cross-browser-tests-runner Documentation

Release 1.2.3

Reetesh Ranjan

Feb 05, 2018

Contents:

CHAPTER 1

Overview

1.1 What is cross-browser-tests-runner?

It is a tool to help you run cross-browser javascript unit tests and selenium tests on multiple cross-browser testing platforms.

1.2 Why this tool?

This tool was created to test browse.js, with following requirements:

- Be able to run Jasmine 1.x unit tests to cover oldest browsers not supported by other newer versions or other frameworks like Mocha
- Be able to send the Jasmine 1.x test data to server that can display it in a popular test results format (Mocha chosen eventually)
- Be able to collect code coverage data and send to test server that can store it in a widely supported format (lcov) which could be uploaded to any third party code coverage tool/website
- Be able to run tests in parallel on multiple cross-browser testing platforms e.g. browserstack.com, saucelabs.com, crossbrowsertesting.com to minimize build duration

Existing tools were not able to bring all the above pieces together, as was seen while trying to write the tests, and this tool was born.

1.3 Get Started

- Install:

```
$ npm install cross-browser-tests-runner
```

- *Quick Start*: Get started quickly and see some sample tests running

- *How To Test*: See a more detailed description of the testing steps

1.4 References

1.4.1 Configuration Files

- *Platform Configuration*: About how and why we store each platform's supported browser/os configuration locally
- *Browsers YAML*: Syntax of specifying browsers for your test
- *Settings*: Test settings file serving multiple purposes

1.4.2 Executables

- *cbtr-init*: Binary that generates test settings file given an input browsers YAML file
- *cbtr-testem-browserstack-init*: Binary that helps your generate `testem.json` for testing on BrowserStack
- *cbtr-testem-saucelabs-init*: Binary that helps your generate `testem.json` for testing on SauceLabs
- *cbtr-testem-crossbrowsertesting-init*: Binary that helps you generate `testem.json` for testing on CrossBrowserTesting

1.4.3 Components

- *Server*: Description of the test server
- *Native Runner*: Description of the in-built unit and selenium tests runner

1.5 Troubleshooting

- See *Troubleshooting*

CHAPTER 2

Quick Start

2.1 Selenium Tests

First do the following:

- For BrowserStack, complete *BrowserStack common steps*
- For SauceLabs, complete *SauceLabs common steps*
- For CrossBrowserTesting, complete *CrossBrowserTesting common steps*

Replace `{platform}` in the following commands with one of: `browserstack`, `saucelabs`, `crossbrowsertesting`.

```
$ cp -r ./node_modules/cross-browser-tests-runner/samples/ samples/
$ ./node_modules/.bin/cbtr-server --native-runner --config ./samples/cbtr-{platform}-
↪selenium.json
```

2.2 JavaScript Unit Tests

For any of the following sections first do the following:

- For BrowserStack, complete *BrowserStack common steps*
- For SauceLabs, complete *SauceLabs common steps*
- For CrossBrowserTesting, complete *CrossBrowserTesting common steps*

2.2.1 Using In-built Native Runner

Replace `{platform}` in the following commands with one of: `browserstack`, `saucelabs`, `crossbrowsertesting`.

```
$ cp -r ./node_modules/cross-browser-tests-runner/samples/ samples/
$ ./node_modules/.bin/cbtr-server --native-runner --config ./samples/cbtr-{platform}-
↪js-testing.json
```

2.2.2 Using Testem

NOTE: You need to have a `testem.json` with `src_files` or `test_page` setting.

Replace `{platform}` in the following commands with one of: `browserstack`, `saucelabs`, `crossbrowsertesting`.

```
$ ./node_modules/.bin/cbtr-quick-start -p {platform} -r testem
$ ./node_modules/.bin/cbtr-server &
$ testem ci
```

2.3 Common Steps

2.3.1 BrowserStack

```
$ export BROWSERSTACK_USERNAME=<your-browserstack-username>
$ export BROWSERSTACK_ACCESS_KEY=<your-browserstack-access-key>
```

2.3.2 SauceLabs

```
$ export SAUCE_USERNAME=<your-saucelabs-username>
$ export SAUCE_ACCESS_KEY=<your-saucelabs-access-key>
```

2.3.3 CrossBrowserTesting

```
$ export CROSSBROWSERTESTING_USERNAME=<your-crossbrowsertesting-username>
$ export CROSSBROWSERTESTING_ACCESS_KEY=<your-crossbrowsertesting-access-key>
```

