Qualm: Queueing Applications for Live MIDI

Stephen Peters

October 4, 2011

1 Rationale

Qualm was created to deal with the problem of live shows that need to switch among a large set of pre-defined synthesizer patches. This has come in most handy for amateur productions of musicals, starting with a musical group at MIT.

Many musicals would be scored for an acoustic orchestra, but more and more "rock"-style shows were appearing that wished to make use of different keyboard sounds. For these, we would either use piano throughout, or use whatever pre-programmed sounds the keyboard had. If we wanted to get really fancy, we'd borrow a synth module and select them by hand.

This worked fine until *Bat Boy: The Musical*. Before that show, we were looking at maybe a dozen patches in a show. But *Bat Boy* required two keyboard players, and one of those players had to work with both an upper and a lower keyboard simultaneously. Each part had over two hundred different changes to select.

For the K1 part, we did a souped-up version of our old "select patches by hand" method. We programmed all the patches sequentially into a Korg NS5R synthesizer module, and configured a pedal to go through them one-by-one. This worked, but since the synth module would only hold a limited number, we had to store each act separately, and download lots of information to the module before each act. Plus, if you wanted to change the way one instrument sounded, you had to make the change and then copy it everywhere that sound got used; needless to say, that was a rather tedious operation, and became even more so when you wanted to insert or delete a patch.

To avoid all that – and because the upper- and lower-keyboard thing would have made that setup especially tough – we chose a different route for the second keyboard part. Each patch would be stored just once on a second synth module (a Roland JV-1080), and we created a computer program that would store information about where each patch was, and when the right moment occurred, it would just look up what the next patch should be and tell the synth module to load it. Not only did we not have to worry about copying patches around, but this allowed us to be really flexible about what kind of trigger would switch the patch; the pianist could hit the Enter key on the computer, step on a pedal, hit a particular note on the keyboard, even temporarily set a special key as the trigger so that it would switch automatically while he or she played. And the control file could be edited to insert or delete new patches, or even add in control of a drum module at the last minute.

The software that did all this was dubbed Qualm, ostensibly "Queueing Applications for Live MIDI," but really because the music directors for that show were very nervous about whether this could work. Since then, it's been used for other patch-heavy shows like *Star Wars Trilogy: Musical Edition*, *The Who's Tommy*, and *Children of Eden*. Most of these were operating a single keyboard with many patches, but *Children of Eden*, for one, actually used a single synthesizer module to deal with input from three separate keyboards at the same time, utilizing a separate stream of patches for each one.

2 Installation

Qualm is a Java program, and as such can be installed by simply downloading a jar file. In order to get MIDI support working properly, however, there are certain system requirements that vary from operating system to operating system. Some of this information can also be found on the Java Sound Resources[1] pages.

2.1 MIDI I/O

The first hurdle for getting Qualm working is making sure that the Java subsystem can handle output to MIDI hardware devices. For different operating systems, there are different steps that must be followed:

2.1.1 Linux

For Linux, getting MIDI I/O working means that you need the Advanced Linux Sound Architecture, or ALSA. If your laptop has a reasonably new install (Linux kernel 2.6 series), you already have ALSA. That should mean that you can skip this paragraph. If not, check to see if you have alsa by opening a terminal window and trying to run `alsamixer`. If you have ALSA installed and configured, this should run and display a rather crude-looking graph of the current mixer settings, but you can just hit Control-C to exit. If you don't have ALSA installed, you'll need to do so. I'm not going to go through all the rigmarole for doing this here; there are how-to guides to ALSA available on the web.

2.1.2 MacOS X

The Java that comes with OSX doesn't have support for MIDI I/O out of the box. However, it is fairly easy to locate the Plumstone MIDI[2] project, which will do the right thing for OSX machines. For PPC Macs, this exists as freeware; a similar project (Mandolane) exists as shareware for PPC/Intel installs.

Installing the Plumstone jar file in your computer's `/Library/Java/Extensions` folder will enable MIDI I/O.

This extra software will not be needed for OSX machines that are running the 'Lion' operating system.

2.1.3 Windows

As long as you're using Java 1.5 or later, you shouldn't have a problem with MIDI I/O under Windows.

