Contents

Introduction ................................................................................................................. 6
Unpacking the Stack ...................................................................................................... 6
DUT and master chambers ............................................................................................... 8
Programmable attenuators .............................................................................................. 9
Throughput test configuration ....................................................................................... 9
Alternative test configuration ....................................................................................... 10
DUT placement ............................................................................................................. 11
Controlling the internal PC ........................................................................................... 11

Software installation and getting started ...................................................................... 12
Configuring IxChariot endpoints .................................................................................. 12
Configuring iPerf endpoints ........................................................................................ 13
Configuring the OB-THROUGHPUT script .................................................................. 13
  User-configurable parameters .................................................................................. 13
  octoScope TCL namespace API .............................................................................. 16
  Customizing test flow ............................................................................................... 17
Launching scripts .......................................................................................................... 18
  Wrapper script to terminate script execution .......................................................... 18
Plotting test data .......................................................................................................... 19

Programming the octoBox turn table ........................................................................... 20
Interfacing with the turn table ...................................................................................... 20
Turn table API ............................................................................................................. 20
Turn table example code .............................................................................................. 21
  Initialize turn table .................................................................................................. 21
  Move turn table to -180 degrees ........................................................................... 21
  Reset position to zero and disable the turn table .................................................. 21
Turn table troubleshooting ........................................................................................... 21
  Turn table error reported in console: ................................................................. 21
  Turn table not moving/fails to move to position: .................................................. 22

Programming the octoBox quadAtten .......................................................................... 23
Interfacing with the octoScope quadAtten .................................................................. 23
quadAtten command line API ..................................................................................... 23
quadAtten example code ............................................................................................. 24
quadAtten troubleshooting ......................................................................................... 26
  QuadAttenuatorControl.exe command line tool does not launch ....................... 26
  Device is not responding – USB ........................................................................... 26
  Device is not responding – Ethernet ................................................................ 26
quadAtten advanced ASCII API ............................................................................... 27
quadAtten ASCII API ................................................................................................. 27
quadAtten example code................................................................. 28
Programming the iGen interference traffic and waveform generator ............................................. 29
Overview .......................................................................................... 29
Network Configuration Commands ......................................................................................... 30
system version update [filename_s] .................................................................................... 30
system version get web ....................................................................................... 30
system version get firmware ................................................................................. 30
system version get eos ................................................................................. 30
system version get command ........................................................................... 30
system set mode [mode_s] ........................................................................... 30
netconfig wifi dhcp set enable [en_bool] ......................................................... 30
netconfig wifi dhcp get address .................................................................. 30
netconfig wifi dhcp get enable .................................................................. 30
netconfig wifi dhcp get subnet ................................................................... 30
netconfig wifi static set address [ipaddr_s] .................................................. 30
netconfig wifi static set subnet [ipaddr_s] .................................................... 30
netconfig wifi get mac .......................................................................... 30
netconfig wifi static get address ................................................................ 30
netconfig wifi static get subnet ................................................................ 30
netconfig wifi static set address [ipaddr_s] .................................................. 30
netconfig wifi static set subnet [ipaddr_s] .................................................... 30
netconfig get hostname ...................................................................... 30
netconfig get hostname ...................................................................... 30
netconfig dhcp get address ..................................................................... 30
netconfig dhcp get enable ..................................................................... 30
netconfig dhcp get subnet ..................................................................... 30
netconfig dhcp set enable [en_bool] ........................................................ 30
netconfig dhcp get mac ........................................................................ 30
netconfig static get address .................................................................. 30
netconfig static get subnet .................................................................... 30
netconfig static set address [ipaddr_s] ..................................................... 30
netconfig static set subnet [ipaddr_s] ....................................................... 30
Waveform Commands ........................................................................... 36
waveform pulse get count ..................................................................... 36
waveform pulse get enable ..................................................................... 36
waveform pulse get off ........................................................................ 36
waveform pulse get on .......................................................................... 36
3
www.octoscope.com
octoBox User’s Guide
```plaintext
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>waveform pulse get period</td>
<td>37</td>
</tr>
<tr>
<td>waveform pulse set count [count_u]</td>
<td>37</td>
</tr>
<tr>
<td>waveform pulse set enable [en_bool]</td>
<td>37</td>
</tr>
<tr>
<td>waveform pulse set off [off_u]</td>
<td>38</td>
</tr>
<tr>
<td>waveform pulse set on [on_u]</td>
<td>38</td>
</tr>
<tr>
<td>waveform pulse set period [period_u]</td>
<td>38</td>
</tr>
<tr>
<td>waveform pulse update</td>
<td>39</td>
</tr>
<tr>
<td>waveform synth get enable</td>
<td>39</td>
</tr>
<tr>
<td>waveform synth get freq</td>
<td>39</td>
</tr>
<tr>
<td>waveform synth set enable [en_bool]</td>
<td>39</td>
</tr>
<tr>
<td>waveform synth set freq [Khz_ul]</td>
<td>39</td>
</tr>
<tr>
<td>Wi-Fi Commands</td>
<td>41</td>
</tr>
<tr>
<td>wifi channel get bandwidth</td>
<td>41</td>
</tr>
<tr>
<td>wifi channel get primary</td>
<td>41</td>
</tr>
<tr>
<td>wifi channel get secondary</td>
<td>41</td>
</tr>
<tr>
<td>wifi channel set bandwidth [MHz_sl]</td>
<td>41</td>
</tr>
<tr>
<td>wifi channel set primary [chan_ui]</td>
<td>41</td>
</tr>
<tr>
<td>wifi config get guard</td>
<td>42</td>
</tr>
<tr>
<td>wifi config get interface</td>
<td>42</td>
</tr>
<tr>
<td>wifi config get mode</td>
<td>42</td>
</tr>
<tr>
<td>wifi config get rate</td>
<td>42</td>
</tr>
<tr>
<td>wifi config get wmm</td>
<td>42</td>
</tr>
<tr>
<td>wifi config set guard [interval_u]</td>
<td>42</td>
</tr>
<tr>
<td>wifi config set interface [type_s]</td>
<td>43</td>
</tr>
<tr>
<td>wifi config set mode [mode_s]</td>
<td>43</td>
</tr>
<tr>
<td>wifi config set rate [rate_sl]</td>
<td>43</td>
</tr>
<tr>
<td>wifi config set wmm [access_s]</td>
<td>44</td>
</tr>
<tr>
<td>wifi playback get filename</td>
<td>44</td>
</tr>
<tr>
<td>wifi playback get ips_duration</td>
<td>44</td>
</tr>
<tr>
<td>wifi playback get loopback</td>
<td>44</td>
</tr>
<tr>
<td>wifi playback get run en_bool</td>
<td>44</td>
</tr>
<tr>
<td>wifi playback set filename [name_s]</td>
<td>45</td>
</tr>
<tr>
<td>wifi playback set ips_duration [usec_sl]</td>
<td>45</td>
</tr>
<tr>
<td>wifi playback set loopback [en_bool]</td>
<td>45</td>
</tr>
<tr>
<td>wifi playback set run [en_bool]</td>
<td>45</td>
</tr>
<tr>
<td>Programming Example</td>
<td>47</td>
</tr>
<tr>
<td>continuous wave mode</td>
<td>47</td>
</tr>
<tr>
<td>OOK pulse mode</td>
<td>47</td>
</tr>
<tr>
<td>wifi playback start</td>
<td>48</td>
</tr>
</tbody>
</table>
```
Introduction

octoBox® STACK testbed is used for MIMO throughput measurements of Wi-Fi, LTE, Bluetooth and other devices. The anechoic environment inside octoBox chambers enables OTA (over the air) testing in the frequency range of 700 MHz to 6 GHz. Product literature and test applications information is available on our website.

