appium is an open source project. We would love your help. No, really.
appium project
Ways to be involved

Reporting **bugs**
Helping with the **forums**
Helping triage **issues**
Fixing/updating/translating the **docs**
Convincing your company to **sponsor**
Be a part of the **community**
Ways to be a developer

Fixing bugs

Adding new features, new drivers

Participating in the roadmap

... maybe become a committer?
Current project organization

Core team of committers (sets roadmap, etc…)

Track bugs/features on GitHub

Release according to SemVer
Future project organization

Appium as part of e.g., the Node Foundation
Well-defined governance process
Large companies contribute active devs
Expanded roadmap
Larger team of core contributors
More velocity as a project
appium overview
Project vision

Appium as the JSONWP front-end for all the best app automation technologies

Today: UIAutomation, UiAutomator

Tomorrow: XCUITest, UiAutomator2

Future: …?
Technical vision

Automation backends as independent ‘drivers’

Connect to JSONWP if possible, otherwise write the adapting layer

Use modern JS for OSS love and accessibility

Keep things clean and modular
Appium 1.5 rewrite: why?

Code was not modular enough

Used old (ES5) JS, had various gotchas

Confusing async programming paradigm using callbacks and no standardization

Each driver reimplemented the same basic functionality

JSONWP concepts at all levels of the app
Appium 1.5 rewrite: why?

Tight coupling between user request and internal methods

Error handling duplicated up/down request chain

Test suite poorly organized, incomplete

Mix of npm, bash, gulp, grunt for build

CI system not able to keep up with development
Appium 1.5 rewrite: goals

Modularize the code

Break separate modules into NPM packages (each with own tests, CI)

Migrate from ES5 JS to ES2015

Use async/await instead of callback/promise interfaces

Standardize subprocess management
Appium 1.5 rewrite: goals

Abstract common driver behavior
Encapsulate all JSONWP logic in one place
Handle HTTP requests at edge controller only
Rely on JS-level exception handling (thanks to async/await)
Use only NPM/gulp for build—no more reset.sh
Appium 1.5: what’s included?

Complete top-to-bottom rewrite + architecture overhaul

Deprecate server args that are also caps; use
--default-capabilities '{"foo": "bar"}' instead

Capability validation on session creation

Various minor fixes

Basically no new features
appium new architecture
Main package, pulls together everything else and acts as orchestration server. Handles logging, Selenium Grid registration, etc…
appium/appium-base-driver

The abstract base class for any driver. Encapsulates basic session handling, interoperation with web server, etc...
appium/node-mobile-json-wire-protocol

Knows all about the MJSONWP and nothing else. Keeps track of routes, validates their params, and knows how to do JSONWP proxying between drivers if necessary.
Knows how to forward HTTP requests to a JSONWP server, receive a response, and send back to a client as if it were all under one session id.
appium/appium-express

Simple web server with our default configuration. Consumes node-MJSONWP routes to make a working web server.
appium/appium-ios-driver

Main package for UIAutomation/Instruments-based iOS support.

- **appium-uiauto**: enables communication to/from Apple’s UIAutomation layer
- **appium-instruments**: manages the instruments subprocess
- **appium-remote-debugger**: encapsulates communication with webviews/Safari
appium/appium-android-driver

Main package for UiAutomator-based Android support, and Chromedriver support.

- **appium-android-bootstrap**: enables communication to/from Android’s UiAutomation layer
- **appium-uiautomator**: manages the UiAutomator subprocess
- **appium-chromedriver**: manages Chromedriver subprocess and proxying to/from Chromedriver
- **appium-adb**: JS interface for ADB commands
Main package for Instrumentation-based Android support. Builds Selendroid, prepares AUT for Instrumentation, manages Selendroid subprocess, and proxies to/from Selendroid server.
appium/appium-doctor

Now an independent package, allows users to run various checks against their system to assist in troubleshooting Appium issues.
appium/appium-support

Set of utilities used by nearly all Appium packages. Promise wrappers for Node libraries. Temporary file helpers. Etc…
And more...

Packages for logging, setting up an Appium package environment, building Selenium Atoms, etc…
OK, maybe we went a bit overboard…
appium module anatomy
appium/my-module
build/
lib/
test/
gulpfile.js
index.js
package.json
README.md

Transpiled code goes here
ES2015 code goes here
Test code goes here
Build code goes here
Package entrypoint
Don’t forget a README!
If your package logs, import appium-logger and make a logger for your package.

Separate your tests into ‘unit’ and ‘e2e’ directories.

Unit tests end with ‘-specs.js’.

