Installing and Using Appium

Table of contents

HSAppium Server

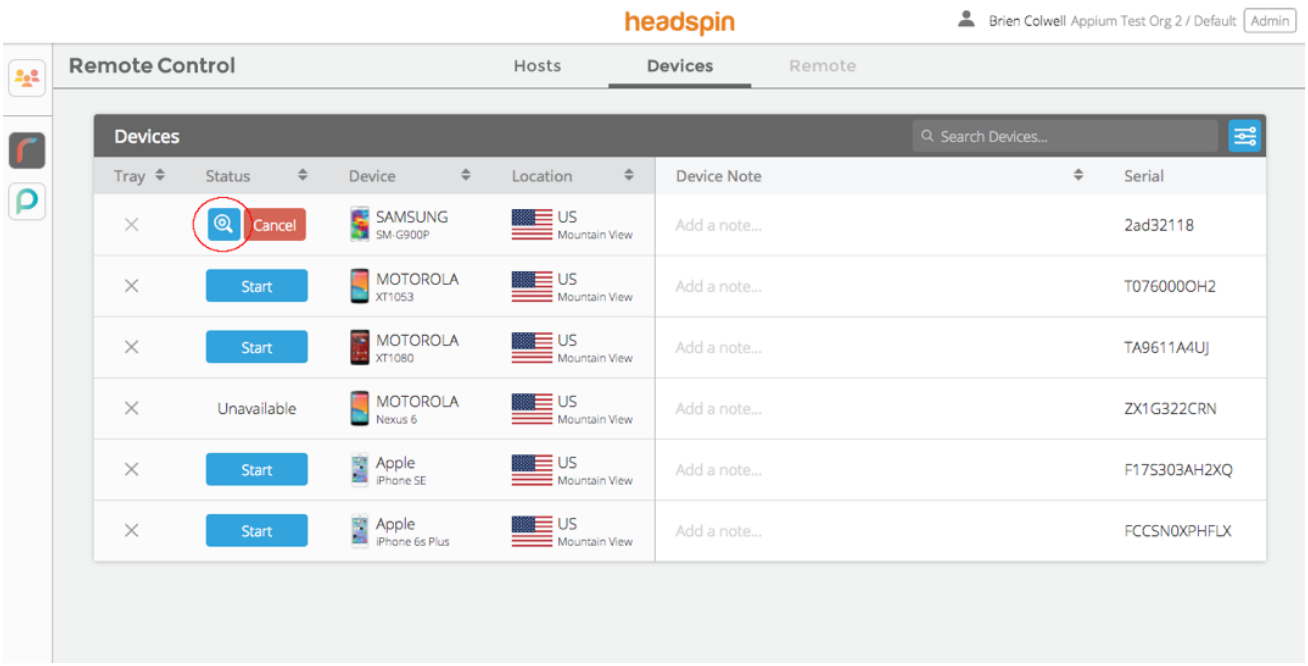
Introduction

The HeadSpin platform runs an HSAppium Server on each host that intelligently manages parallel testing of devices, and integrates appium sessions with the HeadSpin platform authentication, lock, scheduler, and UI. Developers need to change the appium driver URL used to initiate the remote web driver, but all other aspects of the appium script remain unchanged. The main change is to parameterize the driver URL to be passed into the script, from an env var or similar, or to integrate the HeadSpin API as part of the driver setup. For example instead of `http://localhost:4723/wd/hub` the driver URL might be `https://acme-us-sf-0-proxy-0.headspin.io:7201/v0/99fjsa9/wd/hub`. The HeadSpin API server provides the appium driver URL per host via a HTTP GET.