Unpacking the Stack

The door of the crate has slanted features and serves as a ramp for wheeling the Stack out of the crate.

Remove cleats in the front, lower the door and lay it down next to the crate as a ramp.

MPE-26 Stack crate
35”W x 47”D x 55”H (89 x 119 x 139 cm)
520 LBS (236 kg)

MPE-38 Stack crate
38”W x 56”D x 68”H (100 x 140 x 170 cm)
600 LBS (272 kg)
Open the crate door and lay it on the ground as a ramp.

Remove blocks of foam, unscrew and remove the wheel chock and roll octoBox MPE testbed out of the crate.
DUT and master chambers

Typically the DUT (device under test) is placed into the top octoBox chamber and the master device is placed into the bottom chamber. This placement makes it physically easier to access the DUT, but is not a requirement. The master chamber need not be accessed very often once configured and can be at the bottom of the stack where access may be less convenient.

If the DUT is a client device (e.g. phone, tablet or PC), the master is typically an access point (AP) or a base station. If the DUT is an AP or a base station, the master is typically a client device.

The MPE (multi path emulator) module is stacked between the master and DUT chambers. The master and the DUT chambers are coupled through the MPE and programmable attenuators located in the octoBox quadAtten module. The attenuators are programmed to measure throughput vs. path loss.
Programmable attenuators

quadAtten module containing programmable attenuators is mounted on the side of the octoBox.

quadAtten can be powered and controlled from a console PC using either its Ethernet/PoE or USB port. Attenuation range is 60 dB in 0.5 dB steps. quadAtten is the first attenuator module that is well enough isolated to be mounted outside the octoBox while still maintaining testbed isolation.

Throughput test configuration

A typical throughput test configuration has the DUT and master connected through programmable attenuators in series with the MPE, as shown in this block diagram.

Traffic generator software, such as iperf or IxChariot can be used to send traffic between the DUT and the master.

Programmable attenuators add path loss and the MPE module adds multipath between the master and the DUT, simulating typical home or office conditions.
Alternative test configuration

In the test configuration shown below, the console PC is located outside of the octoBox with the link between a bridge (AP in bridge mode) in the bottom chamber and an AP under test in the top chamber.
**DUT placement**

Typically the RF signals to the DUT are coupled over the air (OTA). This example shows a 4x4 MIMO link with 4 MIMO antennas on the right. The DUT is placed on a turn table when a turn table is available. An RF-transparent support (e.g. a block of Styrofoam) can be used to lift up the DUT if necessary. The cables connected to the DUT can be fastened on the top rail (not shown).

**Controlling the internal PC**

If a PC is used as a client in the test link, connection to the internal PC from an external console PC is typically done using the Remote Desktop utility available under Windows. Remote Desktop works via the octoBox Ethernet filter.

Remote Desktop provides remote control of the internal PC.
Software installation and getting started

Please follow the steps in the “octoBox OB-THROUGHPUT Test Quick Start Guide”, document octoBox_throughput_quick_start.pdf in the folder C:\Program Files\octoScope\documentation for installation instructions and how to get started running the OB-THROUGHPUT script using iperf.

To use IxChariot traffic generator with the script, including the SmallNetBuilder.com script, you will need to have IxChariot installed prior to the installation of the octoScope software.

To configure the IxChariot endpoints refer to the IxChariot documentation or follow the next section of this document

Configuring IxChariot endpoints

By default, the IP addresses are configured in both the script files and the IxChariot .tst files so that the scripts will function out of the box, provided you configure the IP addresses of the console and endpoint PCs to the addresses of each endpoint. Refer to the quick start guide (see previous section) for more details on the default IP address settings. The IP address of the IxChariot endpoint has to be configured to the same IP address as the device on which it is installed.

All the IxChariot .tst files are located in the C:\Program Files\octoScope\script folder.

The octoScope throughput.tcl script uses the tpt_config.tst file. The SmallNetBuilder.com script, OB_SNB_script.tcl, uses the three SNB_throughput_*_90sec.tst files.

You should not save any test results into these .tst files being used by the scripts. If you happen to run the test and save the results into one of these .tst files, the script may issue an error message: “set failed: Results were not cleared.” This message comes directly from IxChariot. To resolve this issue, open the affected .tst file(s), clear the test results and then re-save the file(s).
Configuring iPerf endpoints

Typically, TCP traffic flows from the iPerf client running on the console PC to the iPerf server running on the endpoint device. The iPerf test parameters are configured in the throughput script that runs on the console as discussed in the next section. However, in order to run an iPerf throughput test, the server must first be manually started on the endpoint device. If the endpoint device is a Windows PC, open a command prompt and type the following to start the iPerf server:

```
>iperf -s
```

Configuring the OB-THROUGHPUT script

The test automation software contains a core library, `octoScope_OBTPT.tcl` in the C:\Program Files\octoScope\script folder. This script contains the octoScope TCL namespace and testbed control functions. Other scripts, `throughput.tcl` and `OB_SNB_script.tcl`, use this library with different configurations and test flows for various test conditions.

User-configurable parameters

To explore the user parameters, open the `throughput.tcl` file. The user defined parameters are separated into several sections in the header of this file. A description of what each does can be found in the comments as well as in the table below.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Section</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stopOnError</td>
<td>False</td>
<td>True – Script will quit on any error. False – Script will ignore and continue to run if an error is thrown.</td>
</tr>
<tr>
<td>Timeout</td>
<td>60</td>
<td>The total duration that iPerf or IxChariot will run in seconds</td>
</tr>
</tbody>
</table>
The output folder for the .tst and .csv data.

<table>
<thead>
<tr>
<th>Turntable Section</th>
<th>posEnable</th>
<th>True</th>
<th>True – enables the turn table False – disables the turn table</th>
</tr>
</thead>
<tbody>
<tr>
<td>homeTTEnable</td>
<td>True</td>
<td>True</td>
<td>True – At the beginning of the test sequence, home the turn table False – Set the current turn table position as home.</td>
</tr>
<tr>
<td>rotateDuringTestEnable</td>
<td>False</td>
<td>True</td>
<td>True – While IxChariot test is running, rotate the turn table from the ‘posStart’ to ‘posStop’ (degrees) at rotation speed ‘rotationSpeed’ (RPM) False – Do not rotate the turn table while a throughput test is running.</td>
</tr>
<tr>
<td>posStart</td>
<td>-180</td>
<td>The start position of the turn table in degrees. Note: Can be &gt; 360 degrees for multiple rotations (e.g. 720 deg)</td>
<td></td>
</tr>
<tr>
<td>posStop</td>
<td>180</td>
<td>The end position of the turn table in degrees. Note: Can be &gt;360 degrees for multiple rotations (e.g. 720 deg)</td>
<td></td>
</tr>
<tr>
<td>posStep</td>
<td>30</td>
<td>The turntable rotation increment in degrees. Used in the test flow section.</td>
<td></td>
</tr>
<tr>
<td>rotationSpeed</td>
<td>1</td>
<td>The turntable rotation speed in revolutions per minute (RPM)</td>
<td></td>
</tr>
</tbody>
</table>

| Attenuator Section | attenEnable | True | True – Enable attenuator control False – Disable attenuator control |
|-------------------|------------|------|------------------------------------------------|---|
| vaunixEnable | False | True | Enable legacy Vaunix support False – Enable quadAttenuator support |
| attenCount | 3 | Define number of Vaunix attenuators plugged in via usb (Legacy support) |
| quadAttenAddress | “default” | Sets the address of the quadAttten if controlling device via Ethernet. Example: “192.168.15.5” |
| attenStart | 0 | The start attenuation value (dB) |
| attenStop | 60 | The final attenuation value (dB) |
| attenStep | 5 | The attenuation step size (dB) |
### iperf Section

