

# External Control Guide

- [SSDP \(Simple Service Discovery Protocol\)](#)
- [External Control Service Commands](#)
  - [General ECP commands](#)
    - [query/apps](#)
    - [query/active-app](#)
    - [keydown/key](#)
    - [keyup/key](#)
    - [keypress/key](#)
    - [launch/appID](#)
    - [install/appID](#)
    - [query/device-info](#)
    - [query/icon/appID](#)
    - [input](#)
  - [Roku TV ECP commands](#)
    - [query/tv-channels](#)
    - [query/tv-active-channel](#)
    - [launch/tvinput.dtv](#)
- [Input Command Conventions](#)
  - [Sensor Input Values](#)
  - [Accelerometer](#)
  - [Orientation](#)
  - [Gyroscope](#)
  - [Magnetometer](#)
  - [Touch and Multi-Touch](#)
  - [Additional Input Values](#)
- [External Control Protocol Examples](#)
  - [query/apps Example](#)
  - [query/active-app Examples](#)
  - [keypress Example](#)
  - [keyup/keydown Example](#)
  - [launch Examples](#)
    - [launch parameters for the Channel Store app \(channel ID 11\)](#)
    - [launch parameters for the Roku TV Tuner app \(channel ID tvinput.dtv\)](#)
  - [query/icon Example](#)
  - [input Examples](#)
  - [query/device-info Example](#)
  - [query/tv-channels Example](#)
  - [query/tv-active-channel Example](#)
- [Implementing Deep Linking in a Channel](#)
- [Keypress Key Values](#)
- [Security Implications](#)
- [Example Programs](#)
- [DIAL \(Discovery and Launch\)](#)

The External Control Protocol (ECP) enables a Roku device to be controlled over a local area network by providing a number of external control services. The Roku devices offering these external control services are discoverable using SSDP (Simple Service Discovery Protocol). ECP is a simple RESTful API that can be accessed by programs in virtually any programming environment.

## SSDP (Simple Service Discovery Protocol)

SSDP is an industry IETF standard network protocol for discovery of local area network services. Roku devices advertise their external control services using the multicast SSDP so that programs can discover the IP address of Roku devices in the area. There is a standard SSDP multicast address and port (239.255.255.250:1900) used for local area network communication. The Roku device responds to M-SEARCH queries on this IP address and port.

To query for a Roku device IP address, send the following HTTP request to 239.255.255.250 port 1900:

```
M-SEARCH * HTTP/1.1
Host: 239.255.255.250:1900
Man: "ssdp:discover"
ST: roku:ecp
```

Note there *must* be a blank line at the end of the file above. If you put the above request into a file such as roku\_ecp\_req.txt, you can issue the following command on most Linux machines to test the request:

```
% ncat 239.255.255.250 1900 < roku_ecp_req.txt
```

If you view the response using Wireshark, and filter on port 1900, you can see the Roku device response (Ncat has trouble receiving multicast traffic, so viewing the response using Ncat does not work). The response has the following format:

```
HTTP/1.1 200 OK
Cache-Control: max-age=300
ST: roku:ecp
Location: http://192.168.1.134:8060/
USN: uuid:roku:ecp:P0A070000007
```

If you get a 200 status response, the Location header is valid. You can parse out the URL for the Roku device external control services from the Location header. The Roku device serial number is contained in the USN line after uuid:roku:ecp. Note that if there are multiple Roku devices in your local area network, you will get multiple responses. Your program could keep a map of USNs to location URLs, and allow the user to select which Roku device on the network to control. We recommend you let the user assign names to the USNs.

When parsing headers in the response, in accordance with the UPnP Device Architecture specification, field names should not be treated as case sensitive. That means that, for example, the Location header may begin with either "Location:" or "LOCATION:" or "location:", and so forth.

Please note the Cache-Control header. Roku devices multicast NOTIFY messages periodically (approximately every 20 minutes). It is

safe to assume the unit is no longer available if you have not received a new NOTIFY message before the Cache-Control max-age time expires.

## External Control Service Commands

The external control services provided by ECP are included in a simple RESTful API accessed using HTTP on port 8060. Once you have the Roku device IP address, you can issue the following external control service commands to the Roku device.

