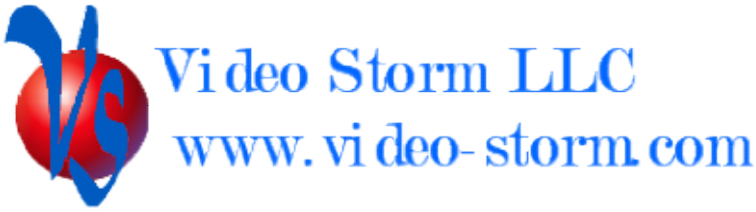


# Lumi Cloud API

Revision 1.2
4/8/2023

## Overview

Lumi Cloud provides several control APIs

- Simple HTTPS GET protocol to Lumi-tv.com (Webhook API)
- Direct loopback SOCKET API for apps on Lumi TV devices
- SSH socket for local Ethernet control
- VNC for local screen sharing connection
- ADB for local Ethernet control

Video Storm provides drivers for many control systems and/or applications. This document is intended for integrators who wish to access the our APIs directly or write their own drivers/applications.

# Webhook API

Setup:
Users must have an active LumiCloud PRO account in order to use the API.

1. Go to lumi-tv.com, setup an account, and register your LumiTV devices
2. Click on Setup-Devices in your lumi-tv.com account. Verify this page shows all your devices. NAME each device with a unique name.
3. Click the "Create Browser Device" button to generate a type = Browser device. The "Token" field is the security token you will use for 3rd party control of your devices.

Control:
Webhook API allows one way control of your connected Lumi TV devices.

Protocol:
**The control protocol** is a single HTTPS GET transfer (you MUST use HTTPS, not HTTP!)

URL: https://lumi-tv.com/console/server/rcontrol.php?PARAMETERS

Parameters:
cmd= Lumi protocol command string (single command per https get)
dev= String for NAME of device to control, or 000 for all devices
token= Your Browser device TOKEN

All parameters must be properly URI encoded.

**The Query protocol** is a single HTTPS GET transfer (you MUST use HTTPS, not HTTP!)

URL: https://lumi-tv.com/console/server/rcontrol.php?PARAMETERS

Parameters:
cmd= String given the command (see below)
dev= String for Device ID to query (only used with NetPlay Cloud)
token= Your Browser device TOKEN

All parameters must be properly URI encoded.

Commands:

QUERY:

Read SplashTiles devices and screens
Return data:  Return data type is JSON

Format is as follows:
{
“devices” : [list of devices],
“scripts”: [list of macros]
}

NPQUERY:

Read back last data from Lumi Cloud MQTT
Return data:  Return data type is JSON

Format is as follows:
{
“TX”:  String of return data
“RX” : String of return data
}

# Socket API

The loopback socket API allows direct communication between APPS INSTALLED on Lumi TV and the priviledged system application (LumiLauncher).

TCP socket
Port 9094
Will only accept connections on the loopback (local) interface

Authentication: none  (local connections only)

Bidirectional communication using Lumi Protocol

# SSH API

This secure API allows encrypted connection via the local network adapters (wired, wifi)

TCP socket, SSH & SFTP
Port 22

Authentication: Requires your SSH public key loaded via device provisioning

Shell and SFTP access via local network.

# ADB API

This secure API allows encrypted connection via the local network adapters (wired, wifi)

TCP socket, ADB
Port 5555

Authentication: Requires your ADB public key loaded via device provisioning

Device ADB shell access for app installation, admin tasks, etc.  Note this is user “shell”, since root access is not available on Lumi TV

# VNC API

This secure API allows encrypted screen sharing & remote control via the local network adapters (wired, wifi)

TCP socket, SSH
Port 22

Authentication: Requires your SSH public key loaded via device provisioning

Screen sharing and remote control via VNC

# Lumi Protocol

Lumi protocol is a simple ascii text based bidirectional UART style interface for accessing the functionality of the priviledged system application (LumiLuancher)

All commands and responses are terminated by carriage return (/r or ascii 0x0d)

Multiple commands can be sent without waiting for responses. All commands execute immediately and in order.