2.2 The GetOpt Package

Assuming ALSA is installed and running, what else does Qualm need? Well, it needs the GNU GetOpt options-parsing package. This is a jar file, available from the author's download page[3]. On Linux, this is probably already available as a package named `java-getopt`, `gnu.getopt`, or `libgetopt-java`, depending on what brand of Linux you're using. On Linux or OSX, this jar file should be installed into `/usr/share/java`; on Windows you should install it and add its location to your CLASSPATH variable.

[1] http://jsresources.org
[2] http://www.mandolane.co.uk/swPlumstone.html
[3] http://www.urbanophile.com/arenn/hacking/download.html

3 Qualm Control Files

Qualm is fairly straightforward. It takes a control file written in XML, consisting of a list of patches, and a set of "cues" (gathered into independent cue "streams") which describe the patch changes and the events that will trigger the next cue in the sequence.

An example Qualm data file is shown in Figure 1.

All the XML information for Qualm is surrounded by a `qualm-data` element. The first element within the `qualm-data` is usually a `title` element, which can give a descriptive name of the task for which the file has been prepared. The rest of the data is divided broadly into three sections; the channel specification, the patch listing, and the cue streams.

3.1 Channel Specifications

```
<midi-channels>
  <channel num="1" device-type="Roland JV-1080">Lower Keyboard</channel>
  <channel num="2">Upper Keyboard</channel>
</midi-channels>
```

MIDI defines 16 different channels that can independently receive note data. Any channel that is used by Qualm in a control file, however, needs to first be specified within the `midi-channels` element section of the file.

Each channel is defined through a `channel` element, which can contain a brief piece of text describing the channel data. This description will be seen in Qualm when it reports on patch changes or other channel updates. `channel` uses the attributes:

num – The MIDI channel number that will be used for control. A required attribute.

device-type – The synth model that Qualm will use when sending events on the channel. This value is used to determine the class name for a delegate which Qualm uses to format messages on the MIDI bus. If not supplied, Qualm will assume a very basic synthesizer model. (Optional)

Using a channel in a Qualm data file that was not defined in `midi-channels` could result in run-time errors.

3.2 Patch Listing

The `patches` element within the Qualm file contains the list of different sounds, or patches, that will be requested of the synth module during Qualm's operation. The `patches` section can contain either `patch` or `patch-alias` elements:

3.2.1 Patch Definitions

```
<patch id="P3" bank="User" num="3" remark="split at C5">Strs/FzzClav</patch>
<patch id="P4" bank="PrA"  num="15" volume="23%">Rhodes</patch>
```

The `patch` element defines the name of the patch that Qualm will display while using that patch in operation. This element can contain the attributes:

id – A unique identifier for this patch. This will be used in the cue section to create patch change events.

num – The patch number (or "program number") that will be sent to the synthesizer to select the sound for this patch.