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iperfEnabled</td>
<td>True</td>
<td>True - Enable iperf test data collection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False – Do not run iperf tests</td>
</tr>
<tr>
<td>iperfServerAddress</td>
<td>192.168.15.6</td>
<td>IP address of iperf server running on remote endpoint</td>
</tr>
<tr>
<td>iperfOutputFile</td>
<td>“tpt_data.csv”</td>
<td>The test results output CSV file name</td>
</tr>
<tr>
<td>iperfSwitch</td>
<td>“-w 256K –l 1300”</td>
<td>iperf run options for the client running on the console. See iperf documentation for a description of all possible options. Note that the options (-c -t -y -f) are set automatically by the throughput test, so do NOT include in the iperfSwitch.</td>
</tr>
</tbody>
</table>

### IxChariot Section

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chariotEnable</td>
<td>True</td>
<td>True - Enable IxChariot test data collection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False – Do not run IxChariot tests</td>
</tr>
<tr>
<td>testMaskEnable</td>
<td>False</td>
<td>True – Script will average the throughput data after ‘testMaskDelay’ seconds has elapsed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False – Script will average all throughput data</td>
</tr>
<tr>
<td>testMaskDelay</td>
<td>0</td>
<td>The number of seconds of throughput data to ignore</td>
</tr>
<tr>
<td>reportRSSI</td>
<td>True</td>
<td>True – Script will report RSSI values from ixChariot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False - Script will not report RSSI values from ixChariot</td>
</tr>
<tr>
<td>testInputFile</td>
<td>“tpt_config.tst”</td>
<td>The input IxChariot test file. Note: This file must be configured for each endpoint before running script.</td>
</tr>
<tr>
<td>testOutputFile</td>
<td>“tpt_data.tst”</td>
<td>The test results output file name. The .csv and .tst data will follow this scheme.</td>
</tr>
</tbody>
</table>

When iperf is enabled, the scripts immediately starts rotating the turn-table in the background and then starts running the iperf test in a parallel command-line session. Results are extracted as soon as the iPerf test completes, regardless of the turn-table state.

When ixChariot is enabled, the script initiates the ixChariot test, waits until the test starts successfully and then starts rotating the turn-table in the background in parallel after waiting the testMaskDelay time. The script then waits for the ixChariot test to complete to extract results, regardless of the turn-table state.
So in both cases, if there is a mismatch between the test time and corresponding RPM setting, results will just be extracted when either test type completes, regardless of whether the turn table has completed turning or not.

**octoScope TCL namespace API**

The ‘octoScope_OBTPT.tcl’ file contains the octoScope::OBTPT::TCL namespace. This namespace contains a variety of functions useful for building a custom test script. The throughput.tcl script is an example of different ways to step through and configure the system for testing. The table below illustrates the octoScope OBTPT library API.

<table>
<thead>
<tr>
<th>Command Name (In namespace octoScope::OBTPT)</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Run Test Section</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addTest $testInputFile $testOutputFile $testOutputDir</td>
<td>Input .tst file, output .tst file name, output directory name</td>
<td>This function adds a test's parameters into a queue to run. It is used to 'load' tests into a sequence that can be run by calling the “runTest” function.</td>
</tr>
<tr>
<td>initializeTestBed</td>
<td></td>
<td>This calls the initialization sequence for the test bed. If homeTTEnable=true then the turn table will auto home itself, otherwise it will treat the current position as zero degrees.</td>
</tr>
<tr>
<td>setPosition $currentPosition</td>
<td>Position in degrees</td>
<td>Moves the turn table to set position.</td>
</tr>
<tr>
<td>setAttenuator $currentAtten</td>
<td>Attenuation in dB</td>
<td>Sets the attenuation on attenuator</td>
</tr>
<tr>
<td>runAllTests</td>
<td></td>
<td>Runs all of the tests that have been added via the ‘addTest’ function. It will sequence through each test. If rotateDuringTestEnable=true then the turn table will rotate from posStart to posStop at rotationSpeed while the ixChariot test is running. If testMaskEnable=true then the test will take the average of the throughput data ignoring the first testMaskDelay seconds of data. It will delay the rotation of the turn table if the rotateDuringTest option is enabled.</td>
</tr>
<tr>
<td>cleanupTests</td>
<td></td>
<td>Clean up used resources. Call this before exiting script.</td>
</tr>
</tbody>
</table>
Customizing test flow

Using these function calls, it is possible to customize the test flow. The following script is an example of the test flow from the TPT vs Orientation vs Range test.

```ruby
#initialize the test bed
octoScope::CBTPT::initializeTestBed

set currentAttenu $attenStart
set currentPosition $posStart

#Outer loop iterates over the Turn table position
while ($currentPosition <= $posStop) {
    #Call setPosition value to set the turn table position
    octoScope::CBTPT::setPosition $currentPosition

    #Inner loop iterates over different attenuation values
    while ($currentAttenu <= $attenStop) {
        octoScope::CBTPT::setAttenuator $currentAttenu
        octoScope::CBTPT::runAllTests
        incr currentAttenu $attenStep
    }

    #reset the attenuation value to the starting value
    set currentAttenu $attenStart

    #increment the turn table step
    incr currentPosition $posStep
}

puts "Test complete."
```
Launching scripts

To launch the `throughput.tcl` script, double-click on `throughput.bat` file in the `C:\Program Files\octoScope\script` folder. To launch the SmallNetBuilder.com script, double-click on the `SNB_script.bat` in the same folder.

The .bat files launch the target scripts using a wrapper script. See next section for the wrapper script description.

Wrapper script to terminate script execution

The wrapper script allows you to cleanly kill a script mid test by pressing the ‘Enter’ key. This is necessary because of limitations of TCL to recognize force quit events. Wrapper script will, for example, cleanly terminate the turn table rotation and terminate any processes running during the script (e.g. `iperf` or `IxChariot`).

To use the wrapper script from the command line, in the command prompt type:

```
tclsh wrapper.tcl <name of script>
```

For example:

```
tclsh wrapper.tcl throughput.tcl
```
Plotting test data
Follow the instructions in the quick start guide (see first section of this document).
Programming the octoBox turn table

The octoBox turn table can be used to make measurements while the DUT is rotated with respect to the test antennas. The turn table is controlled via USB using the fnPerformaxCmd.exe command line program.

Interfacing with the turn table

- When plugged into a USB port, the turn table will appear as a ‘Performax USB’ device once the proper Arcus drivers have been installed.
- Once installed the ‘fnPerformaxCmd.exe’ can be launched via command line or TCL script with different options to control the Turn Table.