E2E tests end with ‘-e2e-specs.js’.
appium development
[Step 1] (Clone and) pull from GitHub
[Step 2] Update NPM dependencies

rm -rf node_modules
npm install
[Step 3] Verify existing tests pass on your env

gulp once && gulp e2e-test
[Step 4] Watch code for changes to re-transpile

gulp
A ‘gulp’ interlude

gulp transpile - just transpile

gulp watch - watch for changes (same as ‘gulp’)

gulp once - transpiple and run unit tests

gulp e2e-test - run e2e tests

_FORCE_LOGS=1 gulp - show package logging
[Step 5] Make your changes!

(Talk to us first?)
[Step 6] Ensure changes have tests/doc/README
[Step 7] Push code to your fork, make a PR
[Step 8] Ensure that CI passes
[Step 9] Engage with us in the review process
[Step 10] Code is merged; you’re now a contributor!

\o/
Creating a new package

Use appium-package-master

clone http://github.com/appium/appium-package-master

npm install

gulp create -n my-new-package

Update boilerplate and you’re good to go
Contributing to appium/appium?

Until 1.5 is officially released, work is taking place on the ‘1.5’ branch—master still has the old code/architecture in case we need to make patches.

Soon 1.5 will become master and master will become old-master. This will break pulls, sorry!
ES2015 is the new version of JS
https://babeljs.io/docs/learn-es2015/
let & const instead of var

ES5

```javascript
var SOME_VALUE = 'foo';

var myVal = 'bar';
myVal += 'baz';
```

ES2015

```javascript
const SOME_VALUE = 'foo';

let myVal = 'bar';
myVal += 'baz';
```
for/of instead of for/_._.each

ES5

```javascript
for (var i = 0; i < list.length; i++) {
    var element = list[i];
    // do something with element
}

_.each(map, function (value, key) {
    // do something with key and value
})
```

ES2015

```javascript
for (let element of list) {
    // do something with element
}

for (let [key, value] of _.pairs(map)) {
    // do something with key and value
}
```
class instead of prototype

ES5

```javascript
var MyClass = function (opts) {
    // initialize
}
MyClass.prototype.myMethod = function () {
    console.log(this.foo);
}
_.extend(MyClass.prototype, myOtherStuff);
```

ES2015

```javascript
class MyClass extends MyOtherClass {
    initialize (opts) {
        // initialize
    }
    myMethod () {
        console.log(this.foo);
    }
}
```
import instead of require

ES5

```javascript
var foo = require('foo');

var bar = function () {
    return foo.baz('lol');
}

module.exports = bar;
```

ES2015

```javascript
import { baz } from 'foo';

export function bar () {
    return baz('lol');
}
```
async/await instead of callbacks

### ES5

```javascript
function foo (cb) {
    bar(function (err) {
        if (err) return cb(err);
        baz(function (val, err) {
            if (err) return cb(err);
            async.eachSeries(list, function (i, cb) {
                fun(i, function (err) {
                    if (err) return cb(err);
                    otherFun(i, function (newVal, err) {
                        if (err) return cb(err);
                        val += newVal;
                    })
                }, function (err) {
                    if (err) return cb(err);
                    cb(val);
                })
            })
        })
    }, function (err) {
        if (err) return cb(err);
        cb(val);
    })
}
```

### ES2015+

```javascript
async function foo () {
    await bar();
    let val = await baz();
    for (let i of list) {
        if (i % 2 === 0) {
            await fun(i);
        }
        val += await otherFun(i);
    }
    return val;
}
```
teen_process instead of child_process

ES5

```javascript
var spawn = require('child_process').spawn;

function boredTail (filePath, boredAfter, cb) {
  cb = _.once(cb);
  if (!boredAfter) boredAfter = 10000;
  try {
    var proc = spawn('tail', ['-f', filePath]);
  } catch (e) {
    return cb(e);
  }
  proc.on('error', function (err) {
    cb(err);
  });
  proc.stdout.on('data', function (chunk) {
    console.log('STDOUT: ' + chunk.toString('utf8')).trim();
  });
  proc.on('exit', function () {
    cb();
  });
  setTimeout(boredAfter, function () {
    proc.kill();
  });
}
```

ES2015

```javascript
import { SubProcess } from 'teen_process';

async function boredTail (filePath, boredAfter = 10000) {
  let p = new SubProcess('tail', ['-f', filePath]);
  p.on('output', stdout => {
    if (stdout) {
      console.log(`STDOUT: ${stdout.trim()}`);
    }
  });
  await p.start();
  await Bluebird.delay(boredAfter);
  await p.stop();
}
```
...and much more!

Template strings
Default parameters
Arrow functions
Native promises + Generators = async/await
asyncbox/appium-support libraries
appium next steps
How can I get started?

Read through the code, starting with appium master and diving deeper into other packages.

Find bugs that don’t look too difficult and use this presentation as a reference for getting started.

Ping core contributors with your PRs or questions on how to help!
Thanks! Questions?

http://appium.io
https://github.com/appium/appium
@AppiumDevs • @jlipps • @saucelabs