Architecturally, the HSAppium Server wraps a real appium server per device. The test runner client no longer starts and stops the appium server process; instead, the version passed in the `appiumVersion` capability will tell the proxy to choose an installed version to use. The proxy intelligently starts and stops

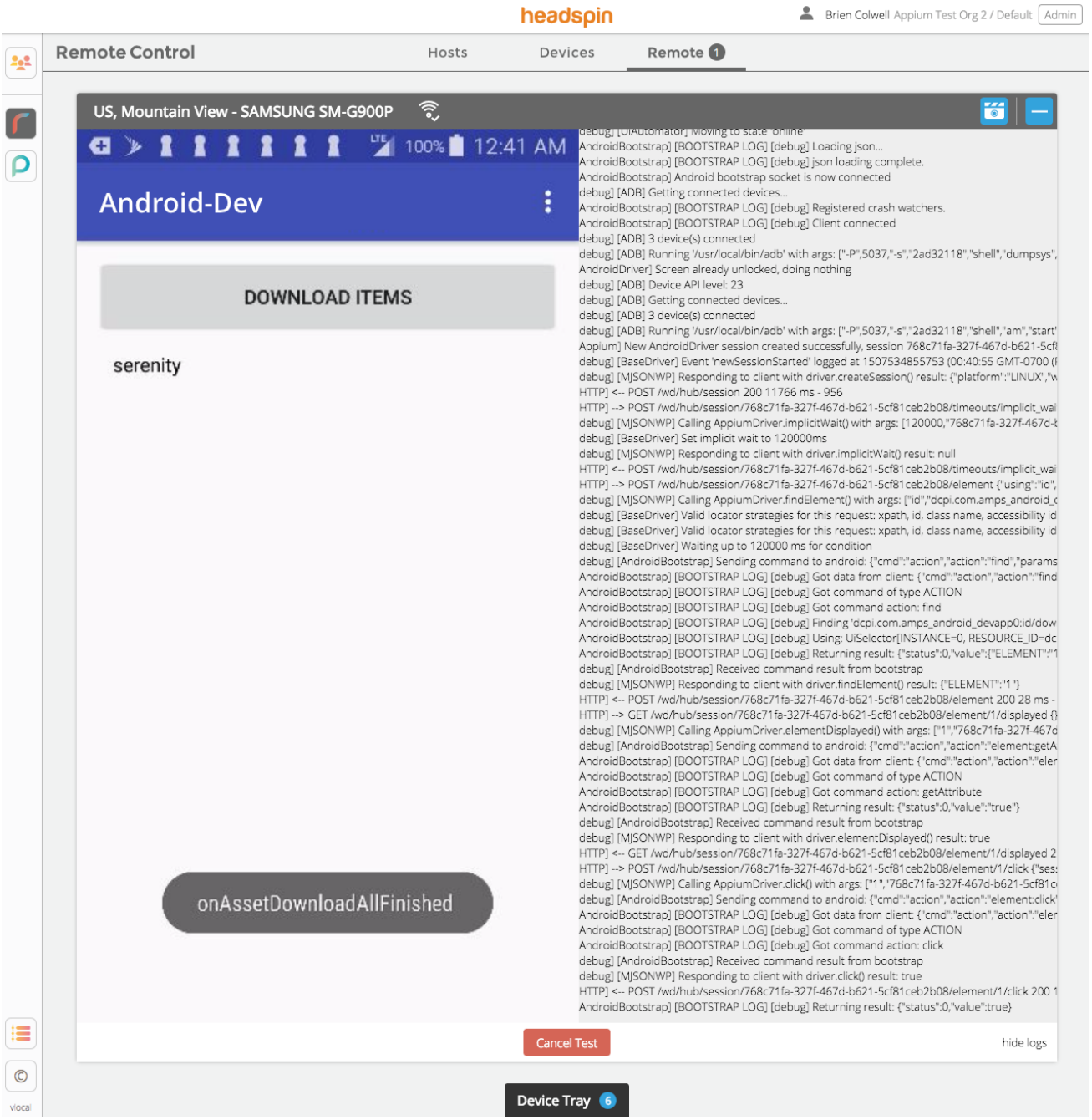
the appium server process, as well as performs cleanup, and tries to address common issues with the appium platform. Tests running with the appium proxy just work, with no additional maintainance of the appium server needed.