Turn table API

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td>-help</td>
<td>Access help menu.</td>
</tr>
<tr>
<td>-q</td>
<td>-q</td>
<td>Home the turn table.</td>
</tr>
<tr>
<td>-d nn</td>
<td>-d 0</td>
<td>Selects the device to work with (default 0)</td>
</tr>
<tr>
<td>-i</td>
<td>-i</td>
<td>Initialize default motor settings.</td>
</tr>
<tr>
<td>-p nn</td>
<td>-p -135</td>
<td>Move turn table where nn is the absolute position in +/- degrees.</td>
</tr>
<tr>
<td>-z</td>
<td>-z</td>
<td>Zero current position.</td>
</tr>
<tr>
<td>-e nn</td>
<td>-e 1</td>
<td>Turn motor on or off. 1 = on, 0 = off.</td>
</tr>
<tr>
<td>-r n.n</td>
<td>-r 4.5</td>
<td>Set the angular velocity (RPM). If no velocity is specified will use the default speed (5.5RPM).</td>
</tr>
</tbody>
</table>

Return Codes

<table>
<thead>
<tr>
<th>Return Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful (no errors)</td>
</tr>
<tr>
<td>1</td>
<td>Error (invalid command)</td>
</tr>
<tr>
<td>2</td>
<td>Error (no devices found)</td>
</tr>
<tr>
<td>3</td>
<td>Error (device setup error)</td>
</tr>
<tr>
<td>4</td>
<td>Error (send/receive command error)</td>
</tr>
</tbody>
</table>
**Turn table example code**

This example TCL script initializes the turn table and then moves it to -180 degrees at 4RPM. It then moves the turn table to its zero position and disables the motor.

**Initialize turn table**

# -d 0 -> selects the first device (default 0)  
# -i -> initialize default settings  
# -e 1 -> enable turn table motor  
# -z -> sets current position as zero  
set result [exec fnPerformaxCmd.exe -d 0 -i 1 -e 1 -z]

**Move turn table to -180 degrees**

# -d 0 -> selects the first device (default 0)  
# -p -180 -> move turn table to -180 degrees  
# -r 4 -> move at 4RPM  
set result [exec fnPerformaxCmd.exe -d 0 -p -180 -r 4]

**Reset position to zero and disable the turn table**

# -d 0 -> selects the first device (default 0)  
# -p 0 -> move turn table to 0 degrees  
# -e 0 -> disable motor  
set result [exec fnPerformaxCmd.exe -d 0 -p 0]  
set result [exec fnPerformaxCmd.exe -d 0 -e 0]

**Turn table troubleshooting**

**Turn table error reported in console:**

Error in getNumDevices:

```
Motor Test Program
error in fnPerformaxConGetNumDevices
```

This error occurs when no USB turn table resource is available for the command line to use. It is most commonly caused by connection issues (cables not plugged in properly), but it could also be caused by improperly killed scripts (leaving a zombie fnPerformaxCmd.exe thread consuming the USB resource). To fix the issue please disconnect/reconnect the turn table USB cable. If problem persists, try running the [wrapper.tcl script](https://www.octoscope.com) to properly kill the script.
Turn table not moving/fails to move to position:
This is most commonly caused by an uninitialized turn table. To fix this, the table needs to manually be initialized via the command line tool. With the turn table connected, open a command prompt and type:

```
>fnPerformaxCmd.exe -d 0 -e 1 -i
```

```
>fnPerformaxCmd.exe -d 0 -q
```
Programming the octoBox quadAtten

The octoBox® quadAtten module contains 4 individually programmable RF attenuators. quadAtten offers the best RF isolation on the market, making it suitable for highly controlled octoBox wireless testbed environment. While off-the-shelf attenuators have to be internally mounted due to inadequate isolation, octoBox quadAtten module mounts externally for easy access. Each of the 4 attenuators can be individually controlled via USB or Ethernet/PoE.

Each attenuator can be individually assigned an attenuation value from 0 to 60 dB via USB control, IP sockets, or a web based interface. It can either be powered via USB or through power over Ethernet (POE).

Interfacing with the octoScope quadAtten

USB:
- When plugged into a USB port, the quadAtten will appear as a serial COM port when the proper USB driver has been pre-installed.
- Once installed, the ‘QuadAttenuatorControl.exe’ command line tool can be used to send commands and configure the device.

Ethernet:
- Once the IP settings of the device have been set using the USB control, the quadAtten can be addressed via IP.
- Once configured, navigating to the device’s IP address in a browser will bring up a web form to view and set the attenuation values for each channel. Alternatively, you can use the ‘-d’ option on the ‘QuadAttenuatorControl.exe’ to select the IP address of the device you would like to work with.

quadAtten command line API

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d</td>
<td>-d COM11</td>
<td>Selects the device to use either the com port, name, IP address, or serial number. If -d is not specified will look for a QuadAtten that is connected via USB.</td>
</tr>
<tr>
<td></td>
<td>-d QuadName</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-d 192.168.15.56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-d QA40516-02</td>
<td></td>
</tr>
<tr>
<td>-c</td>
<td>-a 14.5 –c 1</td>
<td>Specifies the channel to set.</td>
</tr>
<tr>
<td>-a</td>
<td>-a 14.5</td>
<td>Sets the attenuation value. If no channel is specified will set all channels equal to this value.</td>
</tr>
<tr>
<td></td>
<td>-a 14.5 –c 1</td>
<td></td>
</tr>
<tr>
<td>-i</td>
<td>-i 192.168.15.56</td>
<td>Sets the IP address of the device.</td>
</tr>
</tbody>
</table>
-g 192.168.15.1  Sets the Gateway address of the device.
-s 255.255.255.0  Sets the subnet mask of the device.
-n SignalAtten1  Sets a custom name used to help identify and address multiple quadAttens. (Max 20 characters no spaces)
-z  Returns information settings of the device. (Name, serial, firmware, IP address, subnet, gateway, MAC address, Ch1/2/3/4 settings)

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>No device found</td>
</tr>
<tr>
<td>2</td>
<td>Connection timeout</td>
</tr>
<tr>
<td>3</td>
<td>Invalid Parameters</td>
</tr>
</tbody>
</table>

quadAtten example code

Command line

Set IP settings (Note, each command should be done individually and with time allowed for device to configure itself).

Set IP Address

>QuadAttenuatorControl.exe –i 192.168.15.56

Gateway:

>QuadAttenuatorControl.exe –g 192.168.15.1

Subnet Mask:

>QuadAttenuatorControl.exe –s 255.255.255.0

Set device name

>QuadAttenuatorControl.exe –n SignalAtten1
**Set attenuation**
Set all channels to 14.5dB attenuation

>`QuadAttenuatorControl.exe -a 14.5`

Set channel 2 to 14.5dB attenuation:

>`QuadAttenuatorControl.exe -a 14.5 -c 2`

Set all channels on IP address 192.168.15.56 to 14.5dB attenuation

>`QuadAttenuatorControl.exe -d 192.168.15.56 -a 14.5`

**Device selection**
Automatically select the Attenuator connected via USB and display information:

>`QuadAttenuatorControl.exe -z`

Select the Ethernet device at IP address 192.168.15.56

>`QuadAttenuatorControl.exe -d 192.168.15.56 -z`

Select the USB device on COM port 11:

>`QuadAttenuatorControl.exe -d COM11 -z`

Select the USB device with serial “QA40516-02”:

>`QuadAttenuatorControl.exe -d QA40516 -z`

Select the USB device with name “SignalAtten1”:

>`QuadAttenuatorControl.exe -d SignalAtten1 -z`

**Display current settings:**
>`QuadAttenuatorControl.exe -z`

**TCL**

**Set attenuation**
Example code to set attenuation values:

```tcl
#Designate the QuadAtten device
set device 192.168.15.56
set device COM11
```
set device default  #will automatically find a device to send via USB
set device SignalAtten1
set device QA40516-02

#Set the attenuation values
set atten 45.5
#Define the channel (optional)
set channel 1

set result [exec QuadAttenuatorControl.exe -d $device -a $atten -c $channel]

**quadAtten troubleshooting**

**QuadAttenuatorControl.exe command line tool does not launch**
- The command line tool requires network access and access to connected devices, so make sure the command line program has been launched with administrative privileges.
  - To do this, goto: “Start Menu – type ‘cmd’ in the search field – right click and open as administrator – navigate to command line tool directory and launch tool.

