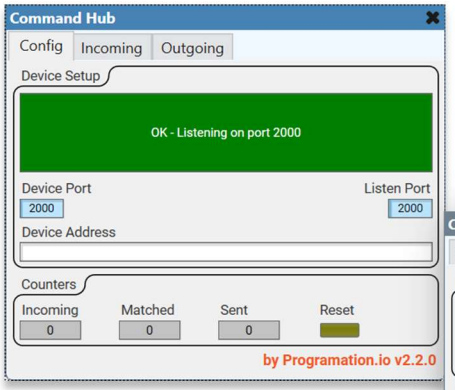
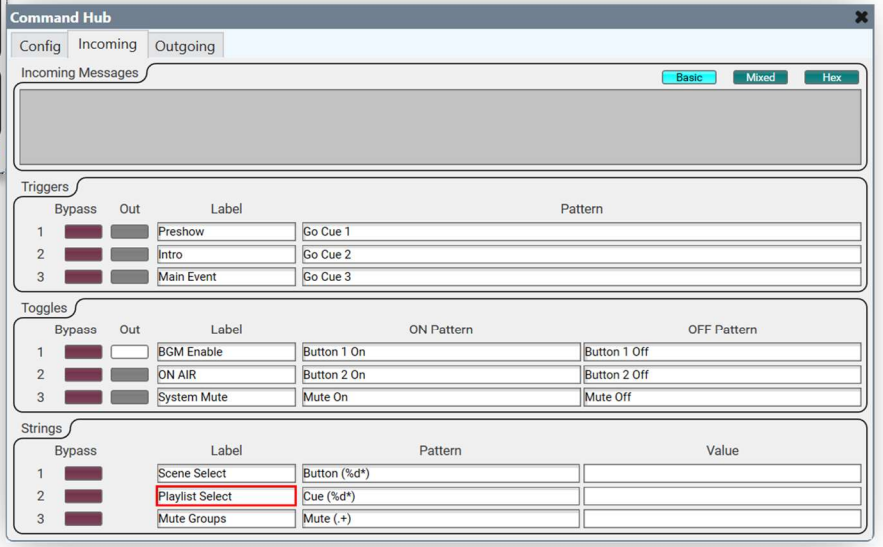
Plugin Guide

COMMAND HUB

Version 2.2.0
February 12, 2026





Programation.io

COMMAND HUB

is a simple yet powerfully flexible external control interface utility. It allows for simple network messages to be received and sent from any external device using UDP, TCP, or HTTP. Custom string patterns and commands can be configured for use with any protocol for any device. This is ideal for getting simple state information from devices.

Command Hub works great with QLab, for example, allowing cues to be sent from a QLab script to activate trigger and toggle controls. You can also easily activate cues from controls within Q-Sys. This opens a world of possibilities for automated playback, advanced audio routing, paging control, and discrete GPIO control.

With the "Companion" app, a Stream Deck button box could also be used to control a Q-Sys Core using your own custom control codes. Virtually any device that can send UDP or TCP commands can now control your core. No API, no programming, no problem. Just define the patterns to match and you are up and running.

The plugin features helpful information displays that shows the last incoming message and outgoing message with the timestamp to help verify that communication is functional with a device. Messages can be viewed in basic ASCII or in hexadecimal, or a mix of both making it easy to deal with more complex protocols.

Incoming messages are matched using patterns, similar to regular expressions. This allows for even greater flexibility for watching for triggers where the message sent may be formatted differently each time or you simply want to catch every incoming message. You can also use patterns to capture text snippits from incoming messages for more comprehensive feedback.