A test running via the appium proxy gives the user the ability to view the test, logs, and cancel the test from the UI. In the device list UI, click the looking glass icon next to the red cancel button to view the Appium test for a device. The looking glass icon is circled in red below. You will only see the looking glass icon if you are currently logged in as the same user that is running the test.



Device List with Running Tests

Once the looking glass icon is clicked the user will be taken to a screen that displays the running test on the device. The appium logs can be seen by clicking the ‘view logs’ button.



Device View of a Running Test

Appium API

The appium proxy driver URL per host follows the pattern `https://{host}:{port}/v0/{api_token}/wd/hub`. To find the driver URL for a device, either use the `hs` commmand line tool or by sending an HTTP GET request to the HeadSpin api server.

Start by installing the `hs` command line tool if you do not already have it:

```
$ pip install headspin-cli
```

Then configure the api server as specified in your HeadSpin deployment. Ask your local administrator for details on the HeadSpin deployment.

```
$ hs config server api.headspin.io
```

`api.headspin.io` with your api server.

Use the `appium url` sub-command of `hs` to get the driver url for a specific device. The api token must be passed as an argument to `-t`.

```
$ hs appium url <device id> -t <api token>
https://acme-us-mv-0-proxy-2.headspin.io:7203/{api_token}/wd/hub
```

resulting URL is the appium driver url. Be sure to replace `{api_token}` in the URL with your actual api token.

You can also send an HTTP GET request to the api server at `https://{api_host}/v0/devices/automation-config` to get the driver URL from the API server, passing the API token as the header `Authorization: Bearer {api_token}`. The response body will be a JSON list of objects, in the shape below, where the field `driver_url` is the driver URL.

For example,

```
$ curl -H 'Authorization: Bearer a53221c8ad3e11e786fbd8eb97b81a38' https://api.headspin.io/v0/dev
ices/automation-config
```

Replace the API token, `a53221c8ad3e11e786fbd8eb97b81a38`, and api host with your own.


```
{
  "TA9611A4UJ": {
    "lock_url": "https://us-mvo.headspin.io:7200/v0/{api_token}/lock",
    "working_dir": "/home/acme/appium_tests/TA9611A4UJ",
    "capabilities": {
      "udid": "TA9611A4UJ",
      "deviceName": "TA9611A4UJ",
      "platformName": "Android",
      "autoAcceptAlerts": true,
      "noReset": true
    },
    "host": "proxy-us-mvo-1.headspin.io",
```

```
      "unlock_url": "https://us-mvo.headspin.io:7200/v0/{api_token}/unlock",
      "driver_url": "https://us-mvo.headspin.io:7200/v0/{api_token}/wd/hub",
      "os": "android",
      "control_url": "https://us-mvo.headspin.io:7100"
    },
    "461454f875c56c95c9080ac17767eccc55953be4": {
      "lock_url": "https://us-mvo.headspin.io:7201/v0/{api_token}/lock",
      "working_dir": "/Users/acme/appium_tests/461454f875c56c95c9080ac17767eccc55953be4",
      "capabilities": {
        "udid": "461454f875c56c95c9080ac17767eccc55953be4",
        "deviceName": "iPhone",
        "automationName": "XCUITest",
        "useSimpleBuildTest": true,
        "platformName": "iOS"
      },
      "host": "proxy-us-mvo-2.headspin.io",
      "unlock_url": "https://us-mvo.headspin.io:7201/v0/{api_token}/unlock",
      "driver_url": "https://us-mvo.headspin.io:7201/v0/{api_token}/wd/hub",
      "os": "ios",
      "control_url": "https://us-mvo.headspin.io:3000"
    }
  }
```


In this example, the driver url for device `TA9611A4UJ` is


```
      "driver_url": "https://us-mvo.headspin.io:7200/v0/{api_token}/wd/hub",
```


Assuming the api token is `a53221c8ad3e11e786fbd8eb97b81a38`, the complete url will be

```
https://us-mvo.headspin.io:7200/v0/a53221c8ad3e11e786fbd8eb97b81a38/wd/hub
```

