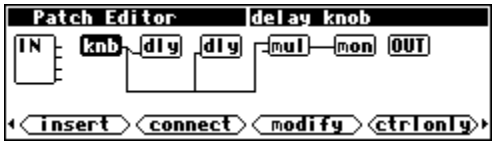
Thus, the module named "delay knob" shows up in the PARAMETER area as a parameter (Each Delay).



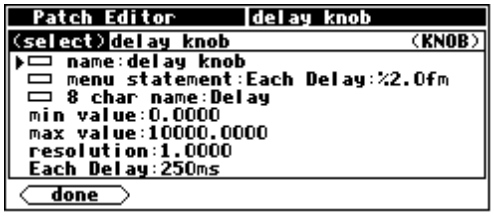
Selecting a parameter in the PARAMETER area and rotating the KNOB will change the value of the "parameter adjuster's" control output. The change will also be reflected in a textual or graphical display change. In this case, rotating the KNOB changes the Each Delay parameter in the PARAMETER area. . .

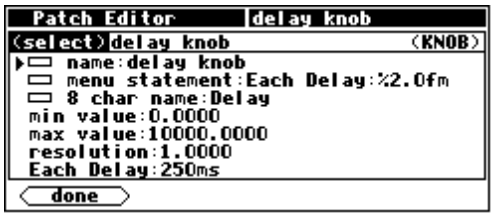


. . .and it changes the value sent from the "delay knob's" control output into both dly modules and the mul module (of course this screen doesn't *show* the change, but the value has changed nonetheless!) .



The actual text used for a parameter and the way changes made to a parameter's value in the PARAMETER area translate to control output changes in the Patch Editor area are set up by <modify>ing the "parameter adjuster's" module in the Patch Editor area.. This screen shown to the right is a composite picture of the <modify> menu for the knob module in the Patch Instruct program.





- The module name *specifier* is set to "delay knob."
- The menu statement *specifier* is set to "Each Delay:%2.0fms" *(the last 's' is hidden)*. This means that on a menu page in the PARAMETER area where this knob module's menu statement shows up, the text "Each Delay: xxxxxms" will show, where xxxxx actually reflects the value that the parameter is set to. The "%2.0f" part is described in some detail later.
- The 8 char name is "Delay." The 8 char name is what would show up as a SOFT KEY if this module's *userobject* were connected to the head module.
- The min value *specifier* sets the minimum value that the parameter can be set to.
- The max value *specifier* sets the maximum value that the parameter can be set to.
- The resolution *specifier* sets the "jump" that the parameter value makes when the KNOB is rotated. In other words, when the user rotates the KNOB, the resolution is how far the parameter value changes per incremental movement.
- The last line in the <modify> menu shows "Each Delay:250ms" This is called the "example line." It is an example of what the menu statement actually looks like when viewed in the PARAMETER area. If the example line is selected, the parameter value can be set and the parameter will behave the same as it does when used in the PARAMETER area.

Let's look at these lines in more detail, shall we?

Menu Statement

The menu statement is a crucial *specifier* used in the basic knob module, which is the most common "parameter adjuster." The menu statement is the line that will appear in PARAMETER area menu page. The menu statement may contain up to 20 characters *including* the parameter value. Anything over 20 characters will not be displayed.

The first job of the menu statement is to indicate to the user what the parameter is for. It should also contain the *format* for the parameter value that will be displayed, indicating the number of spaces that the parameter value will take up and how many digits will be after the decimal point for a numerical parameter value *(parameter values can be text as well, more on this later)*. You must specify this format bearing in mind the min value, the max value, and the resolution.
The syntax of the format is:

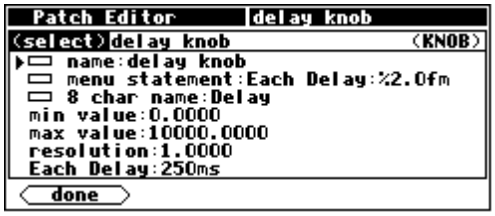
%Y.Xf

where Y is the number of spaces reserved for display and X is the maximum number of digits after the decimal point. The percent(%), period(.), and f must be used as shown. If the period(.) is removed, the

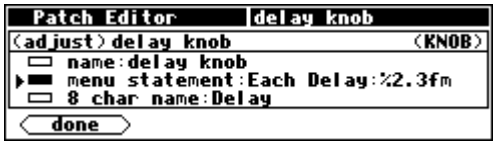
Harmonizer will display six digits after the decimal point. Here are example formats and results that would be displayed on a menu page in the PARAMETER area. "~" represents a space that will be inserted.

format	for 1.2345	for 23456.0013	for .1234	for 1	for -55.234
%1.2f	1.23	23456.00	0.12	1.00	-55.23
%4.2f	1.23	23456.00	0.12	1.00	-55.23
%5.2f	~1.23	23456.00	~0.12	~1.00	-55.23
%5.0f	~~~~1	23456	~~~~0	~~~~1	-55.23
%7.1f	~~~~1.2	23456.0	~~~~0.1	~~~~1.0	~-55.23
%9.4f	~~~1.2345	23456.2345	~~~0.1234	~~~1.0000	~-55.2300
%2f	1.234497	23456.001300	0.123398	1.000000	-55.234000

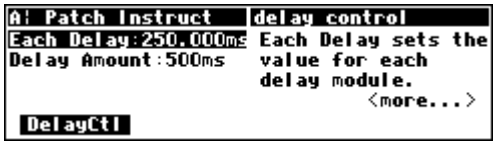
Refer to the separate ***User Manual*** on entering text for a list of the characters included in the text insert menus. The formats shown here can be created using the % character, numbers, a small f, and a period(.).



