Q2Q 0.7.0 User Guide

John Wostenberg

November 2022

Contents

Overview

This user guide is a work-in-progress document and may be missing some information.

Q2Q is a real-time, cross-platform cueing system for live performances. You can use it for all of your sound needs. Q2Q organizes a show across two axes instead of one: first by cues (in rows), and secondly by tracks (in columns). This is described in more detail in the section about the cue-track grid.

App layout overview

Toolbar

The toolbar appears at the top of the main window, below the menu bar and just above the cue-track grid. This includes:

- The "GO" button, which triggers the currently armed cue when pressed (also triggerable via the space bar).
- The trigger palette, which contains the controls for every trigger type in Q2Q.
- The default cue name editor, which allows you to edit the default cue name for on a project-by-project basis (see Automatic cue naming).
- The Infobox, which displays helpful information when you hover over parts of the application.

Cue-track grid

Q2Q organizes itself in a two-dimensional layout rather than one-dimensionally. Cues run down the screen, while tracks run across.

Cues represent a set of actions (triggers) to be taken when the cue is taken. In this way, Q2Q's "y-axis" can be thought of as corresponding to time: the lower down you go, the later in the show the cue is. There is a one-to-one correspondence between "pressing the GO button" and a cue: in other words, when you press "GO," *exactly one cue* will be taken in a completely predictable fashion. There is no guesswork to determine where the "armed cue" cursor will end up next, as it is always the cue right after the one just started. To make more than one thing happen in a cue, you can use tracks and triggers.

Tracks provide a way to organize related triggers/sounds together, and are laid out in columns. If cues running along the y-axis correspond to time, tracks running along the x-axis therefore correspond to a spacial (aka organizational) axis, similar to layers in many other applications (such as Photoshop or Audacity). For example, you may want to group all the music into a "Music" track and all the sound effects into another "SFX" track. This allows useful things like adjusting the levels/panning of the track all together.

Cue editor

The cue editor is found on the bottom-left corner of the main window, and shows you some information about the cue as a whole, such as the cue's raw and display names. It also allows you to view and edit notes for the cue, which is a blob of arbitrary text stored alongside the cue. The cue editor (and therefore the notes widget) is always visible, no matter what trigger is selected within the cue, making it useful to convey thing such as cue lines, page numbers, or even just informational notes about the to the operator. A future version of Q2Q will have an "Operator View," which will take heavy advantage of these notes. It will display information about the next-up cue in a much more condensed manner, excluding the detailed trigger editor and collapsing tracks down.

Trigger editor

The trigger editor, adjacent to the cue editor, sits on the bottom-right corner of the main window. Here you will find controls for attributes of the selected trigger, such as slice points, levels, and OSC parameters, to name a few.

Track editor

The track editor can be found on the rightmost side of the main window, and displays controls for viewing and editing settings for the selected track. The output device dropdown allows you to set where the track's audio will route to, allowing you to choose from a list of all the detected system devices, from each supported API. If an unknown device is selected for output (often caused by disconnecting the device, or by loading the project on a different computer with different devices connected), the device will show as "offline," and instead output to the system default until you change it. The levels control adjusts the levels for the track as a whole (which get applied after individual sound instance levels).

On Windows, Q2Q works with WASAPI and ASIO output devices. ASIO devices work with any number of channels, while WASAPI devices are restricted to 1 or 2 channels. Professional audio interfaces often support exposing themselves to multiple different Windows audio APIs, so if you want to make full channel use of your hardware, make sure to select the ASIO version of the device from Q2Q.

On macOS, Q2Q works with any CoreAudio output device, which can support any number of channels.

Views

There are two views accessible in Q2Q: Designer View, and Operator View; with Designer View being the default. You can switch between the two under the View menu, or by the keyboard shortcuts.

Designer View is a detailed workspace suitable for building or making changes to the project, allowing the full freedom to see all the tracks and the cues as well

as edit properties of both. It contains the toolbar, cue-track grid, cue/trigger editor, and track editor.

Operator View presents a simplified, clutterless view-only experience suitable for show operators. It collapses the cue-track grid down into a summarized cue list, displaying information such as the cue name, the triggers in the cue, the total duration of the cue, as well as notes. It does not display individual tracks. This makes it easier to tell at a glance exactly what a cue is going to do and how long it will take to do it. Unlike Designer View, in Operator View, the GO button is placed prominently at the bottom of the screen, instead of at the top of the screen.

Triggers

A *trigger* is the smallest discrete unit of action that can be taken by Q2Q. There are several different kinds of triggers, and a cue can hold many triggers. You can edit a trigger's attributes by clicking to select the trigger in the cue-track grid, and interacting with the trigger editor that appears.