**Device is not responding – USB**
- If QuadAtten appears as an unrecognized device, please install drivers:
  - Navigate to device manager: “Start – type ‘Device Manager’ in search field”
  - Check both “Other” and “Ports (COM&LP)” section for a device called “OctoScope QuadAtten”.
  - Right click on device and click update driver.
  - Select “Browse computer for driver”/”Manually select driver”
  - Navigate to folder containing ‘quadAtten.inf’ in the installer. (Default path octoscope\quadAtten\drivers)
  - Press ok to install drivers.
  - Reconnect QuadAtten
- If QuadAtten appears as a COM port but is not responding to ‘QuadAttenuatorControl.exe’ please ensure that no other programs are accessing the COM port resource (ie: PuTTY, other scripts running) and retry.

**Device is not responding – Ethernet**
- Use the ‘–z’ command while device is plugged in via USB to view and verify IP settings.
Check for a ping response (open command line prompt, type “ping xxx.xxx.xxx.xxx” where xxx is the IP address of the device.

- If no response or there is a timeout, then check the following list:
  - Device is powered
  - Network path is resolved
  - Gateway/Subnet Mask/IP are set properly
- If no response, but settings are ok. Then cycle power (remove both POE and USB power) and try again.

If device settings are correct and responds to ping requests, then ensure that port 80 is open on the network (for web form interface) and port 5025 is open (for socket commands via the command line tool).

quadAtten advanced ASCII API
If the quadAtten command line tool is not used, direct ASCII can be written either via an IP socket or a COM to the device. It is important to note that the quadAtten command line tool should be the primary way to interface with the device.

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSerial</td>
<td>getSerial</td>
<td>Returns the serial number of the attenuator.</td>
</tr>
<tr>
<td>getAttenuation –ch N</td>
<td>getAttenuation –ch 1</td>
<td>Returns the current attenuation value of a specified channel.</td>
</tr>
<tr>
<td>getInfo</td>
<td>getInfo</td>
<td>Reports serial, firmware, IP, Gateway, Subnet, MAC and channel settings.</td>
</tr>
<tr>
<td>getIP</td>
<td>getIP</td>
<td>Returns the IP address.</td>
</tr>
<tr>
<td>getGateway</td>
<td>getGateway</td>
<td>Returns the local gateway address.</td>
</tr>
<tr>
<td>getSubnetMask</td>
<td>getSubnetMask</td>
<td>Returns the subnet mask.</td>
</tr>
<tr>
<td>getName</td>
<td>getName</td>
<td>Returns the name identifier.</td>
</tr>
<tr>
<td>setAttenuation –ch N –db N</td>
<td>setAttenuation –ch 1 –db 45.5</td>
<td>Sets the attenuation level of a specified channel.</td>
</tr>
<tr>
<td>setAttenuation –db N</td>
<td>setAttenuation –db 20</td>
<td>Sets the attenuation levels of all channels.</td>
</tr>
<tr>
<td>setIP</td>
<td>setIP 192.168.0.15</td>
<td>Sets the IP address.</td>
</tr>
<tr>
<td>setGateway</td>
<td>setGateway 192.168.0.1</td>
<td>Set’s the gateway address.</td>
</tr>
<tr>
<td>setSubnetMask</td>
<td>setSubnetMask 255.255.255.0</td>
<td>Sets the subnet mask.</td>
</tr>
<tr>
<td>setName</td>
<td>setName Room1</td>
<td>Sets the name parameter of the device.</td>
</tr>
</tbody>
</table>
**quadAtten example code**
This is a simple TCL example of how to send an ASCII command via Ethernet through a socket. This example sets the attenuation on Ch2 to be 20dB.

**Interfacing via socket**

```tcl
#Create a socket connection to 192.168.15.15, port 5025
set socket [socket 192.168.15.150 5025]
set db 20
set ch 2
puts $socket "setAttenuation -ch $ch -db $db"
flush $socket
puts "Return value was [gets $socket]"
close $socket
```

**Interfacing via COM port**

1) Connect to the COM port via a client such as PuTTY.

![PuTTY](image)

2) Send ASCII commands via the command prompt:

```
::FOR /l %%A IN (1, 1, 255) DO ECHO "setIP 192.168.1.15" > \COM%%A
```

6) Send ASCII commands via the command prompt, but do it without needing to know the COM port name in advance. To run this code, put it in a .bat file and run.

```
::FOR /l %%A IN (1, 1, 255) DO ECHO "setIP 192.168.1.15" > \COM%%A
```

This will iterate over all COM ports and try to send the ASCII command “setIP 192.168.1.15”
Programming the iGen interference traffic and waveform generator

For instructions on operating the iGen module via a web browser interface, please refer to the octoBox iGEN Interference Generator Quick Start Guide document.

Following is the documentation on programming the iGen via its API.

Overview

The iGen device provides a socket server listening at TCP port 8889 that receives text commands, processes them, and returns text responses.

A received text command is formatted as a sequence of text tokens separated by spaces, with a carriage return and line feed terminating the command after the last token.

The format of the text response depends on the received command, but generally the response is a combination of text and whitespace terminated by a carriage return and line feed. Example commands and responses are specified in this document using the style of C language syntax. An example response is "0\r\n".

This document defines the commands that can be received and the responses that will be generated.

Unrecognized commands always generate a response of "-1\r\n".

This document's section headers specify for each command the fixed set of tokens that identify the command, and zero or more variable parameters. Parameters names end in an underscore followed by a specification of the parameter data type, and are enclosed in brackets.

_bool boolean 0 is false, nonzero is true
_sl signed long integer [-2147483648 ... 2147483647]
_ul unsigned long integer [0 ... 4294967296]
_s string text
_u unsigned integer [0 ... 65535]
Network Configuration Commands
The network configuration commands provide control and readout of the Internet Protocol settings for
the PoE (Power over Ethernet) and Wi-Fi communication interfaces.

IP addresses and subnet masks are specified using dotted-decimal notation, i.e. x.y.z.n . The IP address
consists of four decimal numbers each in the range [0 ... 255], separated from each other by the period
(dot) character.

netconfig ether dhcp get address
Returns the IP address that has been received from the external DHCP server for the PoE ethernet port.
Examples follow.

"192.168.15.5\n" DHCP success, IP address is 192.168.15.5
"169.254.17.143\n" DHCP failure, IP address is 169.254.17.143
"\n" DHCP client is disabled

netconfig ether dhcp get enable
Returns the DHCP client enable status of the PoE Ethernet port. Examples follow.

"0\n" DHCP client is not enabled
"1\n" DHCP client is enabled

netconfig ether dhcp get subnet
Returns the IP subnet mask that has been received from the external DHCP server for the PoE ethernet
port. Examples follow.

"255.255.255.0\n" IP subnet mask is 255.255.255.0
"\n" DHCP is disabled

netconfig ether dhcp set enable [en_bool]
Sets the enable state for the DHCP client of the PoE Ethernet port.

This command is the only one that causes the Ethernet network configuration to be transferred to the
hardware for processing, so this command should be last in any sequence that updates multiple
Ethernet network configuration parameters.

When the iGen DHCP client is enabled, the DHCP client attempts to obtain valid values for IP address
and subnet mask. If no DHCP server is found, the device will auto-generate an address in the range
[169.254.0.0 ... 169.254.255.255].