### General ECP commands

Command	Description
<b>query/apps</b>	Returns a map of all the channels installed on the Roku device paired with their application ID. This command is accessed using an HTTP GET.
<b>query/active-app</b>	Returns a child element named 'app' that identifies the active application, in the same format as 'query/apps'. If no application is active, such as when the user is in the homescreen, the element only contains "Roku". If a screensaver is active, a second element will be included containing "screensaver". If the screensaver is an application-provided or plug-in screensaver, the same information is provided as 'query/apps'. If the screensaver is active, but is not running (such as due to system limitations), the screensaver element contains "black". This command is accessed using an HTTP GET.
<b>keydown/key</b>	Equivalent to pressing the remote control key identified after the slash. This command is sent using an HTTP POST with no body.
<b>keyup/key</b>	Equivalent to releasing the remote control key identified after the slash. This command is sent using an HTTP POST with no body.
<b>keypress/key</b>	Equivalent to pressing down and releasing the remote control key identified after the slash. You can also use this command, and the keydown and keyup commands, to send keyboard alphanumeric characters when a keyboard screen is active, as described in <a href="#">Keypress Key Values</a> . This command is sent using an HTTP POST with no body.
<b>launch/appID</b>	<p>Launches the channel identified by appID. You can follow the appID with a question mark and a list of URL parameters to be sent to the application as an associative array, and passed to the RunUserInterface() or Main() entry point. This command is sent using an HTTP POST with no body.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p> The launch command should not be used to implement deep-linking to an uninstalled channel, because it will fail to launch uninstalled channels. Use the install command instead for uninstalled channels.</p> </div>
<b>install/appID</b>	Exits the current channel, and launches the Channel Store details screen of the channel identified by appID. You can follow the appID with a question mark and a list of launch parameters to be sent to the application as an associative array, and passed to the RunUserInterface() or Main() entry point. If launch parameters are given, the channel is launched immediately after the user installs the channel, and deep-links to content provided in the launch parameters. This command is sent using an HTTP POST with no body.
<b>query/device-info</b>	Retrieves device information similar to that returned by roDeviceInfo. This command is accessed using an HTTP GET.
<b>query/icon/appID</b>	<p>Returns an icon corresponding to the application identified by appID. The binary data with an identifying MIME-type header is returned. This command is accessed using an HTTP GET.</p> <p>Example:</p> <pre>GET /query/icon/12</pre>
<b>input</b>	<p>Sends custom events to the current application. It takes a user defined list of name-value pairs sent as query string URI parameters. The external control server places these name-value pairs into an associative array, and passes them directly through to the currently executing channel script using a Message Port attached to a created roInput object. <a href="#">Input Command Conventions</a> includes detailed recommendations on how to pass your data. Messages of type roInputEvent have a GetInfo() method that will obtain the associative array. The arguments must be URL-encoded. This command is sent using an HTTP POST with no body.</p> <p>Example:</p> <pre>POST /input?acceleration.x=0.0&amp;acceleration.y=0.0&amp;acceleration.z=9.8</pre>

### Roku TV ECP commands

Roku TV devices additionally support the following external control services.

Command	Description
<b>query/tv-channels</b>	Returns information about the TV channel / line-up available for viewing in the TV tuner UI.
<b>query/tv-active-channel</b>	Returns information about the currently tuned TV channel.
<b>launch/tvinput.dtv</b>	<p>Launch the TV tuner UI.</p> <p>Can optionally be passed a TV channel parameter of the form "ch=1.1" to tune to the specified TV channel number.</p>

## Input Command Conventions

As the firmware simply marshals the arguments to the **input** command and passes them to the channel script, the forms below compose a conventional way to communicate input from several common input device types.

### Sensor Input Values

### Sensor input values

There are four sensor input values to report: accelerometer, orientation, gyroscope (rotation), and magnetometer (magnetic). All except orientation are vectors in a cartesian coordinate system relative to the device in its default orientation:

- +x = to the right of the front face of the device (usually the short side)
- +y = to the top of the front face of the device (usually the long side)
- +z = out of the front face of the device (toward the viewer)

The orientation coordinate system is relative to the point on the surface of the Earth between the device and the center of the Earth:

- +x = east
- +y = north
- +z = towards the center of the Earth (down)

The type in all such cases is a string representation of a signed floating point number, with or without an explicit decimal, and with or without a signed integer exponent following the letter E. A missing decimal will be presumed after the rightmost present digit, and a missing exponent will be presumed 0.