bank – A number or string that will select the appropriate "bank" of sound programs. Most synthesizers have more than 128 sounds, and thus need to select among multiple banks. Because the different

```
<?xml version="1.0"?>
<!DOCTYPE qualm-data PUBLIC "-//QUALM/DTD Qualm Data//EN" "http://qualm.berlios.de/qualm.dtd">

<qualm-data>
  <title>Example Qualm File</title>
  <midi-channels>
    <channel num="1" device-type="Roland JV-1080">Lower Keyboard</channel>
    <channel num="2">Upper Keyboard</channel>
    <channel num="4" device-type="Alesis">Drum Kit</channel>
  </midi-channels>

  <patches>
    <patch id="Nice_Piano" bank="User" num="1">Nice Piano</patch>
    <patch id="P2" bank="User" num="2">TremStr/Str</patch>
    <patch id="P3" bank="User" num="3" remark="split at C5">Strs/FzzClav</patch>
    <patch id="P4" bank="PrA"  num="15" volume="23%">Rhodes</patch>
    <patch-alias id="P5" target="Nice_Piano">Good Piano</patch-alias>
    <patch id="P95" bank="User" num="95">LKB/TmpF</patch>
    <patch id="P96" bank="User" num="96">TmpG/TmpD</patch>
  </patches>

  <cue-stream id="First_Stream">
    <global>
      <trigger><note-on channel="1" note="c6"/></trigger>
      <trigger reverse="yes"><note-on channel="1" note="c2"/></trigger>
      <map-events>
        <map-from><control-change channel="1" control="damper"/></map-from>
        <map-to><control-change channel="2" control="80"/></map-to>
      </map-events>
    </global>
    <cue song="1a" measure="10">
      <events>
        <program-change channel="1" patch="P3"/>
        <advance stream="Second_Stream" song="2" measure="10"/>
      </events>
    </cue>
    <cue song="3" measure="1">
      <events><program-change channel="2" patch="P3"/></events>
      <trigger><note-on channel="2" note="g2"/></trigger>
    </cue>
    <cue song="3" measure="16">
      <events>
        <program-change channel="4" patch="P96"/>
        <program-change channel="1" patch="P5"/>
        <note-on channel="10" note="g2"/>
      </events>
    </cue>
  </cue-stream>
  <cue-stream id="Second_Stream">
    <global>
      <trigger><note-on channel="2" note="c6"/></trigger>
    </global>
    <cue song="1" measure="1">
      <events>
        <program-change channel="2" patch="P96"/>
<control-change channel="2" control="pan" value="10"/>
<sysex>F04110421240007F0041F7</sysex>
      </events>
    </cue>
    <cue song="2" measure="10">
      <events><program-change channel="2" patch="P95"/></events>
      <map-events>
        <map-from><note-on channel="1" note="C4-Bb6"/></map-from>
        <map-to><note-on channel="2"/></map-to>
      </map-events>
      <map-events>
        <map-from><note-off channel="1" note="C4-Bb6"/></map-from>
        <map-to><note-off channel="2"/></map-to>
      </map-events>
    </cue>
  </cue-stream>
</qualm-data>
```

Figure 1: An example Qualm data file.

synthesizers might choose different naming conventions for the banks, string values for the bank are interpreted by the target synth. [4] (Optional)

remark – A comment on the patch which the synth programmer can use to make notes about the development of the sound. (Optional)

volume – When loading this patch, tell the synthesizer to set the volume for that channel to the given value. This value is either a number from 0-127, or can be written as a percentage using a `%` character (as in `"50%"`).[5] (Optional)

3.2.2 Patch Aliases

```
<patch-alias id="P5" target="Nice_Piano">Good Piano</patch-alias>
```

The `patch-alias` element defines a patch which is the same sound as an already existing patch in the list. This can be used to call the same sound by a variety of different names, or to create specific scenarios where a patch should be played noticably louder or softer than usual. This element can contain the attributes:

target – The patch identifer that this alias references. When this alias is called in the Qualm cue stream, it will load the same sound as the referenced patch. *Note:* This should be a patch identifer that exists earlier in the list of patches than this alias; trying to "forward reference" a later patch will cause warnings when loading your control file.

id – A unique identifier for this patch alias. All patches and patch aliases must have a unique identifier; there can be no duplicates in the list of patches.

volume – Overrides the volume level for the given patch when accessed through this alias. For more information, see the `volume` attribute on the `patch` element in the previous section. (Optional)

3.3 Cue Streams and Cues

The remaining section of the Qualm control file is for defining *cues* – namely, what happens at a particular moment in the score, and what Qualm will expect to see in order to switch to the next cue.

These cues are bundled up into one or more `cue-stream` elements, each of which represents an independent series of cues. Each `cue-stream` element can contain an optional `id` attribute which is used to reference the stream when necessary (see 3.3.1 for more details).

Within the `cue-stream` element are a set of `cue` elements, each of which has a pair of attributes, `song` and `measure`, which define the order in which the cues should be visited. Cues will be ordered first by the song number, and then by the measure number within a song. If a letter appears in either the song or the measure information, it is assumed to come before the next-higher number, if appropriate. As such, a song labeled "13b" will come after songs "13" and "13a" and before song "14". A measure numbered simply "a" or "A" will be the first measure in the song, even before measure "1". This corresponds to accepted notations.

The contents of a `cue` element are for defining the cue's actions. These are divided into *events*, *map events*, and *triggers*.