When the iGen DHCP client is disabled, the static IP address and subnet mask are used to configure the
PoE Ethernet port.
0
disable DHCP client
nonzero
enable DHCP client

Response values follow.

"0\n" enable state updated successfully
"-1\n" enable state not updated

netconfig ether get mac
Returns the MAC address of the PoE ethernet port. Example follows.

"70:B3:D5:EF:30:5C\n" MAC address is 70:B3:D5:EF:30:5C

netconfig ether static get address
Returns the static IP address of the PoE Ethernet port. Example follows.

"192.168.15.5\n" IP address is 192.168.15.5

netconfig ether static get subnet
Returns the static IP subnet mask of the PoE Ethernet port. Example follows.

"255.255.255.0\n" IP subnet is 255.255.255.0

netconfig ether static set address [ipaddr_s]
Sets the static IP address of the PoE Ethernet port.

Response values follow.

"0\n" ipaddr_s is specified with correct format and static IP address is updated
"-1\n" static IP address is not updated

netconfig ether static set subnet [ipaddr_s]
Sets the static IP subnet mask of the PoE Ethernet port.

Response values follow.

"0\n" ipaddr_s is specified with correct format and IP subnet is updated
"-1\n" static IP subnet mask is not updated

netconfig get hostname
Returns the network host name of the iGen. Example follows.

"iG50211-03\n" host name matches device serial number
netconfig wifi dhcp get address
Returns the IP address that has been received from the external DHCP server for the Wi-Fi radio.
Examples follow.

"192.168.15.7
DHCP success, IP address is 192.168.15.7
"169.254.250.12
DHCP failure, IP address is 169.254.250.12
"\r\n"  
DHCP client is disabled

netconfig wifi dhcp get enable
Returns the DHCP client enable status of the Wi-Fi radio. Example follows.

"0\r\n"  
DHCP client is not enabled
"1\r\n"  
DHCP client is enabled

netconfig wifi dhcp get subnet
Returns the IP subnet mask that has been received from the external DHCP server for the Wi-Fi radio.
Examples follow.

"255.255.255.0
IP subnet mask is 255.255.255.0
"\r\n"  
DHCP is disabled

netconfig wifi dhcp set enable [en_bool]
Sets the enable state for the DHCP client of the Wi-Fi radio.

This command is the only one that causes the Wi-Fi radio network configuration to be transferred to the hardware for processing, so this command should be last in any sequence that updates multiple Wi-Fi radio network configuration parameters.

When the iGen DHCP client is enabled, the DHCP client attempts to obtain valid values for IP address and subnet mask. If no DHCP server is found, the device will auto-generate an address in the range [169.254.0.0 ... 169.254.255.255].

When the iGen DHCP client is disabled, the static IP address and subnet mask are used to configure the PoE Ethernet port.

0  
disable DHCP client
nonzero  
enable DHCP client

Response values follow.

"0\r\n"  
enable state updated successfully
"-1\r\n"  
enable state not updated
**netconfig wifi get mac**
Returns the MAC address of the Wi-Fi radio. Example follows.

"04:F0:21:0D:4A:EF\n"  MAC address is 04:F0:21:0D:4A:EF

**netconfig wifi static get address**
Returns the static IP address of the Wi-Fi radio. Example follows.

"192.168.15.7\n"  IP address is 192.168.15.7

**netconfig wifi static get subnet**
Returns the static IP subnet mask of the Wi-Fi radio. Example follows.

"255.255.255.0\n"  IP subnet is 255.255.255.0

**netconfig wifi static set address [ipaddr_s]**
Sets the static IP address of the Wi-Fi radio.

Response values follow.

"0\n"  ipaddr_s is specified with correct format and static IP address is updated

"-1\n"  static IP address is not updated

**netconfig wifi static set subnet [ipaddr_s]**
Sets the static IP subnet mask of the Wi-Fi radio.

Response values follow.

"0\n"  ipaddr_s is specified with correct format and IP subnet is updated

"-1\n"  static IP subnet mask is not updated
**System Commands**
The system commands configure and control modes of operation that affect more than one subsystem.

**system get atten**  
Returns the attenuation level of the RF output. Examples follow.

"0\n" RF attenuation is set to 0 dB  
"15.5\n" RF attenuation is set to 15.5 dB

**system get mode**  
Returns the primary mode of the iGen device. Example follows.

"radio\n"

**system set atten [db_s]**  
Sets the attenuation level of the RF output.

The db_s string value is parsed using C-style sscanf format string "%u.%u" and then converted to a decimal representation. Only values in the range [0 ... 63] are accepted. Decimal values are rounded down to the nearest half-dB. Example: "15.75" is converted to 15.5 dB.

Response values follow.

"0\n" db_s converted successfully and attenuation level changed if necessary  
"-1\n" db_s not converted successfully and attenuation level not changed

**system set mode [mode_s]**  
Sets the primary mode of the iGen device. There are five valid values for mode_s.

startup initial mode set at powerup  
standby disable RF output  
cw continuous wave RF output  
pulse on-off keying (OOK) pulsed RF output  
radio 802.11 radio RF output

Example follows.

"system set mode standby\n"

Response values follow.

"0\n" mode_s is valid and system mode is updated  
"-1\n" mode_s is invalid and system mode is not updated
**system version get command**
Returns the software version identifier of the command server subsystem. Example follows.

"0.13.4\n"  web server version is 0.13.4

**system version get eos**
Returns the software version identifier of the embedded operating system. Example follows.

"201503041917\n"  embedded operating system version is 201503041917

**system version get firmware**
Returns the software version identifier of the firmware subsystem. Example follows.

"3.0.14\n"  firmware version is 3.0.14

**system version get web**
Returns the software version identifier of the web server subsystem. Example follows.

"0.1.0\n"  web server version is 0.1.0

**system version get wifi**
Returns the software version identifier of the Wi-Fi radio subsystem. Example follows.

"1.5\n"  Wi-Fi radio version is 1.5

**system version update [filename_s]**
Updates the system firmware using the contents of a specified firmware file.

The specified firmware file must have been already loaded onto the iGen system by using a web client to access the iGen web server.

The filename_s value must specify an absolute path to the firmware file, and the path delimiter character is '/'. This means every filename_s must start with '/'. An example for filename_s is "/201505010845.tar.gz".

After response to this command is returned, if the response was zero, the version update process will automatically start. This may cause the iGen command and web servers to become temporarily unresponsive.

Response values follow.

"0\n"  filename_s specifies a firmware file that is ready for update
"-1\n"  file error (e.g. file not found, file wrong format, file empty)
Waveform Commands

The waveform commands configure and control the iGen waveform generator subsystem. The waveform commands must be preceded by a system command that selects continuous wave or on-off keying (OOK) pulse mode. The waveform commands provide control of output frequency, level, and OOK pulsing parameters.

There are four parameters that control the OOK pulsing mode.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>pulse width</td>
</tr>
<tr>
<td>off</td>
<td>inter-pulse gap</td>
</tr>
<tr>
<td>count</td>
<td>pulses per burst</td>
</tr>
<tr>
<td>period</td>
<td>interval between start of pulse bursts</td>
</tr>
</tbody>
</table>

**waveform pulse get count**
Returns the burst pulse count setting of the OOK pulsing parameters. Example follows.

"16\r\n"

**waveform pulse get enable**
Returns the mode setting of the OOK pulsing parameters. Only two responses are specified.

"0\r\n" pulse mode is not enabled
"1\r\n" pulse mode is enabled

When the pulse mode is not enabled, the waveform generator output is a continuous wave with no pulsing.

**waveform pulse get off**
Returns the pulse off time setting of the OOK pulsing parameters, in microseconds. Example follows.