### Accelerometer

indicates: acceleration in each dimension relative to free fall

units: meters/sec<sup>2</sup>

names: acceleration.x, acceleration.y, acceleration.z

### Orientation

indicates: angular displacement from flat/level and true (or magnetic?) north.

units: radians

names: orientation.x, orientation.y, orientation.z

notes: Accurate indication of this is not generally possible without correlation with other sensors or assumptions. Devices make assumptions to flip the display, for example, that assume that the device is usually not moving (much) so that all force is simply opposed to gravity, and that can be assumed to be the "up" direction. Deviation from magnetic north depends on a magnetometer, and deviation from true north also depends on geolocation.

### Gyroscope

indicates: angular rotation rate about each axis using the right hand rule for sign

units: radians/sec

names: rotation.x, rotation.y, rotation.z

### Magnetometer

indicates: magnetic field strength

units: micro-Tesla

names: magnetic.x, magnetic.y, magnetic.z

### Touch and Multi-Touch

Touch and Multi-Touch commands take the same form. The resource is the same "input" as all other generic input commands.

Each action is decomposed to an argument in each dimension (of 2, x and y with the same orientation as for the sensor inputs, with origin in lower left). There is an additional "op" argument which can specify down, up, press (down and up), move, or cancel. Each input is also qualified with a pointer id that indicates the initial order of down touches in a multi-touch gesture.

Several such points can be specified in a single POST, especially a move, but a full triad of x, y, and op arguments should be sent, and expected for each point, within a POST that contains any of them.

### Additional Input Values

Other information you might want to pass using the **input** command may include:

- sensor accuracy
- geolocation (from GPS)
- device-provided derivations of above sensor readings, for example "shake" from accelerometer, or "pinch" from multi-touch

## External Control Protocol Examples

The following are some example ECP commands sent via the curl command.

### query/apps Example

The following command gets a list of channels and their corresponding app ids.

```
$ curl http://192.168.1.134:8060/query/apps
<apps>
  <app id="11">Roku Channel Store</app>
  <app id="12">Netflix</app>
  <app id="13">Amazon Video on Demand</app>
  <app id="14">MLB.TV®</app>
  <app id="26">Free FrameChannel Service</app>
  <app id="27">Mediafly</app>
  <app id="28">Pandora</app>
</apps>
```

### query/active-app Examples

The query/active-app command if the user is in the homescreen.

```
$ curl http://192.168.1.134:8060/query/active-app
<active-app>
  <app>Roku</app>
</active-app>
```

```
</active-app>
```

The query/active-app command if the user is in the homescreen but the default screensaver is active.

```
$ curl http://192.168.1.134:8060/query/active-app
<active-app>
  <app>Roku</app>
  <screensaver id="55545" type="ssvr" version="2.0.1">Default screensaver</screensaver>
</active-app>
```

The query/active-app command if the user is in the Netflix app.

```
$ curl http://192.168.1.134:8060/query/active-app
<active-app>
  <app id="12" type="appl" version="4.1.218">Netflix</app>
</active-app>
```

The query/active-app command if the user is in the Roku Media Player with an active screensaver.

```
$ curl http://192.168.1.134:8060/query/active-app
<active-app>
  <app id="2213" type="appl" version="4.1.1507">Roku Media Player</app>
  <screensaver id="5533" type="ssvr" version="1.1.1">Roku Digital Clock</screensaver>
</active-app>
```

### keypress Example

The following command simulates a user hitting the "Home" button

```
$ curl -d '' http://192.168.1.134:8060/keypress/home
```

### keyup/keydown Example

The following commands move the cursor to the far left by holding down the Left key for 10 seconds

```
$ curl -d '' http://192.168.1.134:8060/keydown/left
$ sleep 10
$ curl -d '' http://192.168.1.134:8060/keyup/left
```

### launch Examples

The following command will launch the dev app on the box. The simplevideoplayer app that comes with the SDK will process the "url" and "streamformat" parameters and launch the roVideoScreen to play the passed in video. We assume simplevideoplayer is installed as the side-loaded developer application.

```
$ curl -d '' 'http://192.168.1.134:8060/launch/dev?streamformat=mp4&url=http%3A%2F%2Fvideo.ted.com%2Ftalks%2Fpodcast%2FvilayanurRamachandran_2007_480.mp4'
```