3.3.1 Events

In Qualm, *events* are the actions which occur when a cue is reached. These are broadly defined into three groups – *program changes* which switch patches, *stream advances* which cause another `cue-stream` element to change at the same time, and *note window changes*, which are used to limit the range of notes that the channel can use.

[4] Patch libraries are often synthesizer-specific, so if many different synthesizer modules are used in the same control file, patches should be created with an eye towards which synth the patch will be destined for.

[5] Because this feature uses standard MIDI controls for changing a channel's volume, it may interact poorly with volume pedals or sliders which are meant to perform the same effect. If you're using devices such as these in your synthesizer setup, it may be best to avoid using the `volume` attribute.

Program Changes

```
<program-change channel="2" patch="P96"/>
```

The `program-change` element is the most common type of event. When Qualm switches to a cue that has a `program-change` element, it locates the requested `patch` element or `patch-alias` element from the `patches` element section (see section 3.2) and asks the synth module listening on the associated channel to load the defined patch. The `program-change` element can contain the following attributes:

`channel` – The MIDI channel to which the request should be sent. This should match one of the channels defined in the `midi-channels` element section of the control file. A required attribute.

`patch` – The unique identifier for the patch, as previously defined in the `patches` element section. If this value does not match one of the defined patches, an error will occur when loading the control file. A required attribute.

Note Events

```
<note-on channel="10" note="g2"/>
```

You can specify that certain note events be triggered by using the `note-on` element and `note-off` element. Attributes for these elements include:

`channel` – The MIDI channel to which the note request should be sent. A required attribute.

`note` – The designation for the pitch of the note request. This can be either an integer for the MIDI note number, or a name like "C4" or "Bb3".

`value` – The velocity of the key strike, from 0 to 127. (Optional)

Control Changes

```
<control-change channel="2" control="pan" value="10"/>
```

Control changes that should be initiated at a cue change can be specified using a `control-change` element. Attributes will include:

`channel` – The MIDI channel to which the control-change should be issued. A required attribute.

`control` – A designation for which control is to be manipulated; either as a MIDI controller number or a name for the controller. Table 1 lists the allowable names.

`value` – The value to which the control should be set.

System Exclusive Messages

```
<sysex>F04110421240007F0041F7</sysex>
```

At times, it may be useful to issue a system-specific set of raw data, in order to perform specialized functionality that does not fit into the standard defined API. In MIDI parlance this is referred to as a system-exclusive message, and can be accomplished in Qualm through a `sysex` element in the `events` element section. The data for the system-exclusive should be notated in hexadecimal notation, so the value `F04110` would send three bytes, "F0", "41", and "10" as a system-exclusive message.

Whitespace in the `sysex` element content will be ignored.

Advancing Streams

```
<advance stream="Second_Stream" song="2" measure="10"/>
```

When working with multiple input streams, the `advance` element can be used to allow input on one stream to advance the patches on another. For example, if you wanted one keyboard's patch-change request to alter both itself and a second keyboard simultaneously. The `song` and `measure` attributes can be used to advance the target stream to a particular point; without these attributes the target stream will be advanced one cue.

`stream` – The id of the cue-stream to advance.

`song` – An optional attribute indicating the target song number for the stream. This is usually used in conjunction with the `measure` attribute below.

`measure` – An optional attribute indicating the target measure number for the stream.

Note Window Changes

```
<note-window-change channel="1" bottom="G4"/>
```

Some synthesizers can handle requests to change the top and bottom notes of a patch's range, so that they will only respond to notes in a certain part of the keyboard. This allows you to create splits – for example, making the left side of the keyboard play an acoustic bass, while the right side of the keyboard plays piano. This can be done through MIDI "SysEx" messages, but are not a standard part of MIDI, so are not available for all synthesizer modules.[6]

If the `note-window-change` element action is available for your synthesizer, you can provide the following attributes:

`channel` – The MIDI channel to which the request should be sent. This should match one of the channels defined in the `midi-channels` element section of the control file. A required attribute.