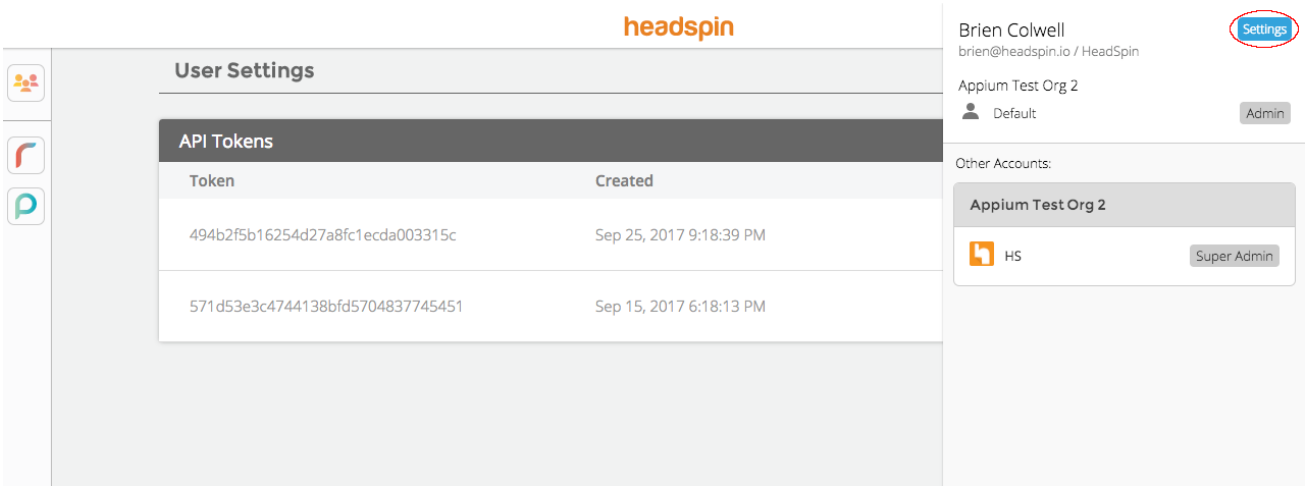
The `capabilities` object per device should be treated as a default set of capabilities. While the final set of capabilites passed to the appium proxy can not include these, they are recommended for stability.

For iOS, the capabilities can omit the capabilites `xcodeConfigFile`, `keychainPath`, and `keychainPassword`. When all three are missing, the proxy will fill them using the remotecontrol credentials already installed on the system. This is the recommended approach, which reduces the burden to maintain the provisioning profiles and keys on the host.

API Tokens

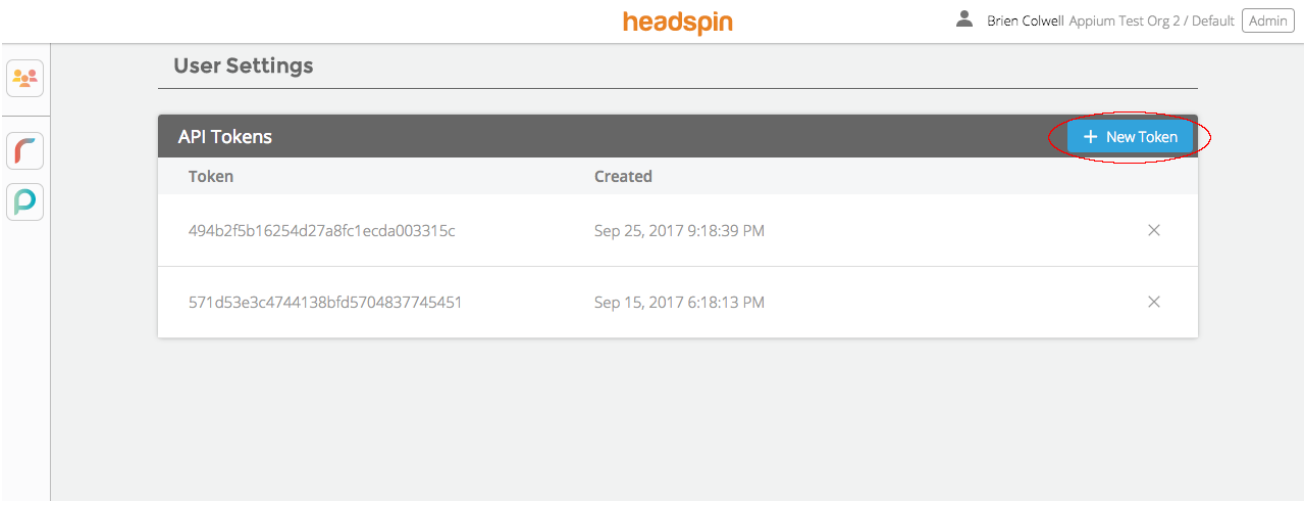
API tokens must be created on an Admin or Team Member user, using either the Settings section in the top-right dropdown, or from the `hsops` command line tool.