**Available commands:**

| | |
|---|---|
| QSTATVER | Get current Lumi TV version |
| QCECON | Turn on attached TV via CEC |
| QCECSEL | Set attached TV input select to Lumi TV port |
| QCECOFF | Turn off attached TV via CEC |
| QSETLED int | Set Lumi TV LED display |
| QAPP pkgname | Launch App with given package name<br>(com.example.app) |
| QLAUNCH intenturl | Execute the android intent URL<br>(see appendix for examples) |
| QDREAM | Run screen saver now |
| SHELL shellcmd | Run given shell command<br>(shellcmd may contain spaces etc)<br>Note shell access can be disabled in provisioning |
| QSTATS | Instruct LumiTV to post stats history to lumi-tv.com<br>(can use webhook api to retrieve) |
| QSCREEN | Instruct LumiTV to post screen shot to lumi-tv.com<br>(can use webhook api to retrieve) |
| QSTATSEND | Instruct LumiTV to post current status to MQTT<br>(can use webhook api to retrieve) |
| QWEBHOOK url | Execute given Webhook URL |

| | |
|---|---|
| QRESTART | Reboots Lumi TV NOW |
| QUPDATEPV | Check for provisioning profile update NOW |
| QUPDATEFW | Check for ota update NOW |
| QDOGSTART | Start Watchdog for this interface |
| QDOGSTOP | Stop Watchdog for this interface |
| QDOGKICK | Kick Watchdog: If dog started must send this<br>Command every 15min to avoid reboot<br>(note this is additional to the normal hardware watchdog)<br>(intended to extend watchdog to external apps / processes) |
| QPLAYLx file | Play given local media (image, video)<br>x = 0 (web) 1 (video) 2 (image) 3 (audio) |
| QPLAYUx url | Show the given URL<br>x as above |
| QPLAYD dur dir | Slideshow image files in local directory<br>dur = duration in seconds |
| QRESUME | Resume running schedule<br>(call this after a QPLAY command to immediately resume schedule) |
| QSENDKEY int | Execute the given keyevent<br>Defined at https://developer.android.com/reference/android/view/KeyEvent.html |
| QSENDTEXT text | Send the keyevents corresponding to the given text |
| QSIRPULSE hexcode | Send the given IR hex code<br>(IRUSB required)<br>(can include id=xxxxxxx if multiple IRUSB devices) |

# Android Intents (for QLAUNCH)

QLAUNCH allows launching apps as well as sending specific data to other apps. This can also be used to configure apps which do not support managed profiles.

QLAUNCH commands use standard Android Intent URLs with the following format:

QLAUNCH android-app://PACKAGE_NAME#Intent;component=COMPONENT_TO_LAUNCH;end

Both the PACKAGE_NAME and COMPONENT_TO_LAUNCH names are required. You can find these by looking at the ADB log output when manually starting any app. You can filter the log with cmp= to find these (after manually launching the app using the gui).

## Deep links:

Lumi supports several methods to link directly to programs or channels within apps. This is known as 'Deep linking'.

1. Using the web site URL syntax
2. Using the 'intent://' URL syntax
3. Using the 'android-app://' URL syntax

### Web site URL syntax (try this method first):

This method is the most straightforward. You simply find the web url syntax for your title, video, or show. For Youtube, examples are:

- Video: QLAUNCH http://www.youtube.com/watch?v=VIDEOID
- Playlist: QLAUNCH http://www.youtube.com/playlist?list=PLAYLISTID

Remember to include the 'www' in the address!

### 'intent://' URL syntax:

This method is better if your device can't figure out which app to send the web url to. You are adding the package name as well as the web url. For Youtube, examples are:

- Video: QLAUNCH intent://www.youtube.com/watch?v=VIDEOID#Intent;package=com.google.android.youtube.tv;scheme=https;end

- Playlist: QLAUNCH intent://www.youtube.com/playlist?list=PLAYLISTID#Intent;package=com.google.android.youtube.tv;scheme=https;end

Remember to include the 'www' in the address! Also keep in mind the package name will be different on Amazon devices.

**'android-app://' URL syntax:**

This method is just a different format of the 'intent' type. The following examples are for Netflix which requires extra data (see below):

- Play Video: QLAUNCH android-app://com.netflix.ninja/http/www.netflix.com/watch?VIDEOID#Intent;package=com.netflix.ninja/.MainActivity;S.source=30;end

Keep in mind the package and component names will be different on Amazon devices.

**Extra Data syntax:**

Some apps require sending 'extra data' in order to select content directly. Both intent types and android-app types support this.

QLAUNCH android-app://PACKAGE_NAME/DATA#Intent;component=COMPONENT_TO_LAUNCH;EXTRADATA;end
QLAUNCH intent://DATA#Intent;package=PACKAGE_NAME;component=COMPONENT_TO_LAUNCH;EXTRADATA;end

The format of EXTRADATA is

- i.VAR_NAME=VALUE;  (integers)
- S.VAR_NAME=VALUE;  (Strings)

DATA is usually a web url (http://host/path)

As there is no standard covering what values can be passed to different apps, Google is the best way to find information on what values are needed.