The following command will launch the dev app on the box. The launchparams app that comes with the SDK will process the "contentID" and "options" parameters and display them on a SpringBoard page. We assume launchparams is installed as the side-loaded developer application. This technique is a useful way to create "clickable" ads that launch a springboard page for a particular title in your channel. Roku now supports clickable ads on the home screen as well.

```
$ curl -d '' 'http://192.168.1.134:8060/launch/dev?contentID=my_content_id&options=my_options'
```

### launch parameters for the Channel Store app (channel ID 11)

The Channel Store app (channel ID 11) can be passed a "contentID" parameter with the channel ID of a target application. The following command will launch the channel store app (11) on the box with a contentID equal to 14 (the MLB app). You can get the plugin ID for your app using the /query/apps example above, which returns the installed apps on a Roku player. (You should test that the channel is installed before using the launch command, and use the install command for uninstalled channels). This technique would be useful in creating clickable ads in a free "Lite" version of a paid app. When a user clicks on the ad, the channel store page to purchase the full version could be launched.

```
$ curl -d '' 'http://192.168.1.134:8060/launch/11?contentID=14'
```

### launch parameters for the Roku TV Tuner app (channel ID tvinput.dtv)

The TV Tuner app can be launched and can optionally be passed a "ch" parameter with the channel number to tune to. The following command will launch the TV tuner UI and display channel 1.1 (assuming that channel is available).

```
$ curl -d '' 'http://192.168.1.134:8060/launch/tvinput.dtv?ch=1.1'
```

### query/icon Example

This following command will return the icon for the channel with id 12 (Netflix). The response will be raw binary picture data, after HTTP headers, including one with the MIME type of the picture data.

```
$ curl 'http://192.168.1.134:8060/query/icon/12' > img.png
```

### input Examples

The following command passes three components of acceleration through to the channel app. All query string parameters are passed to the currently running app. The remote app and the currently running app just need to agree on the query string parameters and any communication can be developed.

```
$ curl -d '' 'http://192.168.1.134:8060/input?acceleration.x=0.0&acceleration.y=0.0&acceleration.z=9.8'
```

The following command indicates that a touch at the given x and y has touched down on the screen.

```
$ curl -d '' 'http://192.168.1.134:8060/input?touch.0.x=200.0&touch.0.y=135.0&touch.0.op=down'
```

### query/device-info Example

Below is an example query/device-info request and response.

```
$ curl 'http://192.168.1.134:8060/query/device-info'
<device-info>
  <udn>015e5108-9000-1046-8035-b0a737964dfb</udn>
```

```

<serial-number>1GU48T017973</serial-number>
<device-id>1GU48T017973</device-id>
<vendor-name>Roku</vendor-name>
<model-number>4200X</model-number>
<model-name>Roku 3</model-name>
<wifi-mac>b0:a7:37:96:4d:fb</wifi-mac>
<ethernet-mac>b0:a7:37:96:4d:fa</ethernet-mac>
<network-type>ethernet</network-type>
<user-device-name/>
<software-version>7.00</software-version>
<software-build>09021</software-build>
<secure-device>true</secure-device>
<language>en</language>
<country>US</country>
<locale>en_US</locale>
<time-zone>US/Pacific</time-zone>
<time-zone-offset>-480</time-zone-offset>
<power-mode>PowerOn</power-mode>
<developer-enabled>true</developer-enabled>
<keyed-developer-id>70f6ed9c90cf60718a26f3a7c3e5af1c3ec29558</keyed-developer-id>
<search-enabled>true</search-enabled>
<voice-search-enabled>true</voice-search-enabled>
<notifications-enabled>true</notifications-enabled>
<notifications-first-use>>false</notifications-first-use>
<headphones-connected>>false</headphones-connected>
</device-info>

```

### query/tv-channels Example

Below is an example of the Roku TV query/tv-channels response.

```
$ curl 'http://192.168.1.134:8060/query/tv-channels'
```

```

<tv-channels>
  <channel>
    <number>1.1</number>
    <name>WhatsOn</name>
    <type>air-digital</type>
    <user-hidden>>false</user-hidden>
  </channel>
  <channel>
    <number>1.3</number>
    <name>QVC</name>
    <type>air-digital</type>
    <user-hidden>>false</user-hidden>
  </channel>
</tv-channels>