Start by clicking on the `Settings`



Auth Settings

Then click the `+ New Token` to add a new api token.



API Tokens List

Creating API tokens on the command line can be done with the `hsops` tool. The relevant `hsops` commands are:

```
hsops api-token ls
hsops api-token create --user_id=<user_id> --role_id=<role_id>
hsops api-token delete <token>
```

HSAppium Proxy

Sometimes rather than changing the code that starts the appium process, and client driver URL and log locations, it's faster to use the `hsappium` server to drop into an existing workflow. `hsappium` is a direct connection to an appium server process, and is compatible with the port command line option from `appium`. It prints the session logs to standard out. The hsappium process can be started and stopped on different ports, to point to the local appium proxy.

For example, a script looks like the first block below can be changed to the second block below to use the appium proxy. All functionality when the script runs remains the same.

```
appium -p 4723 &
pid=$!
./my_test.sh
kill $pid
```

HSAppium Proxy

```
hsappium -p 4723 -t <api_token> &
pid=$!
./my_test.sh
kill $pid
```

Using Customer Appium Server

In the case that changing the driver URL and using hsappium is not possible, a script `hslock` is provided to allow a custom appium process to integrate with the core aspects of the device lock and scheduler, and also provide a basic UI integration. The command is below. `hslock lock` must be called before the web driver client connects, and `hslock unlock` must be called when the test is done.

```
hslock -t <api_token> -u <device_id> [lock | unlock] [--log=<log path>]
```

`hslock lock` blocks until the device is locked. A caller must check the exit code to tell if the lock was acquired successfully (0 is success). When using `hslock`, the view mode is always enabled. The logs are shown in the UI if `--log=` is passed. The cancel button is never shown in the UI.

Installing custom appium versions

HeadSpin comes packaged with a modern stable release of appium. However, more versions of appium can be installed and chosen using a capability named `appiumVersion`. Note: currently, we only support appium versions compatible with Node version 8.

Under `/home/hs/headspinio-pboxagent/appium` (Ubuntu) and `/Users/hs/headspinio-pboxagent/appium` (macOS) there is one dir per version, e.g. `1.6.5`. Create a new dir to match the version being installed, then under it create this structure.

```
{version}/
  node_modules/
    .bin/
      appium # executable to start the server
```

This structure follows the structure created by `npm` when appium is installed using `npm`. To install appium directly onto the host using `npm` one can do:

```
hsenter pboxagent
mkdir -p $HOME/headspinio-pboxagent/appium/{version}/node_modules
cd $HOME/headspinio-pboxagent/appium/{version}
npm install appium@{version}
exit
```

On Mac OS X, the following command is necessary in order to enable manual signing on appium's WebDriverAgent:

```
hsenter remotecontrol
$HOME/headspinio-remotecontrol/{env}-remotecontrol/controlfreak/bin/controlfreak \
    wda force-manual-signing --wda-project-path \
    $HOME/headspinio-pboxagent/appium/{version}/node_modules/appium/node_modules/appium-xcuitest-
driver/WebDriverAgent/WebDriverAgent.xcodeproj
exit
```