Properties

Communication Mode

a `choice` of UDP, TCP, TCP Server, or HTTP which defines the connection type and connection parameters that will be used to communicate with an external device or allow devices to connect to the plugin

Incoming Triggers

an `integer` between `0` and `100` that defines the number of available trigger buttons that will react to incoming messages

Incoming Toggles

an `integer` between `0` and `100` that defines the number of available toggle buttons that will react to incoming messages

Incoming Strings

an `integer` between `0` and `100` that defines the number of available string patterns to capture parts of incoming messages

Outgoing Triggers

an `integer` between `0` and `100` that defines the number of available trigger buttons that will send an outgoing command message when activated

Outgoing Toggles

an `integer` between `0` and `100` that defines the number of available toggle buttons that will send outgoing messages when the state of the button is changes

Outgoing Strings

an `integer` between `0` and `100` that defines the number of available formatted string messages that will be sent whenever the value changes

Properties

Web Triggers

an `integer` between 0 and 100 that define the number of available trigger buttons that will perform web requests to a remote server when activated

Web Toggles

an `integer` between 0 and 100 that defines the number of available toggle buttons that will perform web requests to a remote server when the state of the button changes

Controls

Status

displays the network connection status information. Reports when the plugin is listening for incoming network messages and reports when a connection is made (or failed to be made) with a device.

Listen Port

an integer knob that specifies the network port to listen to. This is only enabled when the Communication Mode Property is set to UDP, or TCP Server. These are the only modes where unsolicited messages may be received.

Device Port

an integer knob that specifies the network port that outgoing commands will be sent to. This is unavailable when the Communication Mode Property is set to TCP Server.

Device Address

a textbox that specifies the IP address or connection address of the external device where outgoing commands will be sent to. This is unavailable when the Communication Mode Property is set to TCP Server. When the Communication Mode Property is set to HTTP, the Device Address control should be used to specify the URL of the web server the plugin should connect to for web requests. The address will be auto-formatted to be prefixed with "http://" if it is not typed. Only HTTP connections are supported.

Incoming Counter

a text display that indicates the number of incoming network messages received

Matched Counter

a text display that indicates the number of incoming network messages that matched a pattern in the "Incoming" tab. It is possible for a single incoming message to be matched more than once.

Outgoing Counter

a text display that indicates the number of outgoing network messages sent from button presses in the "Outgoing" or "Web Request" tabs. When using a toggle button and both the ON Command and OFF Command are defined, there will be two outgoing messages for a full cycle of the control.

Reset Counters

a trigger button that will zero-out the counter displays

Incoming Message

a text display that shows the last message received by the plugin, regardless whether it was matched or not. This makes it easier to form patterns based on the bytes received, especially if you are not sure of the formatting of the command from the remote device. The timestamp is also displayed to assist in differentiating between duplicate commands. To prevent processing strain, only the first 512 characters of the incoming message is displayed. All of the received characters are considered for pattern matching, however. The format of the message payload is determined by one of the following three modes:

Incoming Message Display Mode Basic

selects the "Basic" mode to display the incoming message. Non-ASCII or non-printable characters are replaced by a single period to more easily count the total number of characters. Tab (\t), carriage return (\r), and line feed (\n) are shown using these short codes.

Incoming Message Display Mode Mixed

selects the "Mixed" mode to display the incoming message. Non-printable characters are replaced with their Lua-formatted escaped hex sequences. e.g. a carriage return and line feed would appear as "\x0D\x0A"

Incoming Message Display Mode Hex

selects the "Hex" mode to display the incoming message. All characters are displayed in the Lua-formatted escaped hex sequence.

Incoming Trigger Bypass

a toggle button that, when enabled, will ignore the associated pattern(s) when an incoming message is received

Incoming Trigger Label

a textbox that holds a friendly name for this trigger

Incoming Trigger Pattern

a textbox that holds the pattern that will be used to compare with incoming messages

Incoming Trigger Button

a trigger button that is activated when an incoming message matched the given pattern

Incoming Toggle Bypass

a toggle button that, when enabled, will ignore the associated pattern(s) when an incoming message is received

Controls

Incoming Toggle Label

a textbox that holds a friendly name for this toggle

Incoming Toggle ON Pattern

a textbox that defines the pattern that, when matched with an incoming message, will set the output toggle button to the ON state