As an wee exercise, load the Patch Instruct program from the "Programming" bank, go to Patch Editor area, change the screen mode to aud+ctrl, and highlight the knb module. Then use the <modify> SOFT KEY to change the menu statement to "Each Delay:%2.3fms" as shown to the right.

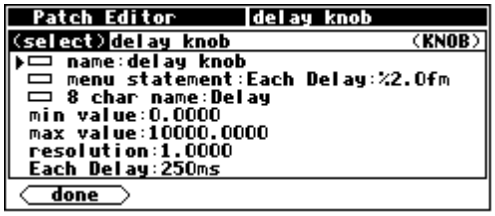


Now go to the PARAMETER area and see the difference that made to the display. Note the decimal value in the "Each Delay" parameter. It used to read 250. Now it reads 250.000. *Contain yourself. . .*

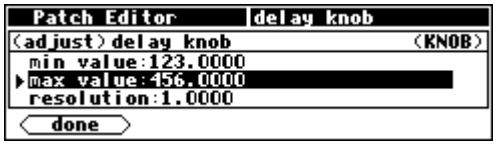


Min and Max Values

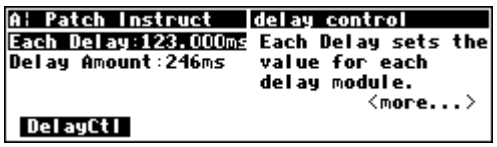
The lower and upper limits of a numeric parameter value are set as *specifier*s in the module that controls the parameter. In the example program Patch Instruct, the "delay knob" parameter value has a range of 0.0000 to 10000.0000 set by the min value and max value *specifier*s. You can adjust these limits using the numeric keypad or the KNOB.



As another wee exercise, load the program Patch Instruct from the "Programming" bank, go to the Patch Editor area, change the screen mode to aud+ctrl, and highlight the knb module. Then use the <modify> SOFT KEY to change the min value and max value to set different limits as shown to the right.

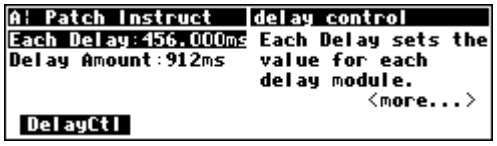


Now go to the PARAMETER area and test the Each Delay parameter. To the right we bump up against the new min value.



And on this screen we bump up against the new max value.

You will find max value and min value *specifier*s in most of the "interface" group modules.



Resolution

The resolution *specifier* controls what minimum change in a parameter value can be achieved by turning the KNOB or by using the INC or DEC key on the numeric keypad. The resolution parameter also controls the "rate of change" as the KNOB is spun. If the resolution is very *fine*, the parameter value will increment in "baby steps" as the KNOB is spun. If the resolution is very *course*, the parameter value will increment in "great leaps" as the KNOB is spun.

For example, if the selected parameter displays a value of 45.30 and the resolution is 1.0000, then slow motion clockwise rotation on the KNOB will change the value to 46.30 *(unless the max value is less than 46.30!)*. If the selected parameter displays a value of 45.30 and the resolution is 0.1000, then slow motion clockwise rotation on the KNOB will change the value to 45.40.

As yet another wee exercise, use the <modify> SOFT KEY the same way as in the Min and Max Value section and adjust the resolution *specifier* of the knb module. Notice the difference this makes in the "step size" of the Each Delay parameter value. *(You'll have to walk yourself through this one. . .)*

Simple "Parameter Adjusters"

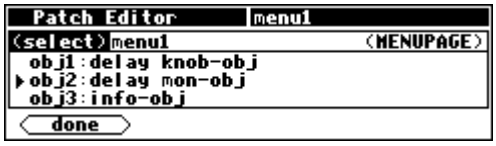
There are seven simple "parameter adjusters" modules:

- A `knob` module for simple numeric values.
- An `rfader` module for "rotary" graphical display instead of numeric values.
- An `hfader` module for "horizontal" graphical display instead of numeric values.
- A `vfader` module for "vertical" graphical display instead of numeric values.
- A `textknob` module for "text-valued" parameters.
- A `tapknob` module for "tapered" (or non-linear) values.
- A `percentknob` module for *percentage* display that corresponds to *fractional* control output.

They have several things in common:

- All have a single control output and no other signal inputs or outputs.
- All have a *userobject* output that can be connected to a `menupage`, `gang`, or `head` module.
- All have a `menu statement` and an `8 char name`.

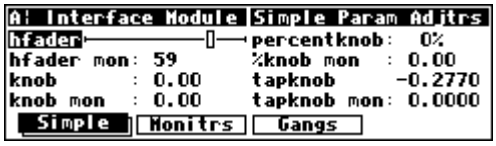
These modules are used by connecting their *userobject* outputs to a `menupage` module *(using the <modify>* SOFT KEY *on the menupage module as shown to the right)* or the `head` module *(using the <modify>* SOFT KEY *on the head module)*. The PARAMETER area menu pages will then show the text or graphic `menu statements` for the connected "parameter adjuster" modules.