CHAPTER 3

How To Test

3.1 Initial Common Steps

3.1.1 Environment Settings

BrowserStack

```
$ export BROWSERSTACK_USERNAME=<your-browserstack-username>
$ export BROWSERSTACK_ACCESS_KEY=<your-browserstack-access-key>
```

SauceLabs

```
$ export SAUCE_USERNAME=<your-saucelabs-username>
$ export SAUCE_ACCESS_KEY=<your-saucelabs-access-key>
```

CrossBrowserTesting

```
$ export CROSSBROWSERTESTING_USERNAME=<your-crossbrowsertesting-username>
$ export BROWSECROSSBROWSERTESTINGRSTACK_ACCESS_KEY=<your-crossbrowsertesting-access-
↪key>
```

3.1.2 Update Supported Browsers (Optional)

See *Platform Configuration* for the significance and details of this step.

Replace `{platform}` in the command below with any of: `browserstack`, `saucelabs`, `crossbrowsertesting`.

```
$ ./node_modules/.bin/cbtr-{platform}-update
```

3.1.3 Specify Your Browsers

See *Browsers YAML* for details on how to specify browsers for your tests.

Here is a list of few sample files you can use, if you want to understand the format later:

- BrowserStack sample: `./node_modules/cross-browser-tests-runner/samples/yml/browserstack.yml`
- SauceLabs sample: `./node_modules/cross-browser-tests-runner/samples/yml/saucelabs.yml`
- CrossBrowserTesting sample: `./node_modules/cross-browser-tests-runner/samples/yml/crossbrowsertesting.yml`

3.1.4 Generate Test Settings

See *Settings* for details on the JSON format test settings `cross-browser-tests-runner` uses.

Generate it using the following command that uses the browsers YAML file as input:

```
$ ./node_modules/.bin/cbtr-init --input <path-to-browsers-yml-file> --output <path-to-
↪settings-file>
```

See *cbtr-init* for usage and how to use defaults for the command line input options.

3.2 JavaScript Unit Testing

First complete the *Initial Common Steps*, as applicable.

3.2.1 Using Native Runner

To run your tests using *Native Runner*, add the following parameters in your test settings file:

- `framework` (See *Parameters*) - the JavaScript unit test framework used in your tests
- `test_file` (See *Parameters*) - the local HTML file that your test would open (See *Test HTML*)

Run the following:

```
$ ./node_modules/.bin/cbtr-server --native-runner --config <path-to-settings-file>
```

This would run all your tests and exit once completed.

See *Server* for details on `cbtr-server` command.

3.2.2 Using Testem

Generate the Testem configuration file `testem.json` for your platform.

Replace {platform} in the command below with one of: `browserstack`, `saucelabs`, `crossbrowsertesting`.

```
$ ./node_modules/.bin/cbtr-testem-{platform}-init --input <path-to-settings-file> --
↪output <path-to-testem-json>
```

It would overwrite `launchers` and `launch_in_ci` settings in an existing testem settings file

See the following for details on platform-specific executable binaries for generating testem settings:

- *cbtr-testem-browserstack-init*
- *cbtr-testem-saucelabs-init*
- *cbtr-testem-crossbrowsertesting-init*

Run the cross-browser-tests-runner server using the following command:

```
$ ./node_modules/.bin/cbtr-server &
```

Now run testem in CI mode as follows:

```
$ testem ci
```

3.3 Selenium Testing

First complete the *Initial Common Steps*, as applicable.

Add the following parameters in the test settings file:

- `test_file` (See *Parameters*) - the local HTML file that your test would open
- `test_script` (See *Parameters*) - a file that contains your Selenium test script (See *Test Script*)

Run the cross-browser-tests-runner server using the following command:

```
$ ./node_modules/.bin/cbtr-server --native-runner --config <path-to-settings-file>
```

This would run all your tests and exit once completed.

CHAPTER 4

Status

4.1 Integrations

Platform	JS - Testem	JS - Native Runner	Selenium
BrowserStack	✓	✓	✓
SauceLabs	✓	✓	✓
CrossBrowserTesting	✓	✓	✓

4.2 JS Frameworks supported by Native Runner

Unit Testing Framework
Jasmine 1.x

4.3 Supported Node.js Versions

Node version	Linux	OS X	Windows
4.8.6	✓	✓	✓
8.9.1	✓	✓	✓

NOTE: v6.9.0 is the minimum version for Selenium tests and core Selenium functionality has been verified on this version. For the sake of time, v6 is not part of CI builds currently, and would be included later.