`bottom` – The note specification for the bottom-most note in the range. This can be either an integer for the MIDI note number, or a name like "C4" or "Bb3". (Optional)

`top` – The note specification for the top-most note in the range. (Optional)

You must provide at least one of either the `bottom` or `top` attributes.

3.3.2 Triggers

```
<trigger><note-on channel="2" note="g2"/></trigger>
```

In Qualm, *triggers* are the actions that advance to the next cue in a stream. Defining a trigger causes Qualm to listen to the MIDI events as they are delivered, and watch for events that match a particular pattern. When a matching event is found, Qualm will automatically advance to the next patch in the cue stream.

Every `cue` element can have a set of `trigger` elements that define the matching events for moving to the next cue. The contents of the `trigger` element define the type of MIDI event that will be matched. Usually these are *note on* events, but *note off* and *control change* events can be matched as well.

The `trigger` element can contain the `reverse` attribute, which if given the value "yes" will indicate to Qualm that it should move a cue *backward* in the stream, rather than forward.

A `trigger` element with a `delay` attribute indicates that the trigger should wait the given number of seconds after activation before advancing to the next cue. If later MIDI events match the trigger after the delay timer has begun (for example, if the same note is played again), a new timer will not be started.

[6] This kind of behaviour can also be handled by Qualm using "note-on" and "note-off" event maps. This can be a useful solution if the synthesizer you're using doesn't support the `note-window-change` element.

Name	*CC #*
modulation	1
breath	2
foot	4
volume	7
balance	8
pan	10
expression	11
effect 1	12
effect 2	13
damper	64
sustain	
portamento	65
sustenuto	66
soft	67
legato	68
sound off	120
reset controllers	121
notes off	123

Table 1: Available Qualm controller names, and the MIDI controller change (CC) number.

3.3.3 Map Events

```
<map-events>
  <map-from><note-on channel="1" note="C4-Bb6"/></map-from>
  <map-to><note-on channel="2"/></map-to>
  <map-to><note-on channel="3"/></map-to>
</map-events>
<map-events>
  <map-from><control-change channel="1" control="damper"/></map-from>
  <map-to><control-change channel="2" control="80"/></map-to>
</map-events>
```

A *map event* can be used to take an incoming MIDI event that matches certain parameters, and output one or more similar MIDI events. For example, it could be used to allow a sustain pedal hooked up to one keyboard to automatically control the sustain on another, or alter a spare MIDI controller so that it operates as a volume control.

The `map-events` element can appear within a `cue` element to specify mappings that will be active while that cue is in effect. It will contain both a `map-from` element and one or more `map-to` elements, each of which can contain either a `note-on` element, `note-off` element, or `control-change` element to define the kind of event that will be handled by the mapper. Each of these may contain a `channel` attribute, which will indicate which MIDI channel the event will be read from (or written to).

Within `map-from` element, a `note-on` element or `note-off` element may contain a `note` attribute, which will indicate the note (or range of notes) that match the event. For example, a value of "A#2" for `note` will match only those notes for that value, whereas closed ranges like "C4-Gb6" or open-ended ranges like "-E3" will match ranges of matching notes. When mapping such a MIDI event to another note element, the actual note value will be copied over.

When specifying `control-change` elements, the `control` attribute specifies which control is being manipulated. Either a MIDI controller number or a name for the controller can be used. Table 1 lists the allowable names.

3.3.4 Global Triggers and Map Events

Most triggers and events are ones that should apply across the entire Qualm control file – for example, a pedal-based input for triggering patch advance, or a dedicated note on the keyboard. For these, each `cue-stream` element can have a `global` element section, containing `trigger` elements and `map-events` elements that apply across all cues in the stream.

4 Running Qualm

Qualm is started by executing the jar file, providing the location of the control file on the command line. The control file location can be either a path on your local system, or a URL to a remote file.

Once Qualm is running, it will occasionally pepper the output with lines like

```
K2 -> Sust. String Pad
```

These are simply announcing that it's just changed the patch on the channel called "K2" to a "Sust. String Pad" sound. Most of the time, though, what you see is something like:

```
14.1-14.54 | 14.19-14.65 | 14.1-14.64>
```

This is the main prompt. It's telling you what cue it thinks each cue stream is currently handling (in this example, there are three streams running simultaneously). Each cue is actually listing both the current cue and the next cue it will expect in that stream. All the cues are indexed by song and measure number, so "14.19-14.65" says that it is playing the patches associated with song 14, between measures 19 and 65.