"6\r\n"

**waveform pulse get on**
Returns the pulse on time setting of the OOK pulsing parameters, in microseconds. Example follows.
**waveform pulse get period**
Returns the pulse burst period setting of the OOK pulsing parameters, in microseconds. Example follows.

"500"

**waveform pulse set count [count_u]**
Sets the burst pulse count setting of the OOK pulsing parameters. Only values in the range [1 ... 30] are accepted.

Setting does not take effect until "waveform pulse set enable" or "waveform pulse update" command is sent.

Response values follow.

"0" count_u is acceptable and OOK pulsing parameter updated
"[nonzero]" count_u is not acceptable and OOK pulsing parameter not updated
the [nonzero] value is an integer in the range [-1 ... 65535]

**waveform pulse set enable [en_bool]**
Sets the mode setting of the OOK pulsing parameters.

0 OOK mode is disabled if it is not already disabled
nonzero OOK mode is enabled if it is not already enabled and parameters are acceptable

When the pulse mode is not enabled, the waveform generator output is a continuous wave with no pulsing.

When en_bool is nonzero and pulse mode is already enabled, any new OOK pulsing parameters are not applied.

When en_bool is nonzero and pulse mode is not already enabled, the OOK pulsing parameters are evaluated to determine if they are acceptable. Each individual parameter must be within its acceptable range (see other waveform pulse set commands for more information). The following logical equation must be true.

\[
( (on_u + off_u) \ast count_u ) \leq (period_u \ast 0.80)
\]
Response values follow.

"0\n" pulse mode changed

"[nonzero]\n" OOK pulsing parameters are not acceptable, pulse mode remains disabled

the [nonzero] value is an integer in the range [1 ... 65535]

**waveform pulse set off [off_u]**

Sets the pulse off time setting of the OOK pulsing parameters, in microseconds. Only values in the range [1 ... 256] are accepted.

Setting does not take effect until "waveform pulse set enable\n" or "waveform pulse update\n" command is sent.

Response values follow.

"0\n" off_u is acceptable and OOK pulsing parameter updated

"[nonzero]\n" off_u is not acceptable and OOK pulsing parameter not updated

the [nonzero] value is an integer in the range [-1 ... 65535]

**waveform pulse set on [on_u]**

Sets the pulse on time setting of the OOK pulsing parameters, in microseconds. Only values in the range [1 ... 100] are accepted.

Setting does not take effect until "waveform pulse set enable\n" or "waveform pulse update\n" command is sent.

Response values follow.

"0\n" on_u is acceptable and OOK pulsing parameter updated

"[nonzero]\n" on_u is not acceptable and OOK pulsing parameter not updated

the [nonzero] value is an integer in the range [-1 ... 65535]

**waveform pulse set period [period_u]**

Sets the pulse burst period setting of the OOK pulsing parameters, in microseconds. Only values in the range [150 ... 5000] are accepted.

Setting does not take effect until "waveform pulse set enable\n" or "waveform pulse update\n" command is sent.

Response values follow.

"0\n" period_u is acceptable and OOK pulsing parameter updated

"[nonzero]\n" period_u is not acceptable and OOK pulsing parameter not updated

the [nonzero] value is an integer in the range [-1 ... 65535]
**waveform pulse update**
If the pulse mode is already enabled, this command quickly disables and then re-enables the pulse mode. This has the effect of setting the OOK pulsing parameters to the new values that have been set up using the waveform pulse set commands.

Note that when the pulse mode is re-enabled, this may fail because the new OOK pulsing parameters are not acceptable. See the waveform pulse set enable command for more information.

Response values follow.

"0\r\n" OOK pulsing output updated successfully
"-1\r\n" OOK pulsing output not updated successfully

**waveform synth get enable**
Returns the state of the waveform generator output. Only two responses are specified.

"0\r\n" waveform generator output is not enabled
"1\r\n" waveform generator output is enabled

When the waveform generator is not enabled, it is essentially powered down and not connected to the output.

**waveform synth get freq**
Returns the frequency of the waveform generator, in KHz. Example follows.

"5625000\r\n" waveform generator output is 5.625 GHz

**waveform synth set enable [en_bool]**
Sets the enable state of the waveform frequency.

0 disable waveform generator output
nonzero enable waveform generator output

Response values follow.

"0\r\n" enable state updated successfully
"-1\r\n" enable state not updated

**waveform synth set freq [Khz_ul]**
Sets the frequency of the waveform generator.

The khz_ul parameter must be in the range [500000 ... 6000000].
Response values follow.

"0\n"  frequency updated successfully
"-1\n"  frequency not updated
**Wi-Fi Commands**
The Wi-Fi commands configure and control the iGen Wi-Fi radio transmit subsystem. The Wi-Fi commands must be preceded by a system command that selects radio mode.

**wifi channel get bandwidth**
Returns the bandwidth of the Wi-Fi radio output. Example follows.

"20\r\n" Wi-Fi bandwidth is set to 20 MHz

**wifi channel get primary**
Returns the primary channel number setting of the Wi-Fi radio. Example follows.

"12\r\n" primary Wi-Fi channel is set to 12

**wifi channel get secondary**
Returns the secondary channel number list of the Wi-Fi radio.

Secondary channels exist only for interface types ac or n when bandwidth is set to 40 or 80. Examples follow.

"\r\n" there is no secondary channel
"11\r\n" secondary Wi-Fi channel is 11 (i.e. for primary channel 12 with bandwidth 40)
"36,44,48\r\n" secondary Wi-Fi channels are (36,44,48) (i.e. for primary channel 40 with bandwidth 80)

**wifi channel set bandwidth [MHz_sl]**
Sets the bandwidth of the Wi-Fi radio output.

There are four valid values for the MHz_sl parameter, they are: (-1, 20, 40, 80). When the MHz_sl parameter is 0, the Wi-Fi radio automatically manages the bandwidth.

Response values follow.

"0\r\n" MHz_sl value is valid and channel bandwidth is updated
"-1\r\n" MHz_sl value is invalid and channel bandwidth not updated

**wifi channel set primary [chan_u]**
Sets the channel number setting of the Wi-Fi radio. Only values in the following range are accepted.

<table>
<thead>
<tr>
<th>802.11a</th>
<th>[36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 149, 153, 157, 161, 165]</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11b</td>
<td>[1 ... 14]</td>
</tr>
<tr>
<td>802.11g</td>
<td>[1 ... 14]</td>
</tr>
</tbody>
</table>
802.11n  [1 ... 14, 36, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 149, 153, 157, 161, 165]  

Response values follow.

"0\r\n"  chan_u is valid and channel number is updated
"-1\r\n"  chan_u is invalid and channel number is not updated

**wifi config get guard**

Returns the guard interval setting of the Wi-Fi radio. Example follows.

"0\r\n"

**wifi config get interface**

Returns the radio interface type of the Wi-Fi radio. Example follows.

"a\r\n"

**wifi config get mode**

Returns the primary Wi-Fi mode of the Wi-Fi radio. Example follows.

"standby\r\n"

**wifi config get rate**

Returns the channel communication rate setting of the Wi-Fi radio. Example follows.

"54\r\n"  Wi-Fi channel rate is set to 54 Mbps

**wifi config get wmm**

Returns the WMM access priority of the Wi-Fi radio when contention mode is enabled. Example follows.

"video\r\n"

**wifi config set guard [interval_u]**

Sets the guard interval setting of the Wi-Fi radio. Only values in the following range are accepted.

0  short interval
1  long interval

Response values follow.