4.4 Minimum Node.js Versions

Platform	JS - Testem	JS - Native Runner	Selenium
BrowserStack	v4.8.5+ v5.x.x	v4.8.5+	v6.9.0+
SauceLabs	v6.9.0+	v6.9.0+	v6.9.0+
CrossBrowserTesting	v4.8.5+ v5.x.x	v4.8.5+	v6.9.0+

CHAPTER 5

Platform Configuration

Cross-browser testing platforms have very different configurations in browsers/platforms supported, selenium/appium capabilities and various intricacies involved with these values. Another important aspect is that these change over time.

To test with multiple platforms, one needs to understand these details separately for each of these platforms and keep a tab on how these change. Cross-browser-tests-runner aims to make it easy to test across different platforms by abstracting these details beneath a simpler interface, which begins with representing each platform's configuration details through values and rules (as applicable) stored into local config files.

5.1 Local Configuration Files

The `./node_modules/cross-browser-tests-runner/conf` directory contains these:

- Platform-specific browsers/operating systems, capabilities and configuration rules/conditions
  - `browserstack-conf.json`
  - `saucelabs-conf.json`
  - `crossbrowsertesting-conf.json`
- Common/Generic config information:
  - `cbtr-conf.json`: pre-v1.0 this file had significant common details. Post-v1.0 it has very little details and may be removed in later releases.

5.2 Updating Configuration

Replace `{platform}` in the command below with any of: `browserstack`, `saucelabs`, `crossbrowsertesting`.

```
$ ./node_modules/.bin/cbtr-{platform}-update
```

5.3 Standard Names & Conversions

Each cross-browser testing platform uses its own browser and OS names and from case-to-case there are intricate details as follows:

- SauceLabs uses 'Browser' as the browser name for all Android appium tests
- BrowserStack uses iphone or ipad name for Mobile Safari on iphones and ipads respectively

Cross-browser-tests-runner helps you specify your test configuration using standard/uniform browser/operating system names (See *Browsers YAML*). The conversions from standard/uniform names is done internally using the conversions available in Platform Configuration files.

5.4 JS & Selenium Browsers

Some cross-browser testing platforms provide different browsers for JavaScript unit testing and Selenium testing. This is taken into account, and the configuration files store `JS` and `Selenium` browsers of each platform separately.

- `JS` browsers: for writing JavaScript unit tests using testing frameworks
- `Selenium` browsers: for writing Selenium-based tests

5.5 Capabilities & Conditions

Apart from browser, browser version, os, os version and device capabilities, which provide the unique identification of a browser, there are several capabilities e.g. selenium version, appium version, resolution, orientation etc. and each platform has its own intricacies e.g.

- Selenium version for OS X Snow Leopard has to be < 2.47.1 on BrowserStack
- Capturing console logs is supported only for Chrome on BrowserStack
- Different platforms have different screen resolutions available, and different sets of them for different operating systems or devices
- Selenium version has to be < 3.0.0 on SauceLabs if it's not Chrome/Firefox on Windows/OS X (and that's one of the conditions)
- SauceLabs allows specific sets of appium versions with each device
- The Gecko driver (for Firefox) has specific values depending on the Selenium version used on BrowserStack

The Platform Configuration files represent all these capabilities and conditions using a novel JSON-based syntax that represents the conditions using rules. The test configurations written by you are parsed and validated against the information in the files, so that you can avoid errors.

CHAPTER 6

Browsers YAML

A typical cross-browser test involves running your code on multiple browsers. Specifying them in the form of what is referred as "Selenium capabilities" and managing them becomes complex as it requires too much of code.

Cross-browser-tests-runner provides you with a compact and smart format in which you can specify the browsers. This helps you manage the browsers to be used in your tests more efficiently.

6.1 Format

6.1.1 Example

```
CrossBrowserTesting:
  JS:
    Windows:
      "10":
        Chrome x64:
          "45.0-49.0, 54.0":
            resolution: 1920x1200
    Android:
      "6.0":
        Dolphin Mobile:
          "11.5":
            Android Nexus 9:
        Maxthon Mobile:
          "4.3":
            Android Nexus 9:
              resolution: 1536x2048
  Selenium:
    OS X:
      El Capitan:
        Safari:
          "9.0":
            resolution: 2560x1440
    iOS:
      "9.3":
        iPhone 6s Plus Simulator, iPad Pro Simulator:
```

6.1.2 Variations

The example above illustrates 3 variations of the format.