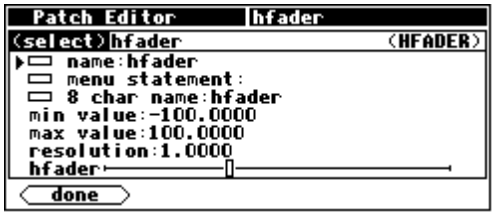
The following pages describe the simple "parameter adjusters." To play along at home, load the program `Interface Modules` from the "`Programming`" bank.

Hfader Module

The `hfader` module creates a horizontal graphic on a PARAMETER area menu page. In the example screen to the right it is the highlighted, upper left parameter. The area taken up by the graphic is one half of the width of the screen and one text line long. Eight of these can fit on a single menu page.

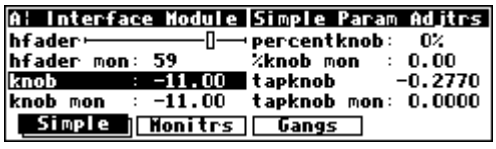


Six characters of the `8 char name` are presented on the display to the left of the graphic. The `menu statement` is not used. Refer to the ***Modules Section*** for complete information.

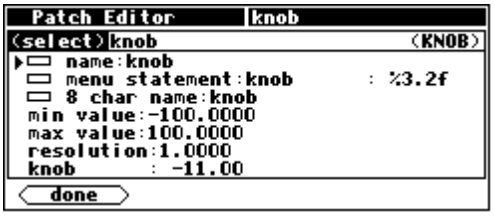


Knob module

The `knob` module is the most popular interface module (in factory programs). It generates a 20-character text string, including a very versatile numerical display from the `menu statement`. Eight of these can fit on a single menu page.

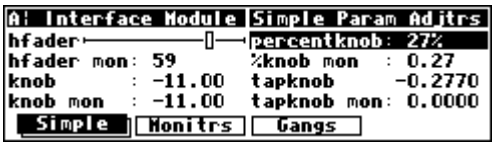


The `8 char name` is used only if this module's *userobject* is connected directly to the `head` module. Normally its *userobject* will be connected a `menupage` module. Refer to the ***Modules Section*** for complete information.



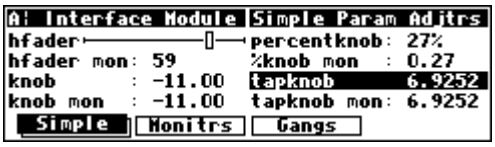
Percentknob Module

The `percentknob` module is very similar to the `knob` module. The only difference is that the control output value is divided by 100. Refer to the ***Modules Section*** for complete information.

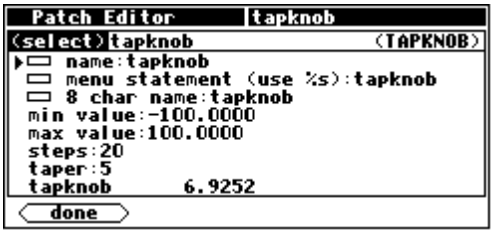


Tapknob Module

The `tapknob` module is a modification of the standard `knob` module. Just like the `knob` module, the `menu statement` is used to create the 20-character text display. However, instead of using the `%f` format, the `%s` format is used. The `tapknob` module creates an 8-character numeric result that is inserted in place of the `%s`.

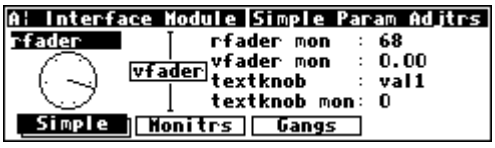


The `tapknob` module creates a tapered (non-linear) control that has a "selectable" number of `steps` (instead of the usual `resolution` parameter) and a "selectable" `taper` waveform. The greater the `taper` *specifier*, the more non-linear the parameter response. Refer to the ***Modules Section*** for complete information.

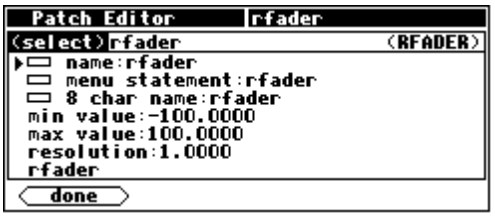


Rfader Module

The `rfader` module creates a graphic on the PARAMETER screen. The graphic, including title, takes up four lines of the screen and one quarter of the width of the screen. Up to four of these modules can fit on one menu page.



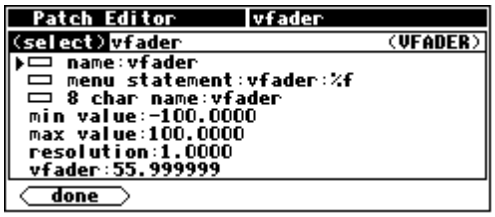
Nine characters of the menu statement are displayed above the graphic as a title. Refer to the ***Modules Section*** for complete information.