You can tell Qualm to advance to a certain point in the score by entering two numbers, separated by a period ("."). So, if you want to go to song 17, measure 22, you can type "17.22", and then Qualm will announce that it has reset the patches for that particular location in the score, and placed each cue stream at the proper cue:

```
14.1-14.54 | 14.19-14.65 | 14.1-14.64> 17.22
K1 -> Harp
K2 -> Dulcimer
K3 -> Pulse Keys
15.1-18.1 | 17.22-17.34 | 17.14-17.45>
```

This is mostly useful for rehearsals, when your fearless music director announces that everyone's going to start playing at a specific spot. It's also occasionally useful during shows, when you just want to sync up during long breaks to make sure you're all on the same page (literally).

Other commands that you can type at the prompt are:

version – reports the current Qualm version and build information.

reset – this resets all the cue streams to the start of the show; kind of the same as typing "1.1".

quit – does what you probably expect it to. Go ahead, type it in. Quitter.

dump – mostly useful for debugging; prints out a text representation of the entire control file.

showmidi – more debugging; shows all the MIDI events that pass through the unit (you can also get this by adding a "–debugmidi" option on the command line). **unshowmidi** will turn this off.

adv <stream> – advance the stream named *stream* forward one patch. If you just hit "Enter" on the keyboard, it has the same effect as typing "adv" with the name of the first stream.

plugin [remove] <plugin-name> – controls the plugin facility of Qualm. For more information, see section 5.

save – saves the current preferences (which in this case is really just the list of plugins) so that Qualm will use them next time.

4.1 Qualm Command Line Options

Some options can be given on the command line to alter Qualm's behavior:

`--input <port>` – sets the name of the input MIDI port to use for receiving incoming MIDI events. You can retrieve a list of possible ports using the `--list` option.

`--output <port>` – sets the name of the output MIDI port to use when sending out new MIDI events. You can retrieve a list of possible ports using the `--list` option.

`--nomidi` – tells Qualm not to bother setting up any MIDI input or output ports, and instead just load the command file. Providing this option will allow you to issue commands to the Qualm prompt, but will not allow for controlling MIDI equipment in any way.

`--debugmidi` – is a useful debugging option; it automatically sets Qualm to print a description of all received MIDI events to the console. This is equivalent to starting Qualm and typing the '`showmidi`' command at the prompt. This optional also prints debug information that is helpful for finding input and output MIDI ports.

`--lint` – is used to carefully check the control file for formatting errors or other potential problems. Once Qualm has done so, it will run normally.

`--help` – prints out a brief description of these options.

5 Plugins

Qualm plugins are used to create additional functionality making use of the Qualm engine. Currently, most plugins that have been developed are for easy-to-read monitors of the Qualm status.

All the plugins use interfaces defined under the 'qualm.plugins' package. These interfaces include:

qualm.plugins.CueChangeNotification – is used for plugins that want updates when one of the running cue streams moves to a new cue.

qualm.plugins.PatchChangeNotification – is used for plugins that want updates when one or more of the patches on a MIDI channel is changed.

qualm.plugins.EventMapperNotification – is used for plugins that want updates when the current set of event maps is changed.

Often, these interfaces are triggered all at the same time – i.e., when Qualm advances to a new cue.

Plugins that use these interfaces can be loaded through the `plugin` command at the Qualm prompt. Typing "`plugin [plugin.name]`" will load a new plugin into the running system, whereas `plugin remove [plugin.name]` will disable it. The current set of plugins can be stored for a later session at any time using the `save` command.

Example plugins include:

qualm.plugins.DisplayCueChange – shows a window indicating the current cue positions for all running cue streams.

qualm.plugins.DisplayPatchChange – shows a window indicating the current patches for all running cue streams.

qualm.plugins.NetworkNotifier – starts a thread that will allow external programs to subscribe to all the plugin notification events over the network. An example reader of this information is in `misc/ExampleNetworkReader.java`.

Index