Desktop Browsers

```
{Platform}:
  {TestType}:
    {OS}:
      {Os Version}:
        {Browser}:
          {Browser Versions}:
            {Properties}
```

Mobile Browsers

```
{Platform}:
  {TestType}:
    {OS}:
      {Os Version}:
        {Browser}:
          {Browser Versions}:
            {Devices}:
              {Properties}
```

Single-Version Native Mobile Browsers

There are cases where a mobile device has exactly one browser that either has exactly one version or has a null version. In such cases, the browser details **must** be skipped, and the format looks like the following:

```
{Platform}:
  {TestType}:
    {OS}:
      {Os Version}:
        {Devices}:
          {Properties}
```

6.1.3 Parameters

Parameter	Values
`Platform`	`BrowserStack`, `SauceLabs`, `CrossBrowserTesting`
`Test Type`	`JS`, `Selenium`
`OS`	Any of the OSes specified in platform-specific configuration for the given `Platform` and `Test Type` e.g. node_modules/cross-browser-tests-runner/conf/browserstack-conf.json for `BrowserStack`
`OS Version`	Any of the versions available for the chosen `OS` in platform-specific configuration
`Browser`	Any of the browsers available for the chosen `OS Version` in platform-specific configuration
`Browser Versions`	A comma-separated list of versions from those available for the chosen `Browser` in platform-specific configuration. As can be seen in the example above, a range of versions can be specified e.g. 12.0-19.0.
`Devices`	A comma-separated list of devices from those available for the chosen `Browser` in platform-specific configuration
`Properties`	Capabilities like resolution, orientation etc. from those available for the chosen `Browser`/`Device` in platform-specific configuration. See *Properties* for details.

NOTE: Please use double quotes around numeric values to avoid unwanted errors caused by the YAML parser.

Properties

Property	Values	BrowserStack	SauceLabs	CrossBrowserTesting
`deviceType`	`phone` `tablet`		✓	
`resolution`	`string` type	✓	✓	✓
`orientation`	`portrait` `landscape`	✓	✓	✓
`isPhysicalDevice`	`true` `false`	✓		

6.2 Multiple Copies

You can create this file anywhere in your project, and you can have multiple such files if you have various tests, with each using different sets of browsers.

6.3 Samples

```
./node_modules/cross-browser-tests-runner/samples/yml/*.yml
```

CHAPTER 7

Settings

Here is a summary of 2 entities of cross-browser-tests-runrer created towards the aim of providing a uniform/standard interface across various cross-browser testing platforms:

1. As one can see in *Platform Configuration*, since different cross-browser testing platform use different names for browsers and OSes, cross-browser-tests-runner helps your write your tests using uniform browser and platform name aliases, and would convert them to platform-specific names internally.

2. To improve efficiency in writing your test configuration, you write browsers involved in your tests in a *Browsers YAML* in a smart and compact format, using the uniform aliases from above.

The 'Settings File' is the third link of the above chain. It is a JSON format file that is generated based on the Browsers YAML file (using *cbtr-init*). It eventually contains other details of your tests e.g. unit testing framework, the test file paths etc. It serves the following purposes:

1. Provides test settings for the in-built JavaScript unit tests and Selenium tests runner

2. Serves as the common input file to generate multiple third party unit tests runner settings file e.g. testem.json, using executable binaries provided by cross-browser-tests-runner. Since each third-party unit tests runner has their own configuration syntax, generating each of them from a common settings file helps reducing efforts.