Vfader Module

The `vfader` module creates a graphic on the PARAMETER screen. The graphic, including title, takes up four lines of the screen and one sixth of the width of the screen. Up to six of these modules can fit on one menu page.

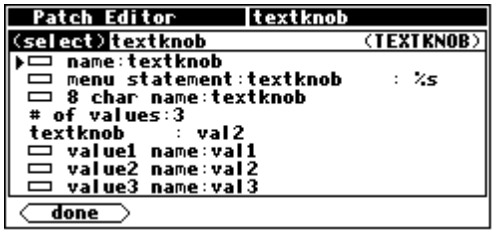
Six characters of the `8 char name` are displayed in the graphic as a title. Refer to the ***Modules Section*** for complete information.



Textknob Module

The `textknob` module creates a multiple choice selection in a single line by half screen width area of a PARAMETER area menu page.

The choices appear in place of the `%s` in the `menu statement`. The `8 char name` is only used if the module's *userobject* is connected to the `head` module. The control output reflects which selection is made. If the 1st selection is made the output will equal 0. If the 3rd selection is made, the output value will equal 2. Refer to the ***Modules Section*** for complete information.

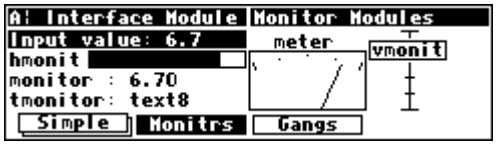


CONTROL SIGNAL MONITORS

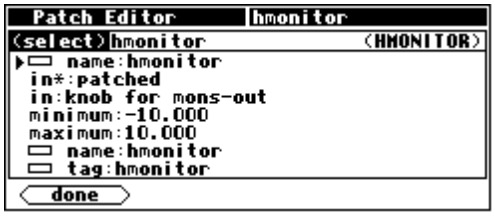
Just as "parameter adjuster" modules are used to *generate* control signals and are displayed as parameters in PARAMETER area menu pages, control signal monitor modules *monitor* the value of control signals and may be displayed on those same menu pages.

Graphical Control Signal Monitors

There are five different monitor modules. Three of these, the `hmonitor`, `meter`, and `vmonitor` modules, produce graphical displays. The screen to the right *(taken from the program `Interface Modules` from the "`Programming`" bank)* shows the three graphical monitors *(among others)*.



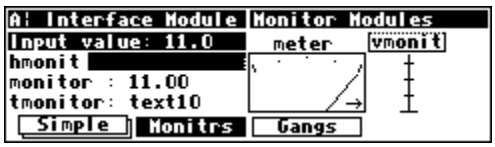
Each of the graphical control signal monitor modules has a control signal input and four *specifier*s: `minimum`, `maximum`, `name` and `tag`.



- `minimum` sets the lowest value that may be indicated by the monitor.
- `maximum` sets the highest value that may be indicated by the monitor.
- For `vmonitor` and `hmon`, the `tag` *specifier* is used to generate the text for the monitor.

- For `meter`, the `name` *specifier* is used to generate the text for the monitor.
- The text fields of the monitors may include `%f` format numeric displays *(to convey numerical as well as graphical information)*.

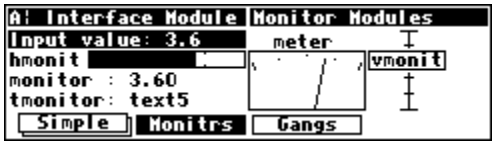
All three graphical monitors will indicate when the control input's value falls above or below the range set by the `minimum` and `maximum` *specifiers*. *(The screen to the right was made by changing the `max value` to `11` on the "`knob for mons`" module in the `Interface Modules` program.)*



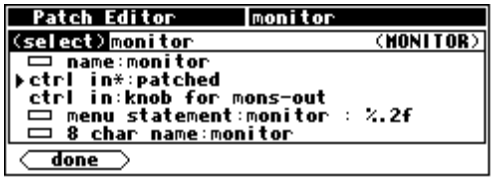
The `vmonitor` module creates a graphic that is one sixth of a screen width and four lines high. The `hmonitor` module creates a graphic that is one half of a screen width and one line high. The `meter` module creates a graphic that is one quarter of a screen width and four lines high. *Form over function. . .*

Textual Control Signal Monitors

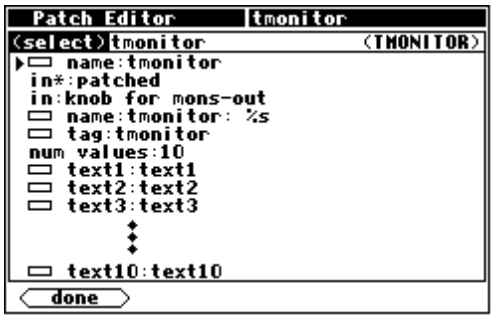
The `monitor` and `tmonitor` modules use text to display their control input values.



The `monitor` module is a mirror image of the `knob` module; it displays the decimal value of its control input. The format for the display is set using the text and `%f` format described earlier.



The `tmonitor` module is a mirror image of the `textknob` module. It uses the control input to determine which of several text strings will be shown. A control input value of 0 chooses `text1`, a value of 3 chooses `text4` and so on.