Incoming Toggle OFF Pattern

a textbox that defines the pattern that, when matched with an incoming message, will set the output toggle button the OFF state

Incoming Toggle Button

a toggle button that is set by the associated ON Pattern or OFF Pattern

Incoming String Bypass

a toggle button that, when enabled, will ignore the associated pattern when an incoming message is received

Incoming String Label

a textbox that holds a friendly name for this string

Incoming String Pattern

a textbox that specifies the pattern to use to search against the incoming message

Incoming String Value

a read-only textbox that displays the value captured by the specified pattern

Outgoing Message

a text display that shows the last message sent by the plugin. This is useful for verifying that automatically issued commands are sent correctly. The format of the message payload is determined by one of the following three modes:

Outgoing Message Display Mode Basic

selects the "Basic" mode to display the outgoing message. Non-ASCII or non-printable characters are replaced by a single period to more easily count the total number of characters. Tab (\t), carriage return (\r), and line feed (\n) are shown using these short codes.

Outgoing Message Display Mode Mixed

selects the "Mixed" mode to display the outgoing message. Non-printable characters are replaced with their Lua-formatted escaped hex sequences. e.g. a carriage return and line feed would appear as "\x0D\x0A"

Outgoing Message Display Mode Hex

selects the "Hex" mode to display the outgoing message. All characters are displayed in the Lua-formatted escaped hex sequence.

Outgoing Trigger Bypass

a toggle button that, when enabled, will ignore state changes of the button and will not send any commands

Outgoing Trigger Label

a textbox that holds a friendly name for this trigger

Outgoing Trigger Button (State)

a trigger button that causes the command to be sent

Outgoing Trigger Command

a textbox that holds the command string that will be sent to the external device

Outgoing Toggle Bypass

a toggle button that, when enabled, will ignore state changes of the button and will not send any commands

Outgoing Toggle Label

a textbox that holds a friendly name for this toggle

Controls

Outgoing Toggle Button (State)

a toggle button that issues either the ON Command or OFF Command to be sent

Outgoing Toggle ON Command

a textbox that defines the command string that will be sent when the State Button changes to ON

Outgoing Toggle OFF Command

a textbox that defines the command string that will be sent when the State Button changes to OFF

Outgoing String Bypass

a toggle button that, when enabled, will ignore value changes of the string and will not send any commands

Outgoing String Label

a textbox that holds a friendly name for this trigger

Outgoing String Bypass

a textbox that defines the pattern to use when formatting the provided value to send as a message

Outgoing String Value

a textbox that provides the value or values to format according to the specified pattern. Any change in this field will cause a message to be sent.

Web Request Message

a text display that shows the last message sent by the plugin

Web Request Trigger Bypass

a toggle button that, when enabled, will ignore state change of the button and will not send any commands

Web Request Button (State)

a trigger button or a toggle button that causes the command to be sent. Toggles will follow either the ON line parameters or the OFF line parameters in accordance with the state of the button

Web Request Verb

a combo box that allows for the selection of either “GET”, “PUT”, or “POST” verbs. For simple web requests, use “GET”. This field may not be left blank, otherwise no message will be sent.

Web Request Path

a textbox that specifies the path or the URL to be requested. The path is appended to the Device Address as specified on the Config tab. This field may not be left blank, otherwise no message will be sent. For the root of the site, use a single forward-slash (/).

Web Request Data

a textbox that allows for additional data to be sent with the web request. See the Note below for more information.

Notes

WEB REQUEST FORMATTING

To make Command Hub more helpful and organized, the URL's for web requests are broken up into different parts to allow for easier readability and changeability in the event that a device or site's address or structure changes. Imagine having to change the IP address in every URL because of a device configuration change! Command Hub's organization approach is also clever in the way it consolidates the formatting of the different types of requests. First, let's review the different parts that make a Uniform Resource Locator (URL). Take the example:

```
http://api.mydevice.com/cgi-bin/control.php?id=123
  (1)  (2)      (3)           (4)             (5)
```

There are multiple parts here that each have a specific usage and relevance in how data is encoded and requested.