7.1 Format

7.1.1 Examples

JavaScript Unit Testing

```
{
  "framework": "jasmine",
  "test_file": "tests/functional/code/tests/jasmine/html/tests.html",
  "retries": 1,
  "limit": "4mb",
  "browsers": {
    "BrowserStack": {
      "JS": [
        {
          "os": "Windows",
          "osVersion": "7",
          "browser": "Chrome",
          "browserVersion": "32.0"
        },
        {
          "os": "iOS",
          "osVersion": "6.0",
          "browser": "Mobile Safari",
          "browserVersion": null,
          "device": "iPad 3rd (6.0)"
        }       ]
    },
    "SauceLabs": {
      "JS": [
        {
          "os": "Windows",
          "osVersion": "7",
          "browser": "Chrome",
          "browserVersion": "32.0"
        },
        {
          "os": "OS X",
          "osVersion": "Sierra",
          "browser": "Mobile Safari",
          "browserVersion": "10.3"
        }
      ]
    },
    "CrossBrowserTesting": {
      "JS": [
        {
          "os": "Android",
          "osVersion": "7.0",
          "browser": "Dolphin Mobile",
          "browserVersion": "12.0"
        },
        {
          "os": "OS X",
          "osVersion": "Sierra",
          "browser": "Safari",
          "browserVersion": "10.0"
        }
      ]
    }
  },
  "capabilities": {
    "BrowserStack": {
      "local": true,
      "localIdentifier": "native-functional-tests",
      "build": "native-runner-build",
      "test": "native-runner-functional-test",
      "project": "cross-browser-tests-runner/cross-browser-tests-runner",
      "screenshots": true
    },
    "SauceLabs": {
      "local": true,
      "localIdentifier": "native-functional-tests",
      "build": "native-runner-build",
      "test": "native-runner-functional-test",
      "timeout": 90,
      "screenshots": true
    },
    "CrossBrowserTesting": {
      "local": true,
      "localIdentifier": "native-functional-tests",
      "build": "native-runner-build",
      "test": "native-runner-functional-test",
      "project": "cross-browser-tests-runner/cross-browser-tests-runner",
      "screenshots": true,
      "timeout": 90
    }
  },
  "server": {
    "port": 8000,
    "host": "127.0.0.1"
  },
  "parallel": {
    "BrowserStack": 2,
    "SauceLabs": 5,
    "CrossBrowserTesting": 5
  }
}
```

Selenium Testing

```
{
  "test_file": "tests/functional/code/tests/selenium/html/tests.html",
  "test_script": "tests/functional/code/scripts/selenium/script-1.js",
  "browsers": {
    "BrowserStack": {
      "Selenium": [
        {
          "os": "OS X",
          "osVersion": "Mavericks",
          "browser": "Firefox",
          "browserVersion": "39.0"
        },
        {
          "os": "Android",
          "osVersion": "4.0",
          "browser": "Android Browser",
          "browserVersion": null,
          "device": "Motorola Razr"
        }
      ]
    },
    "SauceLabs": {
      "Selenium": [
        {
          "os": "OS X",
          "osVersion": "Mavericks",
          "browser": "Chrome",
          "browserVersion": "33.0"
        },
        {
          "os": "Android",
          "osVersion": "5.0",
          "browser": "Android Browser",
          "browserVersion": null,
          "device": "Android Emulator"
        },
        {
          "os": "iOS",
          "osVersion": "8.3",
          "browser": "Safari",
          "browserVersion": null,
          "device": "iPhone 6 Plus Simulator"
        }
      ]
    },
    "CrossBrowserTesting": {
      "Selenium": [
        {
          "os": "OS X",
          "osVersion": "Yosemite",
          "browser": "Firefox",
          "browserVersion": "39.0"
        },
        {
          "os": "Windows",
          "osVersion": "8.1",
          "browser": "Internet Explorer",
          "browserVersion": "11.0"
        }
      ]
    }
  },
  "capabilities": {
    "BrowserStack": {
      "local": true,
      "localIdentifier": "native-functional-tests",
      "build": "native-runner-build",
      "test": "native-runner-functional-test",
      "project": "cross-browser-tests-runner/cross-browser-tests-runner",
      "screenshots": true,
      "timeout": 120
    },
    "SauceLabs": {
      "local": true,
      "localIdentifier": "native-functional-tests",
      "build": "native-runner-build",
      "test": "native-runner-functional-test",
      "project": "cross-browser-tests-runner/cross-browser-tests-runner",
      "timeout": 120
    },
    "CrossBrowserTesting": {
      "local": true,
      "localIdentifier": "native-functional-tests",
      "build": "native-runner-build",
      "test": "native-runner-functional-test",
      "project": "cross-browser-tests-runner/cross-browser-tests-runner",
      "screenshots": true,
      "timeout": 120
    }
  },
  "server": {
    "port": 7883,
    "host": "127.0.0.1"
  },
  "parallel": {
    "BrowserStack": 2,
    "SauceLabs": 5,
    "CrossBrowserTesting": 5
  }
}
```