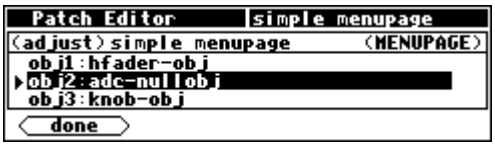


Both the textual monitor modules create displays that are half of a screen width and one line long.

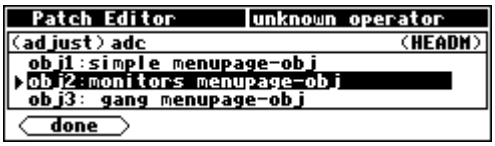
Menupages and Parameter Placement

The use of menupage modules to create menu pages in the PARAMETER area is crucial for creating easy to use programs. This section discusses many of the fine points of menu creation and the care and feeding of *userobject*s. There are several important points regarding PARAMETER area menu pages and their construction in the Patch Editor area:

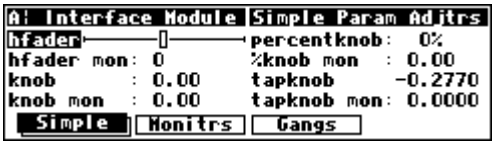
- Null *userobject*s are invisible. They do not take space on a menu page in the PARAMETER area. Having the head module or a menupage module include a nullobj (sometimes titled adc-nullobj *(as shown to the right)* or head-nullobj) creates null *userobject*s.



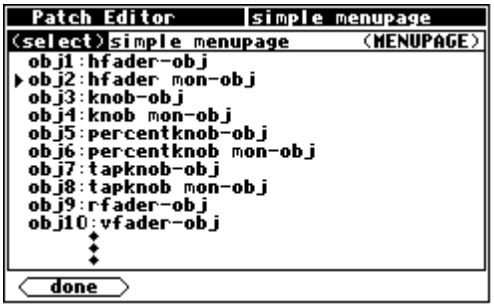
- Any *userobject* that is connected to the head module creates a SOFT KEY (with the exceptions of those *userobject*s that are null).



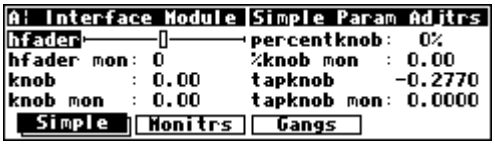
- The order that a *userobject* is listed in the head module determines what location the SOFT KEY will appear in the PARAMETER menu. The first *userobject* gets the first SOFT KEY. The fifth *userobject* gets the fifth SOFT KEY and so on.



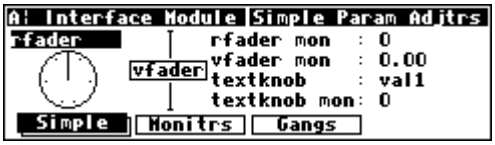
- The order that a *userobject* is listed in a menupage module determines where on a PARAMETER area menu page it will appear.



- Objects are placed on a menu in upper left to lower right order, as listed in the menupage module.



- If a module's PARAMETER area graphic is too large to appear on a menu page with other modules' graphics, it is placed on a later menu page in a menu stack, thus creating a SOFT KEY stack.



- menupage modules may be connected to other menupage modules! A menupage *userobject* output is the same as any other module's *userobject* output, except that a menupage *userobject* output is always big enough to warrant being placed on its own menu page or pages in the PARAMETER area.
- A *userobject* output may be connected to multiple *userobject* inputs. This means a single module's *userobject* output can show up in several menu pages in the PARAMETER area. *Note: If an module's PARAMETER area graphic is shown more than once on a single menu page, the second instance might not be updated when the first instance's value changes and vice versa.*
- When a module with a *userobject* output from the "delay." "detector." "dynamic." "external." "filter." "mixer." "oscillator." "pitchshift." or "reverb" module groups is inserted using the <insert> SOFT KEY, its '*userobject*" output is automatically connected to the head module.

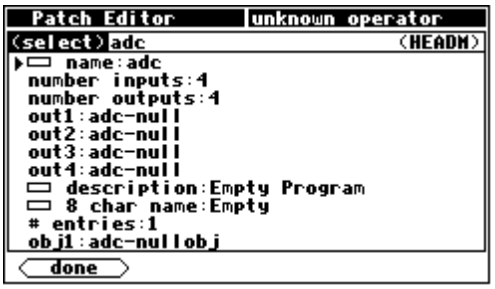
PARAMETER AREA MENU PAGE PLACEMENT

This section goes through a tutorial to show:

- Certain modules that have *userobject* outputs are automatically connected to the head module upon <insert>ion.
- The order of *userobject* connection to the head module affects SOFT KEY location in the PARAMETER area.
- A null *userobject* connected to the head module or a menupage module does not appear in the PARAMETER area menu pages, but rather acts as a place holder.

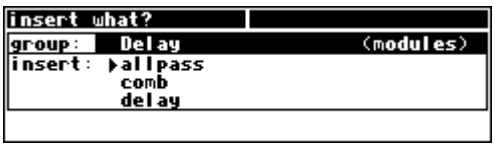
Start with a "clean slate"

- Load Empty Program from the "Utilities" bank.
- Go to the Patch Editor area *(by pressing and holding the PARAMETER key)*
- Select the misc display mode by pressing the <aud only> SOFT KEY three times. Select the module marked hed with the LEFT or RIGHT CURSOR key. Press the <modify> SOFT KEY. To the right is a composite image of what the <modify> menu looks like for the head module at this time.

