Roku

The External Control Guide

Roku Streaming Player Version 4.1

Table of Contents

1.0 Overview

The External Control Protocol enables the Roku to be controlled via the network. The External Control Service is discoverable via SSDP (Simple Service Discovery Protocol). The service is a simple RESTful API that can be accessed by programs in virtually any programming environment.

2.0 SSDP (Simple Service Discovery Protocol)

SSDP is an industry IETF standard network protocol for the discovery of network services. Roku boxes will advertise their External Control Protocol through SSDP so that programs can discover the IP address of Roku's in the area by using the multicast ssdp protocol. There is a standard SSDP multicast address and port that is used for local network communication. The Roku responds to M-SEARCH queries on this ip address and port. The local network ssdp multicast ipaddress is 239.255.255.250 and the port is 1900.

In order to query for the roku ip address, your program can send the following request using the http protocol to 239.255.255.250 port 1900:

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

Remember that there must be a blank line at the end of the file above. If the above request is put into the file 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 via wireshark and filter on port 1900 you can see the Roku's 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
```

When you get a 200 status response, the Location header will be valid. You can parse out the URL for the Roku ECP service from the Location header. The Roku serial number is contained in the USN line after uuid:roku:ecp. Note that if there are multiple Roku boxes in your local 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 on your network to send commands to. It's probably best to also let the user give his own names to the USNs. Please note the Cache-Control header, we multicast NOTIFY messages periodically (currently once per minute, but this may change). It is safe to assume the unit is no longer available if you haven't received a new NOTIFY message before the Cache-Control max-age time expires.

3.0 External Control Protocol Services

The External Control Service is a simple, RESTful service accessed via the http protocol on port 8060. Once you have the Roku ipaddress to connect to, you can issue the following External Control commands to the Roku:

- **query/apps** This 'query/apps' returns a map of all the channels installed on the Roku box paired with their app id. This command is accessed via an http GET.
- **keydown** takes an argument describing the key pressed. Keydown is equivalent to pressing down the remote key whose value is the argument passed. This command is sent via a POST with no body.
- **keyup** takes an argument describing the key to release. Keyup is equivalent to releasing the remote key whose value is the argument passed. This command is sent via a POST with no body.
- **keypress** takes an argument describing the key that is pressed. Keyup is equivalent to pressing down and releasing the remote key whose value is the argument passed. This command is sent via a POST with no body.
- **launch** takes an app id as an argument and a list of url parameters that are sent to the app id as an roAssociativeArray passed the the RunUserInterface() or Main() entry point. This command is sent via a POST with no body.

Since Firmware version 2.8:

- **query/icon** This 'query/icon' takes an app id as an argument and returns an icon corresponding to that app. The binary data with an identifying MIME-type header is returned. This command is accessed via an http GET.
  Example: GET /query/icon/12

Since Firmware version 3.0:

- **input** enables a developer to send custom events to their Brightscript 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 a BrightScript associative array and passes them directly through to the currently executing channel script via a Message Port attached to a created roInput object. Please refer to Section 3.1 below for more 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 via a POST with no body.
  Example: POST /input?acceleration.x=0.0&acceleration.y=0.0&acceleration.z=9.8

3.1 External Control 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

There are four sensor 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)

Orientation's coordinate system is relative to the point on Earth's surface between the device and the Earth's center:
+x = east

+y = north
+z = towards Earth's center (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^2
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.

Other information you might want to pass via the **input** command may include:
1. sensor accuracy
2. geolocation (from GPS)
3. device-provided derivations of above sensor readings, for example "shake" from accelerometer, or "pinch" from multi-touch.

3.2 External Control Protocol Examples

The following are some example commands sent via the ncat command.