7.1.2 Parameters

Parameter	Applies To	Description	Possible Values	Default
`framework`	JavaScript unit tests using the in-built native runner	It is the name of the JavaScript unit tests framework used in your tests	`jasmine`	`jasmine`
`retries`	JavaScript unit tests using the in-built native runner	Number of retries to try a test once it fails (See *Test Retries*)	>=0	0
`limit`	JavaScript unit tests using the in-built native runner	Size limit of data (test reports, and code coverage data) sent by a browser to accept	See Request Size Limit	`"4mb"`
`test_file`	JavaScript unit tests and Selenium tests using the in-built native runner	Path of test HTML file(s), must be relative to root directory of your project (See *Test HTML* for details)	A string or an array of strings - one per test html file	
`test_script`	Selenium tests	Path of Selenium test script(s), must be relative to root directory of your project (See *Test Script* for details)	A string or an array of strings - one per test script file	
`browsers`	All testing	The set of browsers to use for your tests - generated by *cbtr-init* based on browsers specified in your *Browsers YAML* file		
`capabilities`	All testing	Testing "capabilities" other than browser details - generated with defaults by *cbtr-init*	See *capabilities*	See *capabilities*
`server`	All testing	Server host and port - generated with defaults by *cbtr-init*	See *server*	See *server*
`parallel`	All testing	Number of sessions to run in parallel on a cross-browser testing platform - generated with defaults by *cbtr-init*	See *parallel*	See *parallel*

capabilities

Parameter	Platforms	Description
`local`	All	Enforces testing of local pages if set to `t`
`localIdentifier`	All	Identifier for tunnel used for local testing
`screenshots`	All	Enables taking screenshots if set to `true`
`video`	All	Enables capturing a video of test if set to
`timeout`	All	Timeout for a browser/test session in seco
`project`	`BrowserStack`	`username/repo` slug of your project, a
`build`	All	build id for your tests, automatically set if
`test`	All	Name for your test session (applies to all
`tags`	`SauceLabs`	An array of arbitrary tags for a test
`customData`	`SauceLabs`	An object with arbitrary key values
`priority`	`SauceLabs`	To assign higher/lower priority to a test as
`parentTunnel`	`SauceLabs`	While using sub-accounts, use this to use
`noServerFailureScreenshots`	`SauceLabs`	Do not take screenshots of selenium scrip
`videoUploadOnPass`	`SauceLabs`	Upload video even if a test passes
`seleniumVersion`	`BrowserStack` `SauceLabs`	Selenium version to use
`appiumVersion`	`BrowserStack` `SauceLabs`	Appium version to use
`timezone`	`BrowserStack` `SauceLabs`	Time zone to use for a test
`captureConsole`	`BrowserStack`	Capture console logs of a test
`captureNetwork`	`BrowserStack` `CrossBrowserTesting`	Capture network packets of a test
`captureLogs`	`SauceLabs`	Record logs of a test
`captureHtml`	`SauceLabs`	Capture HTML output of a test
`ieNoFlash`	`BrowserStack`	Do not use Flash in Internet Explorer
`ieDriver`	`BrowserStack` `SauceLabs`	Version of Internet Explorer webdriver
`ieCompat`	`BrowserStack`	Compatibility level of Internet Explorer
`iePopups`	`BrowserStack`	Enable pop-ups in Internet Explorer
`edgePopups`	`BrowserStack`	Enable pop-ups in Edge
`safariPopups`	`BrowserStack`	Enable pop-ups in Safari
`safariAllCookies`	`BrowserStack`	Allow all cookies in Safari
`safariDriver`	`BrowserStack`	Version of Safari webdriver
`geckoDriver`	`BrowserStack`	Version of gecko (Firefox) driver
`chromeDriver`	`SauceLabs`	Version of Chrome webdriver
`automationEngine`	`SauceLabs`	Automation engine to use on devices
`autoAcceptAlerts`	`SauceLabs`	Automatically accept JavaScript created a
`prerun`	`SauceLabs`	An object that describes an executable to

server

Parameter	Description	Possible Values	Default
`host`	The IP address on which the test server listens	ip address, host name	`"127.0.0.1"`
`port`	The port on which the test server listens	`number` type	`7982`

No changes in this section are needed unless:

- You need to connect to the server from a separate machine (probably for your local testing) and not using a tunnel. In such case you may want to change it to `"0.0.0.0"`.
- The port `7982` is in use by some other process

parallel

Parameter	Description	Possible Values	Default
`BrowserStack`	Number of sessions that can be run in parallel on BrowserStack	>0	`2`
`SauceLabs`	Number of sessions that can be run in parallel on SauceLabs	>0	`5`
`CrossBrowserTesting`	Number of sessions that can be run in parallel on CrossBrowserTesting	>0	`5`

7.2 Multiple Copies

You can have more than one test settings files, if you need to break your work down into multiple tests.

7.3 Samples

`./node_modules/cross-browser-tests-runner/samples/cbtr/*/*.json`

CHAPTER 8

cbtr-init

This executable binary takes a browsers YAML file as input and outputs a cross-browser-tests-runner test settings file.