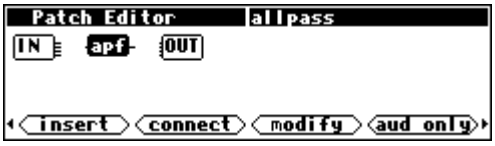


Automatically connecting a userobject to the head module

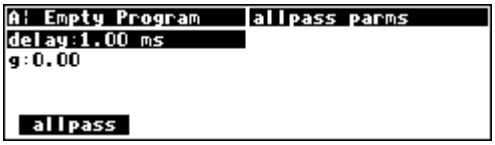
- Press the <done> SOFT KEY to leave the <modify> menu for the head module. Press the <misc> SOFT KEY twice to select the aud+ctrl display mode.
- Press the <insert> SOFT KEY and then the LEFT CURSOR key. Turn the KNOB until the "Delay" group is shown. Press the RIGHT CURSOR key to highlight the allpass module.



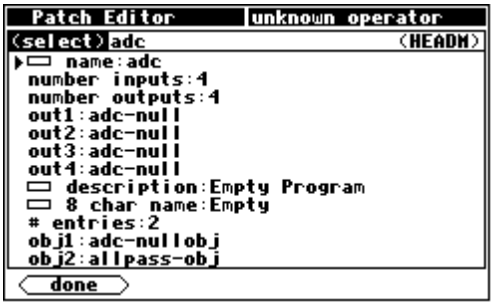
- Press the SELECT key to go ahead with the insertion.



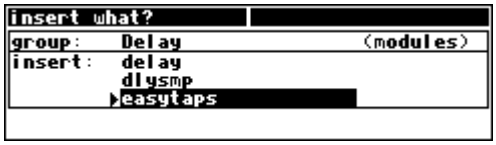
- Press and hold the PARAMETER key to see that there is now a menu page and SOFT KEY for the allpass module.



- Go back to the Patch Editor area and select the misc display mode and use the <modify> SOFT KEY on the head module. This is a composite of what the <modify> SOFT KEY can see in the head module now. This shows that the allpass-obj *userobject* was automatically connected to the head module.

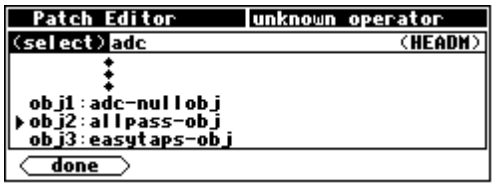


- Go back to the basic Patch Editor area by pressing the <done> SOFT KEY. Now <insert> an easytaps module.



- Return to the PARAMETER area and observe that there is now a second SOFT KEY. This SOFT KEY is stacked because the easytaps' parameters take up more than one menu page.

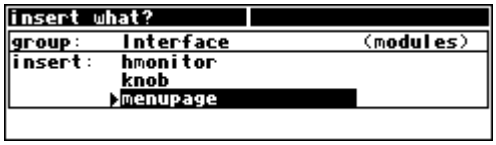
- Go back to the Patch Editor and use the <modify> SOFT KEY on the head module. Note there are now three *userobject*s connected to the head module and that the new *userobject* was connected after the existing two *userobject*s.



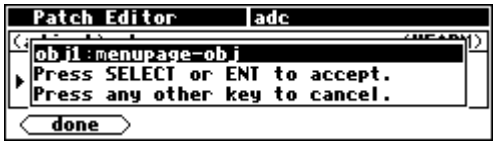
Adding a menupage module

This section demonstrates how an empty menu page shows up in the PARAMETER area with a SOFT KEY.

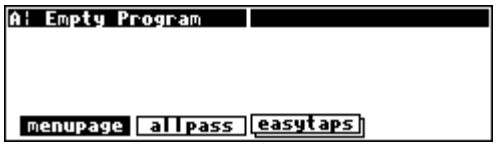
- <insert> a menupage module (from the "Interface" group).



- Use the <modify> SOFT KEY on the head module and change the obj1 *specifier* to menupage-obj.



- Return to the PARAMETER area and observe that the menupage module has created a new menu page and that the allpass and easytaps' SOFT KEYS were bumped over when the null *userobject* was replaced.



STACKED MENU PAGES

This section goes through a tutorial to show:

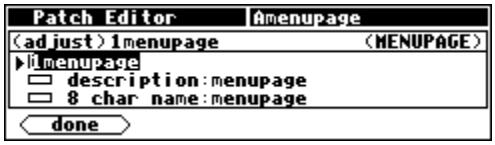
- That modules' parameters are presented on a menu page when their *userobject* outputs are connected to a menupage module *(which is in turn connected to the head module)*.
- That multiple connections of the same module's *userobject* output creates multiple "images" of the module's parameter(s). *Note: The second image of a module's parameter(s) on the same menu page may not be active.*
- That menupage modules' *userobject* outputs may be connected to other menupage modules to create SOFT KEY stacks.