```

### query/tv-active-channel Example

Below is an example of the Roku TV query/tv-active-channel response.

```
$ curl 'http://192.168.1.134:8060/query/tv-active-channel'
```

```

<tv-channel>
  <channel>
    <number>14.3</number>
    <name>getTV</name>
    <type>air-digital</type>
    <user-hidden>>false</user-hidden>
    <active-input>true</active-input>
    <signal-state>valid</signal-state>
    <signal-mode>480i</signal-mode>
    <signal-quality>20</signal-quality>
    <signal-strength>-75</signal-strength>
    <program-title>Airwolf</program-title>
    <program-description>The team will travel all around the world in order to shut down a global crime ring.</program-description>
    <program-ratings>TV-14-D-V</program-ratings>
    <program-analog-audio>none</program-analog-audio>
    <program-digital-audio>stereo</program-digital-audio>
    <program-audio-languages>eng</program-audio-languages>
    <program-audio-formats>AC3</program-audio-formats>
    <program-audio-language>eng</program-audio-language>
    <program-audio-format>AC3</program-audio-format>
    <program-has-cc>true</program-has-cc>
  </channel>
</tv-channel>

```

## Implementing Deep Linking in a Channel

One of the most common ways that channel developers encounter ECP is when implementing deep linking. Deep linking on Roku is used by the Roku search service to launch titles as well as to launch channels and play specific content in response to the user clicking

on an ad in the Roku home screen. Deep linking uses the optional parameters that are sent via the External Control Protocol from an ad or the search engine to the entry point of the channel BrightScript code. Typical BrightScript code to implement deep linking looks like this:

#### Deep Linking BrightScript Code

```
Function Main(args as Dynamic) as void
```

```
    if (args <> invalid)
```

```
        contentID = args.contentID
```

```
        'Call the service provider API to look up the content details

```

```

    Call the service provider API to look up the content details,
    'or pull the right data from the feed, etc. for this contentID
    '...
end if
End Function

```

The standard for deep linking parameters enforced by Roku to support ads or universal search is that channels use the following parameters:

Parameter	Description	Possible Values
contentID	Partner defined unique identifier for a specific piece of content	contentID=12345, and so forth
mediaType	Parameter to give context to the type of contentID passed.	"series", "episode", "movie", "shortform", and "live" There is also "track" and "playlist" for music but Roku does not intend on supporting those in the OS but you could use them from an Ad.

When implementing deep-linking to a channel, you should first test if the channel is installed or not, because the launch command will not launch uninstalled channels. The following shows how this should be done in pseudo-code:

```

if IsChannelInstalled(channelID) then
  ECP_Command("launch/channelID?contentID=deep_Link_param&mediaType=deep_Link_content_type_param)
else
  ECP_Command("install/channelID?contentID=deep_Link_param&mediaType=deep_Link_content_type_param)
end if

```

Developers can test deep linking by invoking the appropriate ECP command from the command line. For example, the install command below launches the side loaded channel on the Roku device on IP address 192.168.1.114 and passes the content ID 13234.

```
curl -d '' 'http://192.168.1.114:8060/launch/dev?contentID=13234&MediaType=series'
```

This example shows using the install command for uninstalled channels:

```
curl -d '' 'http://192.168.1.114:8060/install/8378?contentid=MV005011860000&MediaType=movie'
```

## Keypress Key Values

When the current screen on the Roku box includes an on-screen keyboard, any keyboard character can be sent via the keyup, keydown, and keypress commands. The key parameter can either be a key name, such as the name of a button on a remote control, or a printable character value specified with the prefix "Lit\_".

Printable ASCII character code values can be transmitted "as-is" with the "Lit\_" prefix. For example, you can send a 'r' with "Lit\_r". In addition, any UTF-8 encoded character can be sent by URL-encoding it. For example, the euro symbol can be sent with "Lit\_%E2%82%AC".

There are even some keys you can send that are not available on any physical remote. *Enter* is for completing keyboard entry fields, such as search fields (it is *not* the same as *Select*). *Search* is useful for short-cutting directly to search screens.