```
 # The following command gets a list of channels and their corresponding app ids.
% echo -e 'GET /query/apps HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
HTTP/1.1 200 OK
Content-Length: 802

<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>
 <app id="352">DreamTV dreamtvs.com</app>
 <app id="419_0">RE/MAX University (v1.0.21)</app>
 <app id="1457">UFC </app>
 <app id="1576">CDNTwo</app>
 <app id="1609_6">Vimeo</app>
 <app id="1611_c">UFC </app>
 <app id="1619">Radio Paradise</app>
 <app id="1688">Roku Newscaster</app>
 <app id="1689_f">Chaneru</app>
 <app id="1705">MP3tunes</app>
 <app id="1756">Break.com</app>
 <app id="1816">YuppTV</app>
 <app id="2003">Dyyno Roku Channel</app>
 <app id="2029">MOG 0.9.6</app>
 <app id="dev">Simple Video Player</app>
</apps>

# The following command simulates a user hitting the “Home” button
% echo -e 'POST /keypress/Home HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
HTTP/1.1 200 OK
Content-Length: 0

# The following commands move the cursor to the far left by holding down
# the Left key for 10 seconds
% echo -e 'POST /keydown/Left HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
HTTP/1.1 200 OK
Content-Length: 0

% sleep 10
% echo -e 'POST /keyup/Left HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
HTTP/1.1 200 OK
Content-Length: 0
```

```
# 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.
% echo -e “POST \
>launch/dev?url=http%3A%2F%2Fvideo.ted.com%2Ftalks%2Fpodcast%2FVilayanurRamac\
>handran_2007_480.mp4&streamformat=mp4 HTTP/1.1\r\n\r\n” | ncat 192.168.1.134 8060
HTTP/1.1 200 OK
Content-Length: 0

# 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.
% echo -e 'POST /launch/dev?contentID=my_content_id&options=my_options HTTP/1.1\r\n' \
> | ncat <Roku IP address> 8060
 HTTP/1.1 200 OK
Content-Length: 0

# 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. It returns the installed apps on a Roku box.
# 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.
% echo -e 'POST /launch/11?contentID=14 HTTP/1.1\r\n\r\n' | ncat 192.168.1.114 8060
 HTTP/1.1 200 OK
Content-Length: 0

# This following command will return the icon for the channel with id 12
# (should be Netflix). The response will be raw binary picture data
# after http headers including one with the MIME type of the picture data.
# Using curl in this example as ncat also returns headers
# Use ImageMagick to view Netflix poster
% curl 'http://192.168.1.134:8060/query/icon/12' > img.png
% display img.png

# 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 currently running
# app just need to agree on the query string parameters and any communication can
# be developed.
% echo –e \
 >  'POST /input?acceleration.x=0.0&acceleration.y=0.0&acceleration.z=9.8 HTTP/1.1\r\nr\n' \
 > | ncat 192.168.1.120 8060

# The following command indicates that a touch at the given x and y
# has touched down on the screen
% echo –e \
> ‘POST /input?touch.0.x=200.0&touch.0.y=135.0&touch.0.op=down HTTP/1.1\r\nr\n' \
 > | ncat 192.168.1.120 8060
```

3.3 Valid Keys

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 single keys are transmitted with the “Lit_” prefix. So that 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 our actual physical remotes. 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 specially named keys that are recognized by the Roku ECP:

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

On the on-screen keyboard, ‘roku’ can be sent via the following commands:

```
% echo -e 'POST /keypress/Lit_r HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
% echo -e 'POST /keypress/Lit_o HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
% echo -e 'POST /keypress/Lit_k HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
% echo -e 'POST /keypress/Lit_u HTTP/1.1\r\n\r\n' | ncat 192.168.1.134 8060
```

3.4 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 internet catalogs browsed on the internet that could instantly be watched in your channel on the Roku. The door is open to many creative uses.

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.

3.5 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 boxes on the local network. The first box that responds is the one that the program then sends commands to. 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 roAssociativeArray passed to the Main() entry point.

The simplevideoplayer application immediately launches the roVideoScreen when it is launched with an roAssociativeArray containing valid url and streamformat keys.

With the External Control Protocol, you have complete control of your Roku box 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 app in examples/source/ecp_client/android_remote and a simple applications to find Roku boxes on the local LAN in examples/source/ecp_client/Roku_Finder.