# Debugging Your Application

Testing Roku Channels involves using a debug console and access to a variety of ports. The debug console provides a window into the runtime environment and provides features such as crash logs, stack-traces and much more.

**Table of Contents**



---

**Nested under this page:**



---

## Accessing the debug console

The Roku Plugin for Eclipse contains a built-in console with access to all the available debug ports.

The debug console can also be accessed using telnet through a shell application such as PuTTY for Windows or *terminal* on Mac and Linux:

```
telnet roku-ip-address 8085
```

The console shows the output of your channel during runtime. If the channel crashes, the debugger will display the line number of the error, as well as the contents of variables at the time of the crash. Compilation errors (e.g. "Syntax Error") are also displayed here. The developer console should be open whenever a channel is side-loaded to catch any possible startup errors.

In addition to displaying console output, the shell can also be used as an interactive debugger. When your application is running, simply press `ctrl-C` to break into the application and enter debug mode. You will see a "BrightScript Debugger>" prompt, where you can type commands.

> You can also force your channel to break at a specific point by inserting `STOP` statements in your code. Be sure to remove this when submitting your channel for publication.

## Debug ports

| Telnet Port | Thread | Description |
|---|---|---|
| 8080 | debug server | debug server containing a host of utilities |
| 8085 | BrightScript console | BrightScript runtime environment |
| 8087 | Screensaver | The starting point for screensavers |

## BrightScript console (port 8085) commands

| Command | Description |
|---|---|
| `bsc` | Print current BrightScript component instances |
| `bscs` | Print a summary of BrightScript component instance counts by component type |
| `brkd` | Toggle whether BrightScript should break into the debugger after non-fatal diagnostic messages |
| `bt` | Print backtrace of call function context frames |
| `classes` | Print BrightScript component classes |
| `cont or c` | Continue script execution |
| `down or d` | Move down the, function context chain one |
| `exit` | Exit shell |
| `gc` | Run garbage collector |
| `help` | Print the list of BrightScript console commands |
| `last or l` [3] | Print the last line that executed |
| `list` | List current function |
| `next or n` [3] | Print the next line to execute |
| `over` [1] | Step over a function |
| `out` [1] | Step out of a function |
| `print, p, or ?` | Print a variable or expression |
| `step, s, or t` | Step one program statement |
| `threads <ID>` [2] `or ths <ID>` [3] | List all current executed suspended threads |
| `thread <ID>` [2] `or th <ID>` [3] | Select a suspended thread to debug - all following debug commands will execute within that thread |
| `up or u` | Move up the function context chain one |
| `var` | Print local variables and their types/values |

> [1] *This command is only available in firmware version 7.2 and greater.*
>
> [2] *This command is only available in firmware version 7.5 and greater.*
>
> [3] *Shortcuts for `next`, `last`, `threads`, and `thread` are only available in firmware version 7.6 and greater.*
>
> *BrightScript statements can also be compiled and executed in the console. This can be used to change variables during execution or call a function that prints useful information about the state of your program.*

## SceneGraph applications

Beginning with **firmware version 7.5** and above, the main BrightScript console (port 8085) now provides context for all threads. This eliminates the need to have multiple telnet sessions open for each running thread and **ports 8089 - 8093 will no longer be used.**

As seen below, any break or `stop` in the channel will suspend all threads. All threads will be listed with the following information:

- **ID**: thread ID
- **Location**: file the thread originated from and line number
- **Source Code**: current line of code

The current selected thread will be marked with an *.

```
BrightScript Micro Debugger.
Enter any BrightScript statement, debug commands, or HELP.
Suspending threads...
Thread selected:  0*   pkg:/source/Main.brs(19)               msg = wait(0, m.port)
Current Function:
011:      m.port = CreateObject("roMessagePort")
012:      screen.setMessagePort(m.port)
013:
014:      'Create a scene and load /components/helloworld.xml'
015:      scene = screen.CreateScene("HelloWorld")
016:      screen.show()
017:
018:      while(true)
019:*         msg = wait(0, m.port)
020:          msgType = type(msg)
021:          if msgType = "roSGScreenEvent"
022:              if msg.isScreenClosed() then return
023:          end if
Break in 19
019:         msg = wait(0, m.port)
Backtrace:
#0  Function main() As Void
   file/line: pkg:/source/Main.brs(19)
Local Variables:
global           Interface:ifGlobal
m                roAssociativeArray refcnt=2 count:1
screen           bsc:roSGScreen refcnt=1
scene            bsc:roSGNode refcnt=1
msg              <uninitialized>
msgtype          <uninitialized>
Threads:
ID    Location                                Source Code
 0*   pkg:/source/Main.brs(19)                msg = wait(0, port)
 1    ...                    Task.brs(25)     msg = wait(0, m.port)
 2    ...                    Task.brs(29)     msg = wait(0, m.port)
  *selected
```

> *This information can be recalled anytime during debugging using the `threads` command.*

## Debug server (port 8080) commands

| Command | Description |
|---|---|
| logrendezvous [on\|off] [2] | Enable console logging of thread rendezvous. Set to `off` to disable. |
| loaded_textures | Displays the current set of images loaded into texture memory |
| r2d2_bitmaps | Prints a list of assets loaded into texture memory and the amount of free, used, and maximum available memory on your device, respectively. |
| sgnodes all [1] | Prints every existing node created by the currently running channel |
| sgnodes roots [1] | Prints every existing node without a parent created by the currently running channel. The existence of these un-parented nodes means they are being kept alive by direct BrightScript references. These could be in variables local to a function, arrays, or associative arrays, including a component global `m` or an associative array field of a node. |
| sgnodes node_ID [1] | Prints nodes with an id field set to node_ID, except it, bypasses all the hierarchy and rules and just runs straight down the whole list in the order of node creation. It will list multiple nodes if there are several that match. |
| `sgperf start\|clear\|report\|stop` [2] | This command provides basic node operation performance metrics. This command tracks all node operations by a thread, whether it's being created or an operation on an existing node, and whether it involves a rendezvous.<br>**Settings:**<br>• `start` - enables counting<br>• `clear` - resets counters to zero<br>• `report` - prints current counts with rendezvous as a percentage<br>• `stop` - disables counting |
| `sgversion force` or `default` `1.0` or `1.1` | This command can be used to change the observer callback model and can also override the default `rsg_version` specified in the manifest. For example, `sgversion force 1.0` will set `rsg_version=1.0` regardless of what is specified in the manifest. With `default`, it will set the default `rsg_version` when it is not specified in the manifest. Changing the `rsg_version` will require restarting the channel, but these changes will not survive a device reboot.<br>Please note that support for the "rsg_version=1.0" manifest flag is deprecated as of Roku OS 8. This deprecation means that the 1.0 features continue to work in Roku OS 8, but will no longer be supported (and thus should not be expected to work) starting with the next major firmware release. All channels will have to adopt the current observer callback model in successive firmware updates. |
| `fps_ display` [3] | The command displays frames-per-second and free memory on-screen. Leverage this tool to optimize your channel UI.<br>Following are the commands to use the fps meter:<br>• `fps_display 1` turns on the fps meter. It presents a 1-second moving average of the current frame rate<br>• `fps_display 0` turns the meter back off |

[1] *These commands are similar to the getAll() , getRoots() , getRootsMeta(), and getAllMeta() ifSGNodeChildren methods, which can be called on any SceneGraph node.*

[2] *Available since firmware version 7.6.*

[3] *Available since firmware version 8.*