1) First and foremost is the *protocol* or the *scheme*. It specifies the type of connection that should be used to access the resource. The world wide web primarily uses HTTP and HTTPS but also supports FTP, SFTP, TFTP, mailto, data, and irc. Command Hub only supports HTTP connections without authentication. If a device requires more advanced communication protocols, perhaps it would be wise to develop a custom-built plugin to ensure secure and proper communication.
2) The *sub-domain* is prepended to the main domain. Sub-domains, if any, allow multiple web servers or web services to reside at the same logical domain address. It's a neat way to organize a cohesive web service ecosystem without having to register multiple domains which could be costly; not to mention that this would cause us to run out of domain names much more quickly.
3) The *domain name* is, in short, the address of the web site or service. It's the most memorable part. When dealing with local control of AV devices, most equipment has a web interface to allow for control or configuration. In this case, the address is actually the IP address of the unit.
4) The *path* is a folder structure the requests a specific file through a series of folders on the web server. Sometimes the file extension (shown above as ".php") may not be shown because the web server is configured to assume a default file name or file extension.
5) The *query string* is appended at the end of the string and always begins with the question mark (?). The query string is made up of key-value pairs, joined together with the equal sign (=). Multiple parameters may be joined together using

Notes

the ampersand (&). The query string allows certain information to be passed to the web server as part of the request

to affect the result returned by the server. In the case of controlling AV devices, sometimes these may be commands we want to issue to the device.

When working with a web service or a device with a web interface, the scheme (1), any sub domains (2), and the domain or IP address (3) are the same for every request and never change unless there is a serious or necessary configuration change. These parts of the URL can be entered into the Device Address field of Command Hub to prevent having to duplicate them and possibly change them later.

The path (4) of the URL may or may not be the same for every type of request. Some times for device control, all commands may be passed to the same command router page while other times different pages are used for different categories of functions. Command Hub provides the Path textboxes to hold this piece of information.

When using the “GET” verb, Command Hub will automatically append text in the Data textboxes to the path to make it easy to set up various control commands quickly. This is only used for query strings (5) and Command Hub will automatically add a question mark as part of the request if one is not entered in the Data textbox.

When using the “PUT” or “POST” verbs, Command Hub uses the Data field differently. PUT and POST data in a web request is not transmitted as part of the URL, but instead in the body payload of the request which is not normally seen in an average web browsing experience. Command Hub will transmit the contents of the Data textbox as part of the PUT or POST request that complies with the web standards. The request will not show this data in the Web Request Message display. The Data will be transmitted exactly as it is entered in the Data textbox when the “PUT” or “POST” verb is selected.

Notes

PATTERNS

Command Hub uses the power of Lua "patterns" to match incoming messages. Patterns do not have to exactly match the incoming message. You should carefully consider the various messages that may be received and the patterns that are set to prevent accidental miss-triggering of other lines.

Patterns will only have to partially match the incoming message. Making a pattern as specific as possible ensures that only unique messages will match. For example, the pattern "On" will match both "System Mute On" and "Host Mic On". There is a big difference between those two and a more specific pattern should be used to better differentiate.

Patterns can also contain short-hand codes called character classes to match particular types of characters of unknown values. For example, a four-digit number could be matched using "%d%d%d%d". The character classes are:

.	all characters	%s	space characters
%a	letters	%u	upper case letters
%c	control characters	%w	alphanumeric characters
%d	digits	%x	hexadecimal digits
%l	lower case letters	%z	the character with representation 0
%p	punctuation characters		

In addition, patterns also have special "magic characters" which are:

```
( ) . % + - * ? [ ^ $
```

The character `%´ works as an escape for those magic characters. So, '%.' matches a literal dot; '%%' matches the character `%´ itself.

For more info on patterns, refer to the Lua reference manual.