Start trigger

Starts a sound instance loaded from an audio file. When you add a new start trigger to a show by dragging and dropping, Q2Q will prompt you to pick an audio file for the trigger.

Start trigger properties include:

- Predelay - A time (in seconds) to wait after the trigger is executed before starting the sound.
- Levels - Adjusts the initial volume levels of the sound instance.
- Crosspoints - Defines how the sound instance's channels map to the track's output channels. The rows are source channels, and the columns are output channels.
- Slices - Defines regions within the sound file that may be looped.

Stop trigger

Stops a sound instance if playing.

Fade trigger

Smoothly varies a playing sound instance's levels to a different configuration over a specified duration. This can be used to accomplish fade ins, fade outs, and spacial panning.

The *Pan Law* dropdown determines which interpolation algorithm to use to vary the volume levels. For example, the *linear* pan law uses a linear formula to vary the volume, while the *constant power* pan law employs a non-linear sine/cosine curve.

For more reading about pan laws, see: https://www.cs.cmu.edu/~music/icm-online/readings/panlaws/.

Crossfade trigger

Fades one sound instance in ("Fade in target") and another out ("Fade out target") simultaneously. Duration and constant power behave the same as in fade triggers.

Devamp trigger

Make any currently looping slice region stop looping and continue onto the next slice region. You can use this to loop a section of audio over and over until a you want to smoothly transition to the rest of the audio. For example, you may have a bar of music to loop indefinitely while actors speak a block of dialogue, and to move on to the rest of the music once they finish the dialogue. This is easily possible with slice points and a devamp trigger.

OSC Trigger

OSC triggers send OSC messages to other programs or devices. OSC, short for **O**pen **S**ound **C**ontrol, is an industry standard protocol that allows different systems to communicate with each other in very high-level ways. It was originally designed with sound-based applications (such as synthesizers) in mind, but has many more uses today. You can read more about OSC at: http://opensoundcontrol.org/.

To make Q2Q send OSC messages to another device, add an OSC trigger to a cue. You can customize its message parameters within the trigger editor.

Parameters

Address Specifies the address of the address pattern of the message. This can consist of:

- A simple address, such as `/instruments/lead_synth/frequency`
- An address containing simple wildcards, such as `/instruments/*/frequency` (controlling all instruments' frequencies from our example synthesizer)
- An address with a more specific wildcard, such as `/instruments/{lead_synth,bass}/frequency` (which would control only `lead_synth` and `bass`, but not `pad`).

Arguments Arguments are optional extra data supplied along with the OSC message. Click the plus (+) button to add arguments, and the (X) button to remove arguments. Use the datatype dropdown on an argument to change the argument's datatypes. Q2Q currently supports the following datatypes:

- String
- Integer
- Float
- Boolean
- Impulse
- None (also known as *Nil*)

Specifying where OSC messages are sent to

You can tell Q2Q where to send OSC messages in the *Network patch* window (either by selecting `File>Network Patch...` in the menu, or by clicking the

gear icon in the OSC trigger editor). Edit the host, port, and protocol of the sending connection to match whatever the other device or program is listening on. Pay special attention here—nothing will work if the settings are not exactly correct.

These settings are saved on a per-project basis, not globally.

Examples

Imagine you have an OSC-enabled synthesizer that you want to control from Q2Q, and it understands the following messages:

Message address	Arguments	Description
/instruments/lead_synth/frequency	float	Sets the lead synth's frequency in Hz
/instruments/lead_synth/volume	float	Sets the lead synth's volume in dB
/instruments/lead_synth/eq3	float, float, float	Sets the lead synth's equalizer values: low, mid, and high
/instruments/bass/frequency	float	Sets the bass's frequency in Hz
/instruments/bass/volume	float	Sets the bass's volume in dB
/instruments/bass/eq3	float, float, float	Sets the bass's equalizer values: low, mid, and high
/instruments/pad/frequency	float	Sets the pad's frequency in Hz
/instruments/pad/volume	float	Sets the pad's volume in dB
/instruments/pad/eq3	float, float, float	Sets the pad's equalizer values: low, mid, and high

Message address	Arguments	Description
/master/volume	float	Sets the master volume

Before you start adding OSC triggers, you need to properly set Q2Q's sending connection to be that of the synthesizer's *listening* host and port. If the synthesizer is *listening* (a.k.a receiving) on port `12345` from a device with an IP of `192.168.0.30`, set the host and port of the network patch's send connection to those values. If you use Q2Q's `receiving connection`, it will not work! Finally, choose the correct protocol, which will either be `TCP` or `UDP`, depending on the device.

Say you want to build a cue to set the lead_synth's note. Drag an OSC trigger into your cue list, and select it for editing. Set the address to `/instruments/lead_synth/frequency`. Add a float argument, and set it to the desired value.

