Patch-Ware Documentation!!!

Hooray, I’m finally getting around to starting it!

Patch-ware from a high level:

Patch-ware is a library being built to help novice programmers or DSP engineers or even musicians create custom signal processing effects. The main focus of this project is audio processing, but software works on any type of digital signal. That being said, most of the terminology and tooling is designed specifically from audio processing.

The idea for Patch-ware came from research into audio processing VST plugins. Digital audio signals in the computer are basically just lists of numbers. Effects are often controlled by Parameter, which are also just numbers in some algorithm or equation. Why should these numbers be treated differently? A number is just a number… so why shouldn’t we be able to use a signal to control a parameter for an Effect? This idea led to using digital “patches” to carry signals between Effects, and led to the treating of Parameters as input devices that can be controlled by some signal flowing through a digital patch. This in turn led to the splitting of signals, which led to the realization that these Effect and Patch environments acted kind of like Signal Circuits, where a signal can be split and sent to many different Effects for processing, or can become the control for parameters of other Effects or Signal Processors.

Layered Architecture:

Patch-ware has a layered architecture, designed to expose varying abstraction depth depending on how deep you want to go:

[UI] (coming soon---ish)

[XML]

[Patching layer]

[Processor layer]

There is a UI layer in the works that will allow for a user to simply use point and click gestures to create and edit custom effects… but that is not ready yet and will take some time to get up and running. This UI will be designed general enough to run on standard C++ with minimal dependencies. The goal is to write the UI in such a way that it will be portable to Windows, Linux and Mac, and maybe even Android (if I’m feeling ambitious down the road).

The XML layer allows users of patch-ware to not have to worry about writing C++ to construct their own custom effects. Users can define the properties of their own effects through an easy to understand XML format. XML files are parsed and (if written properly) generate patch-ware devices and signal processors. As of right now, the driver for this whole system still requires some cracking into the C++ code, but the XML greatly reduces the amount of code required for creating your own effects.

The Patching Layer provides an interface for creating and easily tracking different effects. This layer requires the user of the software to explicitly write C++ code to create Effects and create patches to carry the signal. This layer can execute signal processing, but does so at a relatively slow rate because it makes tedious checks to ensure that the signals are flowing through the effects properly. This layer can be used for debugging new Effects or Processors because of the added layer of information and checks at runtime. To walk amongst this layer, it is recommended to have at least a moderate understanding of the C++ language, including topics like: multiple inheritance, pointers (deep and shallow copies), data structures like maps, graphs, linked lists… etc.

The Processor Layer provides an optimized runtime layer built from the upper Patching Layer. This layer is built for speed and is the end-goal for executable code. This layer cuts out all the overhead from the Patching layer and uses Processing Blocking and pointers to quickly execute calculations. This is where the actual meat-and-potatoes of the processing is done. Each effect defines its own method for processing a signal. Defining your own Effect can be as easy as implementing the SignalProcessor abstract class by defining your own processSignal() method. It is recommended that to dig here you have a basic knowledge of DSP, pointers, data structures and algorithms.

Signal Circuits:

Referred to as Circuits, they contain and manage many different Effects and the patches that connect them. All Effects in Patch-ware from the most outer layer are Circuits. Circuits have inputs and outputs just like Effects, and process signals like Effects… Because of this common way of interacting with Circuits, they can be though of *as* Effects. Instead of running some signal processing algorithm or equation on a signal, they relay the incoming signal into their internal Patches. The signal then trickles through the internal Effects and Signal Processors, then comes out the output of the Circuit as a processed signal.

A close up of a logo

Description automatically generated

A simple Circuit

Consider the figure above. This is a VERY simple circuit. The signal coming into the Circuit enters on the left. The first patch (squiggly line) carries the signal into a Gain effect. The gain effect processes the signal (increases or decreases the signal. See Gain documentation), then outputs it the to next patch (the second squiggly line) which carries the signal to the output of the Circuit. Notice the Gain has only a single parameter (the circle “knob”), “Level” (not labeled). This Circuit is effectively a simple Gain effect, but let’s take a look at a more complicated Circuit.