8.1 Usage

```
$ ./node_modules/.bin/cbtr-init [--help|-h] [--input|-i <browsers-yaml-file>] [--
↪output|-o <cbtr-settings-file>]
Defaults:
 input              .cbtr-browsers.yml in project root
 output             cbtr.json in project root

Options:
 help               print this help
 input              input data of browsers to use in a compact format
 output             cross-browser-tests-runner settings file
```

8.2 Defaults

As can be seen in the usage section above, defaults can be used for command line parameters.

Parameter	Default value
`-i|--input`	`.cbtr-browsers.yml` file in root directory of your project
`-o|--output`	`cbtr.json` file in root directory of your project

CHAPTER 9

cbtr-testem-browserstack-init

This executable helps you generate `testem.json` - Testem's settings - from cross-browser-tests-runner *Settings* that includes BrowserStack browsers.

NOTE: Please note that browsers from other platforms would not work even if they are included in the settings file.

9.1 Usage

```
$ ./node_modules/.bin/cbtr-testem-browserstack-init [--help|-h] [--input|-i <cbtr-
↪settings-file>] [--output|-o <testem-settings-file>]

Defaults:
 input              cbtr.json in project root
 output             testem.json in project root

Options:
 help               print this help
 input              cross-browser-tests-runner settings file
 output             testem settings file
```

9.2 Questions asked

The utility would ask you the following questions:

- `Are you using multiple tunnels with different identifiers? (y/n) [If unsure, choose "n"]`
  - BrowserStack supports multiple tunnels with different identifiers as well as a single tunnel without any id. If your tests need multiple tunnels, choose 'y', or else 'n'. The tool would generate tunnel IDs in case you chose 'y'.
- `Do you need to take screenshots of your tests once completed? (y/n)`

- `Do you need to take video of your test? (y/n)`
  - Some browsers may not support taking a video, and this behavior is pretty dynamic. So you need to experiment and figure out for yourself.
- `Please provide a timeout value [60]`
  - Minimum timeout on BrowserStack has to be 60 seconds.

CHAPTER 10

cbtr-testem-saucelabs-init

This executable helps you generate `testem.json` - Testem's settings - from cross-browser-tests-runner *Settings* that includes SauceLabs browsers.

NOTE: Please note that browsers from other platforms would not work even if they are included in the settings file.

10.1 Usage

```
$ ./node_modules/.bin/cbtr-testem-saucelabs-init [--help|-h] [--input|-i <cbtr-
↪settings-file>] [--output|-o <testem-settings-file>]

Defaults:
 input              cbtr.json in project root
 output             testem.json in project root

Options:
 help               print this help
 input              cross-browser-tests-runner settings file
 output             testem settings file
```

10.2 Questions asked

On running, the executable would ask you the following questions:

NOTE: You need to enter a value, and there are no defaults

- `Are you using multiple tunnels with different identifiers? (y/n) [If unsure, choose "n"]`
  - SauceLabs supports multiple tunnels with different identifiers as well as a single tunnel without any id. If your tests need multiple tunnels, choose 'y', or else 'n'. The tool would generate one tunnel ID per browser in case you chose 'y'.

- `Do you need to take screenshots of your tests once completed? (y/n)`
- `Do you need to take video of your test? (y/n)`
- `Please provide a timeout value [60]`

CHAPTER 11

cbtr-testem-crossbrowsertesting-init

This executable helps you generate `testem.json` - Testem's settings - from cross-browser-tests-runner *Settings* that includes CrossBrowserTesting browsers.

NOTE: Please note that browsers from other platforms would not work even if they are included in the settings file.

11.1 Usage

```
$ ./node_modules/.bin/cbtr-testem-crossbrowsertesting-init [--help|-h] [--input|-i
↪<cbtr-settings-file>] [--output|-o <testem-settings-file>]

Defaults:
 input              cbtr.json in project root
 output             testem.json in project root

Options:
 help               print this help
 input              cross-browser-tests-runner settings file
 output             testem settings file
```

11.2 Questions asked

On running, the executable would ask you the following questions:

NOTE: You need to enter a value, and there are no defaults.

- `Are you using multiple tunnels with different identifiers? (y/n) [If unsure, choose "n"]`
  - **Please choose 'n'**. CrossBrowserTesting does not support multiple tunnels yet. This question remains for the sake of uniformity and supporting quick-start executable and would be removed later.
- `Do you need to take screenshots of your tests once completed? (y/n)`

- `Do you need to take video of your test? (y/n)`
- `Please provide a timeout value [60]`

CHAPTER 12

Server

Cross-browser-tests-runner uses a server written using `express` that manages testing state and processes.