Say you want to build a cue to control all the instruments' volumes simultaneously. If you want to set them to the same value, you can use wildcards to acheive this by setting the address to `/instruments/*/volume` and the argument to the desired float value. If, however, you want to use different volumes, you can add multiple OSC triggers in the same cue for each different instrument volume.

Projects

Projects are saved in files with the `.q2q` extension. Audio files used in the project are referenced by relative links when possible, so the projects are portable between machines as long as all audio files used are copied along with the project file. Thus, it is recommended to keep the project file and all its audio files organized together in the same folder (and sub-folders), to make transferring between computers easy. The only time Q2Q has to resort to absolute file links is on Windows when referencing files across drives - for example, if your project is at `C:\Users\joe-smith\Documents\Q2Q\SomeProject\SomeProject.q2q`, while an audio file used in the project is at `D:\SomeAudio.wav`.

Exporting Cue Sheets as CSV Files

Project cue sheets can be generated and exported in the CSV ("Comma Separated Values") file format with the menu option `File>Export>Cue sheet to CSV...`. CSV files are a common table-based file format compatible with many programs, such as Excel. Cue sheets generated in this manner contain `Name`, `Duration`, `Triggers`, and `Notes` columns.

Automatic cue naming

You can edit a cue's name by double-clicking on its current name, typing the new name, and pressing enter.

Q2Q will automatically name cues sequentially if you use some special cue names. For example, if you name all your cues `#`, they will display as successive numbers starting with 1, meaning that a cue list of `#`, `#`, `#` will display as `1`, `2`, `3`. Likewise, `$` will be display with a sequentially incrementing alphabet letter, meaning a cue list of `$`, `$`, `$` will be displayed as `A`, `B`, `C`.

You can find a complete description of all possible substitutions below.

You can also specify a project's default cue naming scheme to use for new cue creation. This setting is found in the toolbar, in the section labelled "Default cue name." You can pick one of the recommended options from the dropdown, or specify your own be selecting "custom." New projects will use `$` by default.

Substitutions

#

Substitutes a number, incremented from the last number-like cue. Multiple `#` symbols will pad the number with additional zeros.

$

Substitutes an alphabetical letter, incremented from the last letter-like cue. Multiple `$` symbols in a row will use that number of letters, similar to the zero-padding of the numbers, but with some differences. For example, "B" and "AAAAAB" are considered different cue "numbers", while "1" and "00001" would be considered the equal cue numbers.

#n and $n

Adding a number, n, to the end of a `\#` or `\$` sequence, will increment the cue name by `n`, instead of the default of 1.

{value}

Displays the contents of the curly braces. If the contents are a number or letter, cues after this cue it will starting numbering/lettering from that number or letter, allowing you to skip swaths of numbers/letters.

Examples

Here are some examples of cue list inputs and what their displayed names would be:

Cue list	Displayed names
$, $, $	A, B, C
$, $, $, BA, $, $$	AA, AB, AC, BA, BB, BC
$$5, $$5, $$5	AA, AF, AK, AP
#, #, #	1, 2, 3
###, ###, ###	001, 002, 003
###, ###, ###, {101}, ###, ###	001, 002, 003, 101, 102, 103
{000}, ###5, ###5, ###5, ###5	000, 005, 010, 015, 020

OSC API

Q2Q can be controlled with OSC from other programs or devices that can send OSC messages. You can set the host, port, and protocol that Q2Q will listen for OSC messages over in the `Network Patch` window under the `Receive connection` setting. It is disabled by default.

Q2Q will ignore OSC messages that it does not recognize.

API

Top-level methods

Address	
`/cue/index/{index}/...`	Accesses the methods of a cue at the index `{index}`, where `0` is the first cue in the cue list, `1` is the second, and so on.
`/cue/{name}/...`	Accesses the methods of a cue with the name `{name}`.
`/cue/armed_prev/.../`	Accesses the methods of the cue right before the currently armed one, if applicable
`/cue/armed/...`	Accesses the methods of the currently armed cue.
`/cue/armed_next/...`	Accesses the methods of the cue right after the currently armed one, if applicable
`/panic`	Stops all cues

Cue methods

Address	
`.../go []`	Triggers the cue, as if pressing GO after arming it
`.../arm []`	Arms the cue, as if selecting it with the mouse or arrow keys

Examples

- Sending `/cue/A/go` will trigger the cue named "A"
- Sending `/cue/{B,C}/go` will trigger the cues named "B" and "C" simultaneously
- Sending `/cue/index/9/go` will trigger the 10th cue in the cue list (remember that cues are zero-indexed)
- Sending `/cue/armed_next/arm` will arm the next cue, as if pressing the down arrow key