If the host is not internet-enabled, one could install appium on a different host that is internet-enabled and then copy the install directory to the real host. (In this case, the architecture of the two hosts has to match. E.g., one cannot install appium on an macOS host and then copy it to a Ubuntu host.) To install an appium version using this method, one can do (on the internet-enabled host):

```
mkdir test # a temporary directory where to install appium
mkdir test/node_modules
cd test
```

```
npm install appium@{version}
# create archive:
tar zcvf appium-{version}.tar.gz *
```

Then, copy the `appium-{version}.tar.gz` archive onto the HeadSpin host, log into the HeadSpin host, and do:

```
mkdir -p $HOME/headspinio-pboxagent/appium/{version}
cd $HOME/headspinio-pboxagent/appium/{version}
# unpack the archive:
tar zxvf /path/to/appium-{version}.tar.gz
```

On Mac OS X, the following command is necessary in order to enable manual signing on appium's WebDriverAgent:

```
hsenter remotecontrol
$HOME/headspinio-remotecontrol/{env}-remotecontrol/controlfreak/bin/controlfreak \
    wda force-manual-signing --wda-project-path \
    $HOME/headspinio-pboxagent/appium/{version}/node_modules/appium/node_modules/appium-xcuitest-
driver/WebDriverAgent/WebDriverAgent.xcodeproj
exit
```

Finally, restart HeadSpin:

```
$HOME/headspinio-pboxagent/{env}-pboxagent/start.sh
```

The admin needs to install appium using the above structure, and restart HeadSpin. The proxy will automatically make the new version available. Users can use this new version by setting the `appiumVersion` capability in their test script. For example, if version `1.7.1` was installed, one can pass `"appiumVersion": "1.7.1"` in the capability dictionary and the proxy will run the test using the new `1.7.1` version. (Note: if multiple versions are installed, and the `appiumVersion` capability is not specified, the appium proxy will choose the highest version available.)

Logs

By default an appending log is stored per device, at `/var/log/headspinio-{env}/appium/{device_id}.log`. Tailing this log will provide the latest output of the test running on the device. A test may pass in the `sessionLog` capability as a path on the local system, to store an additional log per test session. This is useful, for example, when debugging in an environment with many running tests, to see the logging from only one test session.

Uploading the test app

Test apps small in size can be installed on the device using the adb api, for example:

```
curl -X POST https://{env}-api.headspin.io/v0/adb/{serial}/install --data-binary @/local/path/to/
app.apk
```

Users may find better performance to upload large test apps to the host using `rsync` and the customer user, e.g. `rsync -i <customer key> -a my_app.ipa acme@acme-us-sf-0-proxy-0.headspin.io:/Users/acme/appium_tests/461454f875c56c95c9080ac17767eccc55953be4/`. The path to upload is referred to as the working directory for the device. This is also referenced in the Appium API return object as `working_dir`. A typical Jenkins job running an automated upload would parameterize it using a shell script show below.

```
# DEVICE_ID and API_TOKEN are defined as env vars
function get_prop() {
    prop_name="$1"
    curl -H "Authorization: Bearer ${API_TOKEN}" https://acme-api.headspin.io/v0/devices/automati
on-config | jq -r ".\"$DEVICE_ID\".\"$prop_name\""
}
HOST=`get_prop host`
WORKING_DIR=`get_prop working_dir`
rsync -i <customer key> -a my_app.ipa acme@$HOST:$WORKING_DIR/
```

Further Resources

Below are links to the latest set of capabilities for Android and iOS (via xctestrunner).