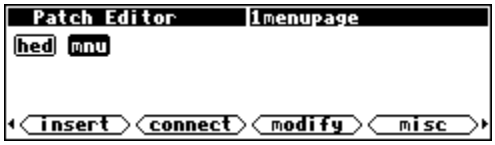
Inserting Multiple menupages

- Load Empty Program from the "Programming" bank.
- Go to the Patch Editor area by pressing and holding the PARAMETER key.
- Select misc display mode by pressing the <aud only> key three times.
- <insert> a menupage module from the interface group.

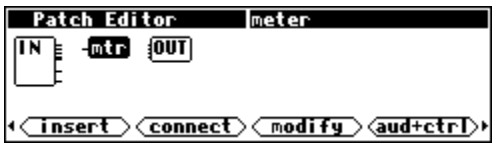
- So that we can differentiate this menupage module from the next one we insert, use the <modify> SOFT KEY on it and change its name to "1menupage." Do this by selecting the name *specifier*, pressing the SELECT key, turning the KNOB right until the highlighted line reads "1menupage." and pressing the SELECT key.

After you've changed the name, press the<done> SOFT KEY and note that the module name is displayed in the upper right corner of the screen.

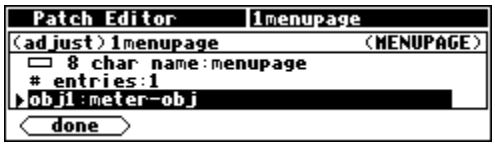




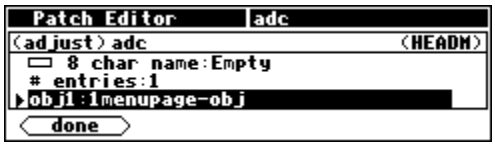
- Select the aud+ctrl display mode.
- <insert> a meter module from the "Interface" group.



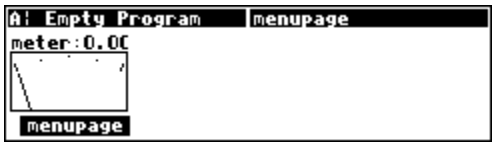
- Go back to the misc display mode and use the <modify> SOFT KEY on the menupage module. (Use the LEFT or RIGHT CURSOR key to choose the menupage module.)
- Set the # entries *specifier* to 1 and then set the obj1 *specifier* to meter-obj. Make sure you use the SELECT key to save each change. Press the <done> SOFT KEY to exit from the <modify> menu.



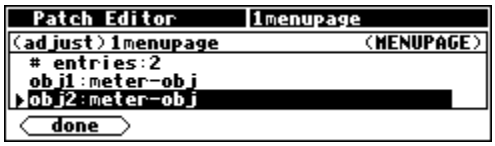
- Use the <modify> SOFT KEY on the head (hed) module. Change its # entries parameter to 1 *(if it isn't already)* and set obj1 to 1menupage-obj.



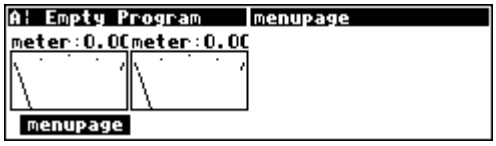
- Return to the PARAMETER area and observe the menu page. Note that it has one meter module monitor. Note also that the SOFT KEY is not stacked and that the menu page's title is "menupage."



- Go back to the Patch Editor, misc display mode and use the <modify> SOFT KEY on the menupage module.
- Set the # entries *specifier* to 2 and then set the obj2 *specifier* to meter-obj (same as obj1).

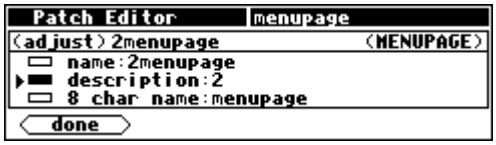


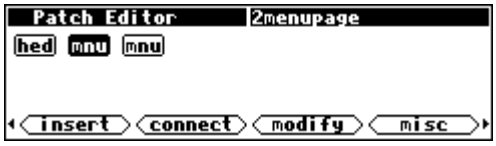
- Return to the PARAMETER area and observe the menu page. Note that it shows two `meter` module monitors *(these are in fact two pictures of the same meter module)*. Notice that the SOFT KEY is not stacked.



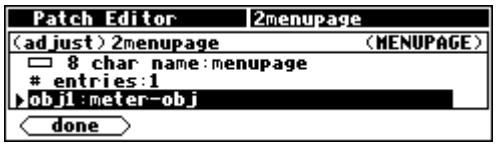
- Go back to the Patch Editor area, `misc` display mode and <insert> another `menupage` module. Edit its module `name` using the <modify> SOFT KEY such that it reads "`2menupage`." Then change the `description` *specifier* to read "`2`" *(you will need to use the CXL key to delete characters)*. Press <done> to exit the <modify> menu.

Note the module name in the upper right corner of the display.

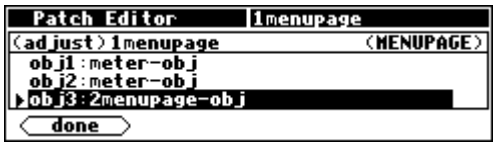




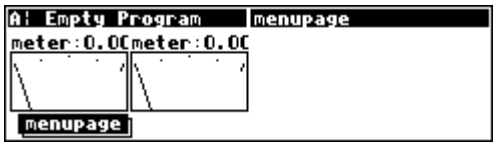
- Go back into the <modify> SOFT KEY menu for the `2menupage` module and set the `# entries` *specifier* to `1` and then change the `obj1` *specifier* to `meter-obj`. Exit from the <modify> menu by pressing <done>.