"0\r\n"  interval_u is valid and guard interval is updated
"-1\r\n"  interval_u is not valid and guard interval is not updated
**wifi config set interface [type_s]**
Sets the radio interface type of the Wi-Fi radio. There are five valid values for type_s.

- a 802.11a interface
- ac 802.11ac interface
- b 802.11b interface
- g 802.11g interface
- n 802.11n interface

Example follows.

"wifi config set interface ac"

Response values follow.

"0" type_s is valid and radio interface type is updated
"-1" type_s is invalid and radio interface type is not updated

**wifi config set mode [mode_s]**
Sets the primary Wi-Fi mode of the Wi-Fi radio. There are five valid values for mode_s.

- AP access point mode
- capture packet capture mode
- client client mode
- playback packet playback mode
- standby standby mode - no TX output

Example follows.

"wifi config set mode playback"

Response values follow.

"0" mode_s is valid and Wi-Fi mode is updated
"-1" mode_s is invalid and Wi-Fi mode is not updated

**wifi config set rate [rate_sl]**
Sets the channel communication rate setting of the Wi-Fi radio. Only values in the following range are accepted. When the rate_sl value is -1, rate setting is automatically controlled by the Wi-Fi radio.

- 802.11a Mbps [-1, 6, 9, 12, 18, 24, 36, 48, 54]
- 802.11ac MCS [-1, 0 ... 9]
- 802.11b Mbps [-1, 1, 2, 5.5, 11]
802.11g  Mbps  [-1, 1, 2, 6, 9, 12, 18, 24, 36, 48, 54]
802.11n  MCS  [-1, 0 ... 9]

Response values follow.

"0\r\n"  rate_sl is valid and channel rate is updated
"-1\r\n"  rate_sl is invalid and channel rate is not updated

**wifi config set wmm [access_s]**
Sets the WMM access priority of the Wi-Fi radio. There are five valid values for access_s.

- **no-contention**  transmit without regard to access priority
- **background**  background access priority
- **best-effort**  best-effort access priority
- **video**  video access priority
- **voice**  voice access priority

Example follows.

"wifi config set wmm best-effort\r\n"

Response values follow.

"0\r\n"  access_s is valid and access priority is updated
"-1\r\n"  access_s is invalid and access priority is not updated

**wifi playback get filename**
Returns the name of the PCAP file that is or will be used during Wi-Fi radio packet playback. Example follows.

"/pcap/capture-2015-01-01.pcap\r\n"

**wifi playback get ips_duration**
Return the inter-packet spacing of the Wi-Fi radio, in microseconds. Example follows.

"5\r\n"  inter-packet spacing is set to 5 microseconds

**wifi playback get loopback**
Returns the continuous loopback mode of the Wi-Fi radio packet playback.

"0\r\n"  loopback mode is not enabled
"1\r\n"  loopback mode is enabled

**wifi playback get run en_bool**
Returns the run state of the Wi-Fi radio packet playback.
wifi playback set filename [name_s]
Sets the name of the PCAP file that will be used during Wi-Fi radio packet playback.

The PCAP file must have been already loaded onto the iGen system by using a web client to access the iGen web server.

The name_s value must specify an absolute path to the PCAP file, and the path delimiter character is '/'. This means every name_s must start with '/'. An example for name_s is "/example.pcap".

Response values follow.

"0\n" name_s specifies a PCAP file that is ready for playback
"-1\n" file error (e.g. file not found, file wrong format, file empty)

wifi playback set ips_duration [usec_sl]
Sets the inter-packet spacing of the Wi-Fi radio, in microseconds.

If the usec_sl value is -1, the inter-packet spacing is determined by the contents of the playback PCAP file. If the usec_sl value is 0, the inter-packet spacing used is the minimum valid IPS value.

Response values follow.

"0\n" inter-packet spacing is updated
"-1\n" inter-packet spacing is not updated

wifi playback set loopback [en_bool]
Sets the continuous loopback mode of the Wi-Fi radio packet playback.

0 disable loopback mode
nonzero enable loopback mode

Response values follow.

"0\n" loopback mode is updated
"-1\n" loopback mode is not updated

wifi playback set run [en_bool]
Sets the run state of the Wi-Fi radio packet playback.
This command is the only one that causes the wifi parameters to be transferred to the hardware for processing, so this command should be last in any sequence that updates multiple parameters and starts or stops a playback run.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stop playback</td>
</tr>
<tr>
<td>nonzero</td>
<td>start playback</td>
</tr>
</tbody>
</table>

Response values follow.

"0\n" run state is updated
"-1\n" run state is not updated
Programming Example
The following examples use pseudo-code to illustrate the proper use of the iGen socket commands.

The socket.send() function is used by a client to send command data to the iGen socket server. The socket.recv() function is used by the client to receive response data from the iGen socket server.

continuous wave mode
The following sequence demonstrates the setup required for continuous wave RF output at 5.625 GHz.

```plaintext
socket.send("system set mode cw\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("system set atten 0\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("waveform pulse set enable 0\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("waveform synth set freq 5625000\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("waveform synth set enable 1\r\n");
r = socket.recv(); /* r should be "0\r\n" */
Read back the frequency.

socket.send("waveform synth get freq\r\n");
r = socket.recv(); /* r should be "5625000\r\n" */
```

OOK pulse mode
The following sequence demonstrates the setup required for OOK pulsing RF output.

```plaintext
socket.send("system set mode pulse\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("system set atten 0\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("waveform pulse set enable 0\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("waveform pulse set count 16\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("waveform pulse set off 10\r\n");
r = socket.recv(); /* r should be "0\r\n" */
```
socket.send("waveform pulse set on 5\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("waveform pulse set period 500\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("waveform pulse set enable 1\r\n");
r = socket.recv(); /* r should be "0\r\n" */

Change the OOK pulsing parameters.

socket.send("waveform pulse set count 10\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("waveform pulse set period 250\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("waveform pulse update\r\n");
r = socket.recv();
/* r should be "count 10\r\n|enable 1\r\n|off 10\r\n|non 5\r\nnperiod 250\r\n" */

wifi playback start

The following sequence demonstrates the setup required to start Wi-Fi radio packet playback.

socket.send("system set mode radio\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("wifi channel set bandwidth 80\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("wifi channel set number 124\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("wifi config set guard 0\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("wifi config set interface ac\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("wifi config set mode playback\r\n");
r = socket.recv(); /* r should be "0\r\n" */

socket.send("wifi config set rate 9\r\n");
r = socket.recv(); /* r should be "0\r\n" */
socket.send("wifi config set wmm video\r\n");
r = socket.recv();          /* r should be "0\r\n" */

socket.send("wifi playback set filename /pcap/example.pcap\r\n");
r = socket.recv();          /* r should be "0\r\n" */

socket.send("wifi playback set ips_duration 0\r\n");
r = socket.recv();          /* r should be "0\r\n" */

socket.send("wifi playback set loopback 1\r\n");
r = socket.recv();          /* r should be "0\r\n" */

socket.send("wifi playback set run 1\r\n");
r = socket.recv();          /* r should be "0\r\n" */

wifi playback stop
The following sequence demonstrates the setup required to stop Wi-Fi radio packet playback.

socket.send("wifi playback set run 0\r\n");
r = socket.recv();          /* r should be "0\r\n" */
References

SmallNetBuilder’s Wireless Testbed V2 paper reprint available here:
http://www.octoscope.com/English/Collaterals/Articles/octoScope_SmallNetBuilder's_Wireless_Testbed.pdf
Support
For support, please call +1.978.222.3114 or email support@octoscope.com

Visit us online at www.octoscope.com