The following are the key names that are recognized by ECP:

```

Home
Rev
Fwd
Play
Select
Left
Right
Down
Up
Back
InstantReplay
Info
Backspace
Search
Enter

```

Some Roku devices, such as Roku TVs, also support:

```

VolumeDown
VolumeMute
VolumeUp

```

Roku TV devices also support changing the channel when watching the TV tuner input:

```

ChannelUp
ChannelDown

```

Roku TV devices also support keys to set the current TV input UI:

```

InputTuner
InputHDMI1
InputHDMI2
InputHDMI3

InputHDMI4
InputAV1

```

Example: On the on-screen keyboard, the string 'roku' can be sent via the following commands:

```

$ curl -d '' http://192.168.1.134:8060/keypress/Lit_r
$ curl -d '' http://192.168.1.134:8060/keypress/Lit_o
$ curl -d '' http://192.168.1.134:8060/keypress/Lit_k
$ curl -d '' http://192.168.1.134:8060/keypress/Lit_u

```

## Security Implications

Note that with the launch command, anyone could write a program that could pass arbitrary parameters to your BrightScript channel. It is important that you validate any parameters that are passed to your program, and check what they may do to your program flow. The

launch command can be very powerful to provide all kinds of interaction between network devices and your program. We envision catalogs browsed on the Internet that could instantly be watched in your channel on the Roku. The door is open to many creative uses.

But if you prefer to shut this door on your channel, you can choose to not process any passed parameters. This will mean external control programs could launch your channel, but they could not change the program flow within your channel.

## Example Programs

The SDK includes a sample External Control Protocol application that requires only glibc to compile. The program is self contained in the `/examples/rokuExternalControl.c` file of the SDK. You can compile and run it with the following commands:

```
% cd SDK_directory
% gcc ./examples/rokuExternalControl.c -o rokuExternalControl
% ./rokuExternalControl
```

On Windows, it can be compiled with the following line:

```
% cl /D "WIN32" rokuExternalControl.c
```

The program first uses SSDP to query for Roku devices in the local area network. The first Roku device that responds is the one to which commands are sent. All requests and responses are printed to stdout so that you can easily follow what the program is doing. The program next moves the cursor to the home screen and highlights the "Netflix" program. It does this by sending the Home key command, and then holding down the left key for twelve seconds. Then it sends the Right key three times. After keeping the "Netflix" program highlighted for five seconds, the program launches the simplevideoplayer application with url and streamformat as keys in the associative array passed to the Main() entry point. The simplevideoplayer application immediately launches the roVideoScreen when launched with an associative array containing valid url and streamformat keys.

With ECP, you have complete control of your Roku device over the network. We can't wait to see what kind of solutions our developer community can create. The sample C program can be quickly modified to run in a variety of environments including Firefox, IE, and other browser plugins, iPhone, iPad, and other mobile device environments.

The SDK also includes a couple of Java applications. There is an Android remote application in `examples/source/ecp_client/android_remote`, and a simple application to find Roku devices on the local area network in `examples/source/ecp_client/Roku_Finder`.

## DIAL (Discovery and Launch)

The Roku platform supports the DIAL (Discovery and Launch) protocol. DIAL is a simple network protocol for discovering first screen devices, and applications from a second screen (such as a mobile iOS or Android application,) and for launching first screen applications on the first screen device from the second screen app. In the context of the Roku platform, the first screen device is the Roku device itself. A first screen application is a DIAL-aware channel installed on the Roku device. Complete details of the DIAL specification can be found here:

<http://www.dial-multiscreen.org/dial-protocol-specification>.

Many current Roku developers are familiar with the Roku external control protocol (ECP) which includes functionality similar to DIAL. An experienced Roku developer may thus fairly ask the question "why do I need DIAL?" One reason is that you may already have a DIAL based second screen implementation for use with other platforms. DIAL support on the Roku platform means that you don't need to add a second protocol to your current application for discovery and launch.

The [Roku DIAL SDK](#) contains detailed documentation of Roku DIAL support, as well as BrightScript, Android, and iOS sample applications. In DIAL parlance, the BrightScript sample is the first screen application, and the Android and iOS apps are the second screen applications. These sample applications should help you get started with your own Roku DIAL support.

## 5 Comments



**Anonymous**

How about sending a literal space char? `POST /keypress/Lit_%20 HTTP/1.1` doesn't seem to work.



**Anonymous**

"\ " im sure with the escape char



**Anonymous**

URLEncode characters and it works great. `URLEncode(" ") == "+"`



**Anonymous**

Requests via i.e. javascript XMLHttpRequest appearantly don't work as this is a cross-domain request and there is no way to control [Access-Control](#) and/or [Access-Control-Allow-Origin](#) headers. Sigh, back to the drawing board for me then...



**Anonymous**

@Anonymous with cross-domain issues: I get around this by using a Raspberry Pi to serve up a mobile-optimised site that uses server-side Python to call the Roku (and other AV components controlled over HTTP)