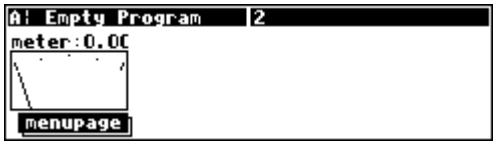
- Now, using the LEFT or RIGHT CURSOR key, select the `1menupage` module. Use the <modify> SOFT KEY and change its `# entries` *specifier* to `3` and its `obj3` to `2menupage`. Press <done>.



- Return to the PARAMETER area. Notice that there is now a stack of SOFT KEY menu pages. The title of the top menu page is still "`menupage`."



- Press the SOFT KEY. The second menu, whose title is "`2`." has only one `meter` module monitor on it.



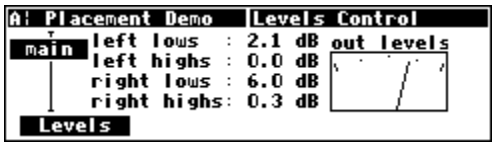
PARAMETER PLACEMENT ON A MENU PAGE

Menu page design may be highly individualized. The "look and feel" of a menu page is important if a program is complex or if there are displayed reactions to adjustments that must be viewed and understood quickly. For instance, if a program uses a `meter` monitor to display the signal level in a given frequency range while a `knob` parameter gives adjustment of the compression level in that frequency range, it is important to have the `knob` parameter and the `meter` monitor on the same menu page. It may also be possible to show the relationship between a `fader` parameter and a `meter` monitor by placing them on the same menu page. This kind of "look and feel" control is performed through the connection of *userobject* outputs to `menupage` modules.

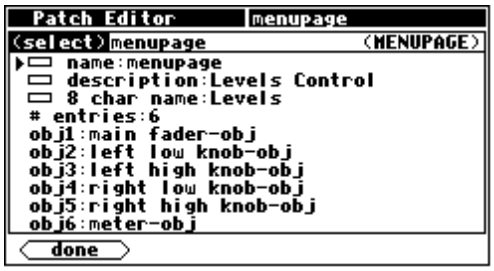
This section goes through a tutorial to show:

- Parameters are presented on a menu page in the PARAMETER area in the order their *userobject*s are connected to a `menupage` module, with placement beginning in the upper left corner of the screen and proceeding to lower right corner of the screen.
- Modules that produce parameters with simple text take up one eighth of the screen *(exception is the textblock module)*. Modules that produce parameters with graphics take up some other portion of the screen. The order that modules are connected to the `menupage` module may cause the parameters to be placed poorly, allowing only a few parameters to appear on a screen where better placement might have allowed more.

Placing a Vfader, a Meter, and Four Knob Modules



It is possible to build a good looking menu page using six modules connected to a `menupage` module. The order of their connection to the `menupage` module is important. The display to the right shows a typical arrangement of the six modules, taken from the program `Placement Demo` found in the "`Programming`" bank.



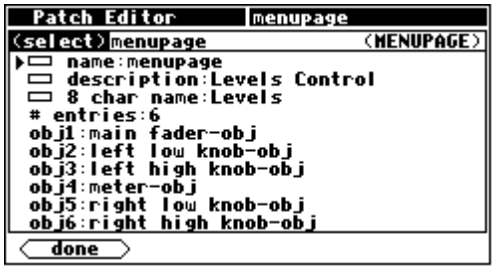
To achieve this arrangement, the modules' *userobject* outputs must be connected to the `menupage` module in exactly the order shown to the right:

1. `main fader-obj`
2. `left low knob-obj`
3. `left high knob-obj`
4. `right low knob-obj`
5. `right high knob-obj`
6. `meter-obj`

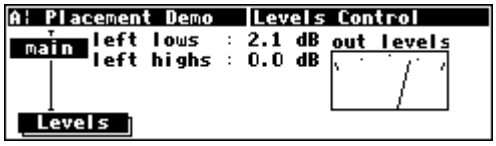
To create a menu page such as the one shown, the programmer inserts the six "parameter" modules and the `menupage` module using the <insert> SOFT KEY and then uses the <modify> SOFT KEY on the `menupage` module. The `# entries` is first set to the desired number (six) and then the `obj` entries are adjusted, one at a time, to connect the *userobject* outputs to the `menupage` module.

If the *userobject* outputs are connected in a different order, the menu page items might not only look bad but might not all fit on the same menu page. For instance, connecting the *userobject* outputs in this order:

1. main fader
2. left low knob
3. left high knob
4. meter
5. right low knob
6. right high knob



results in *two* menu pages *(accessible via a stacked* SOFT KEY*)*!



Since parameters are always placed from upper left to lower right, top to bottom, once the `meter` monitor is placed, (not fitting below the `left highs` parameter) there is no more room below the meter or to the right of the meter on the first menu page. Therefore, a new menu page is created for the latter two `knob` parameters.