Notes

WORKING WITH INCOMING STRING VALUES

When attempting to capture part of an incoming message using the String type, parentheses should be used around the section that you want to appear in the Value field. For example, to parse a volume feedback message that may look like:

```
VOL090\r\n
```

a pattern should be used the specifies to begin after the "VOL" and not include the carriage return and line feed characters:

```
VOL(%d+)%s
```
This line reads as "search for VOL and capture one or more numerical digits before whitespace".

The output value, then, will appear as:

```
090
```

Keep in mind that all patterns that are not in Bypass are attempted to be matched against the incoming message. It is possible for more than one pattern to capture values. In some cases, this may be useful as the multiple values could be joined together for other uses. In some cases, this may cause unintended side effects when the value fields are wired directly to DSP blocks. Being more specific with the pattern will help prevent mis-triggers.

Notes

Outgoing String Formatting

Outgoing Strings use patterns not to match against, but for formatting purposes. The pattern allows static parts of messages to be "hard-coded" (set and forget) while the variable values can be more easily changed. Say for example, an integer knob is wired to an Outgoing String's Value field and it should be used to specify an intensity. The pattern may be setup as:

```
INTENSITY=%s%%
```

(keep in mind a double percent sign is the literal usage of a single character)

With any value specified in the Value field, the outgoing message will appear as:

```
INTENSITY=99%
```

Any time the Value field changes, the value will be formatted and transmitted to the external device.

For extremely advanced applications, the Outgoing String can handle multiple variables. Each value should be surrounded by the pipe character, with double pipes between values. The number of placeholders in the pattern must be equal to the number of available values, otherwise an error will be given. For example, to specify an RGB color, the pattern could appear as:

```
R=%s, G=%s, B=%s
```

with a single Value provided as:

```
|241||88||34|
```

The outgoing message would then be sent as:

```
R=241, G=88, B=34
```

Should a value need to contain a pipe character, it must be encoded using its hexadecimal ASCII value "\x7c"

Notes

QLab Interface Example

QLab accepts UDP commands that look like basic OSC messages on UDP port 53535. While OSC is a far more complicated protocol than what is presented here, QLab can also interpret a basic, similar protocol. To trigger a cue, you can set an outgoing command string to `/cue/1/start`. Simply replace the number with the desired cue number.

QLab can also send messages to Command Hub using Network Cues. A Network Patch must first be created which defines the target IP Address and Port. By default, Command Hub uses port 2000, although any port can be specified using the integer knob on the Config tab. Once the Network Patch is configured, you can add a Network Cue to your script and send any message that you define. Set the same message as an incoming pattern and Command Hub will do the rest.

Change Log

V2.2.0 – February 12, 2026

Added Label fields to all inbound & outbound triggers, toggles and strings for human-readable reference

V2.1.0 – November 28, 2022

Standardized the layout with Incoming Trigger and Toggle buttons.

Left justified all text fields for easier reading.

Added support for Incoming and Outgoing Strings.

V2.0.0 – February 23, 2021

Moved the Communication Mode selection to the Properties palette for easier configuration at design time. Added two new options: TCP Server and HTTP. Only UDP and TCP Server modes listen for incoming connections to prevent possible conflict with port bindings of other services on a core.

Add Web Request Trigger and Toggle buttons on the Web Request page. Web Requests have a similar behavior as normal outgoing messages, however they require more configuration parameters.

Removed momentary buttons for simplicity. If a momentary button is required, use a Custom Controls component before the Command Hub plugin and join the two via control wiring.

Improved error handling and connection management. More useful error messages are included as well as auto-reconnection if a connection could not be established automatically.

Change Log

V1.1.0 – October 8, 2020

Added support for sending and receiving escaped characters. Incoming messages are still allowed to match using magic patterns in addition to escaped characters.

Changed "Basic" view mode to show escape sequences of Tab, Carriage Return, and Line Feed as '\t', '\r', and '\n'. Mixed mode and Hex mode will still show the hexadecimal equivalents of these characters.

V1.0.0 – June 16, 2020

Original release