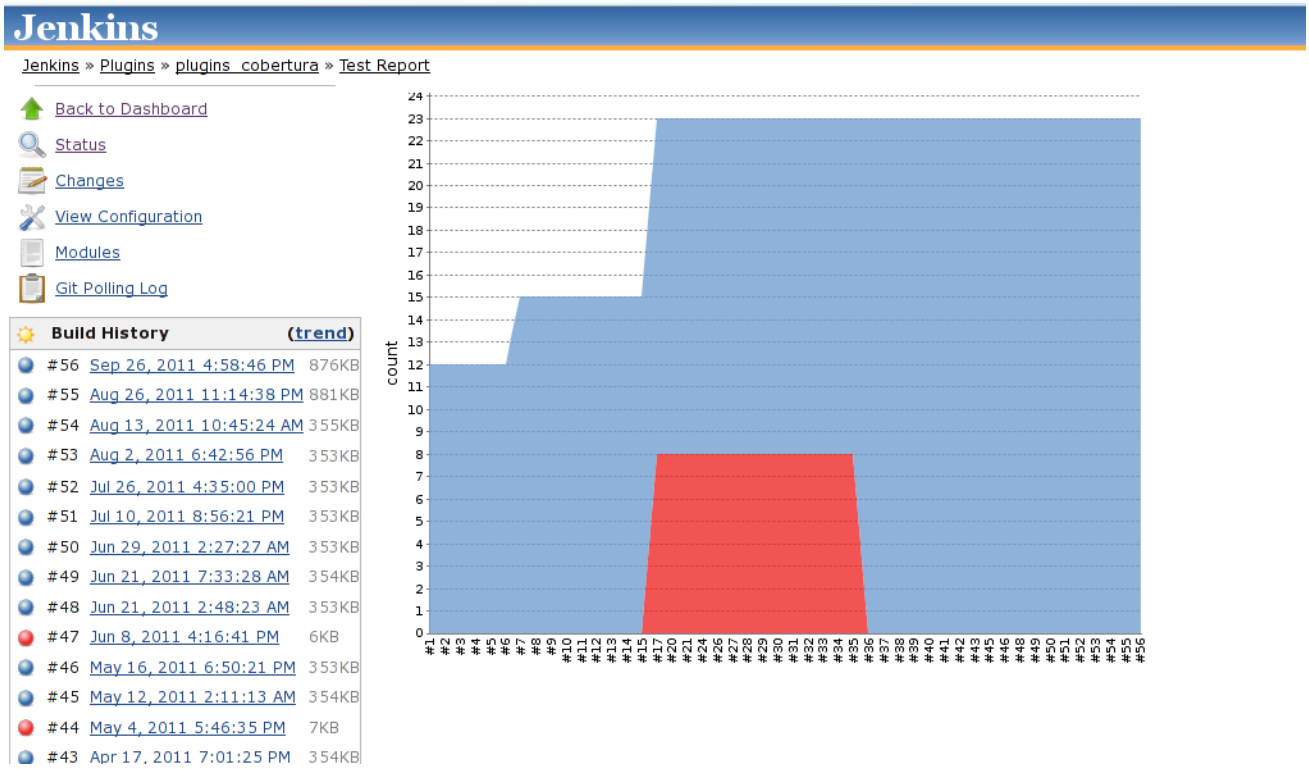
Platforms	Link
Common, Android	https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md#general-capabilities
iOS	https://github.com/appium/appium-xcuitest-driver#desired-capabilities

Exporting Pass/Fail Reports

The standard output format for test reports is the JUnit XML report. An example of an XML report generated by python `xmlrunner.XMLTestRunner` is below.

```
<?xml version="1.0" ?>
<testsuite errors="0" failures="0" name="AppTests-20171108153054" tests="2" time="81.136">
  <testcase classname="AppTests" name="test_download_individual_item" time="46.395"/>
  <testcase classname="AppTests" name="test_items_download" time="34.741"/>
  <system-out><![CDATA[...]]></system-out>
  <system-err><![CDATA[...]]></system-err>
</testsuite>
```

These reports are collected by the CI system running the tests and given to a report generation plugin. Most CI systems have graphing facilities that can be integrate these reports into a total pass/fail trend line. An example integration with Jenkins is shown below.



Jenkins JUnit Test Report

Performance Data

The HeadSpin Appium server by default captures screen video and network data from each test run. The data is added to the performance system when the capability `headspin.testName` is given with a test name to aggregate the data under.

Data capture can be turned off by setting the capability `headspin.createSession=false`.