12.1 Modes

12.1.1 Third Party Test Runners

```
$ ./node_modules/.bin/cbtr-server
```

The server would run and keep waiting. Hooks are provided that bind a third party test runner like Testem with the interfaces provided by the server to create and manage tests. See *Using Testem* for an example of how to use this mode.

12.1.2 Native Runner

Please see *Native Runner* for details on this home-grown tests runner.

```
$ ./node_modules/.bin/cbtr-server --native-runner [--config <path-to-settings-file>]
```

Debugging Mode

If the `--config` option is not provided, or there are no browsers/testing information specified in the settings file mentioned, the server would keep running. A user then can open test HTML files on a local browser. In this mode, you can debug your tests before running the tests on a cross-browser testing platform. For example, if the server is listening on port `7982`, and `tests.html` exists in the root directory of your project, you can open `http://127.0.0.1:7982/tests.html` in a local browser.

CI Mode

The server runs all the tests and exits if the following are provided in the input settings file:

- JS browsers and `test_file` (*Parameters*) parameter, or
- Selenium browsers, `test_file` (*Parameters*), and `test_script` (See *Parameters*) parameters

12.2 Usage

```
$ ./node_modules/.bin/cbtr-server [--help|-h] [--config|-c <config-file>] [--native-
↪runner|-n] [--errors-only|-e] [--omit-traces|-o] [--error-reports-only|-E] [--omit-
↪report-traces|-O]

Defaults:
 config               cbtr.json in project root, or CBTR_SETTINGS env var
 native-runner        false
 errors-only          false
 omit-traces          false
 error-reports-only   false
 omit-report-traces   false

Options:
 help                 print this help
 config               cross-browser-tests-runner settings file
 native-runner        if the server should work as native test runner
 errors-only          (native runner only) print only the specs that failed
 omit-traces          (native runner only) print only the error message and no stack␣
↪traces
 error-reports-only   (native runner only) report only the error specs from browser
 omit-report-traces   (native runner only) do not include stack traces in reports sent␣
↪by browser
```

12.2.1 Important Options

NOTE: Please note that all of the following apply to native runner mode only

- `errors-only`: use this if you are interested in looking at only the failed test specs/suites
- `omit-traces`: use this if you are interested in the failure messages only and not the stack trace
- `error-reports-only`: use this if you want to send data of only the failed test specs/suites from the browser
- `omit-report-traces`: use this if you want to not send stack traces of failures from the browser

First two are aimed at removing clutter in the output. Next two are aimed at reducing the size of test results data sent by browser.

CHAPTER 13

Native Runner

13.1 Need for a tool like cross-browser-tests-runner

The beginning idea for cross-browser-tests-runner was to help make JavaScript testing using cross-browser testing platforms easy. It was seen that several JavaScript test runners like Testem exist; however, they do not naturally extend to cross-browser testing platforms. And, the work to integrate different test runners with different cross-browser testing platforms is very fragmented, and that creates a steep learning curve if experimenting/working with multiple platforms and tests runners is required.

So how if there was one tool that enables you to pick any of your favorite test runners and test on any of your favorite cross-browser testing platforms? That's what cross-browser-tests-runner aims to be.

13.2 Limitations of existing work

However; as work on integrating existing test runners was taken up, several limitations were found:

- Testem's instrumented JavaScript does not work on older browsers, starting with assumptions of JSON object being available in the browser (not true with older browsers). Experiments showed that a risky and major rewrite of the injected code would be required.
- BrowserStack's own runner called browserstack-runner does work on older browsers; but does not have a proxy/bypass mechanism like Testem, so it's not possible to send client-side code coverage data to the test server and store it.
- For local testing, each cross-browser testing platform uses tunnels. It was seen with BrowserStack that tunnel processes die often, which means that the test results would not reach to the test server even if a client browser sends it, so completion of the test cannot be detected. There is no tool that works around such issues specific to cross-browser testing platforms.

13.3 What a cross-browser tester needs

Essentially, a serious and purist cross-browser tester needs a solution that:

- takes care of the complete testing workflow taking the state machine and undocumented/unknown/stability issues of a cross-browser testing platforms into account
- does not cause limitations on browser coverage
- supports essential testing features like code coverage, and
- includes fail-over mechanisms to account for errors

It was seen that modifying existing third-party solutions to include all of above can be difficult. This is where the in-built test runner - called Native Runner - was born.

13.4 Features

13.4.1 Large Browser Coverage

The code injected by Native Runner works on oldest browsers like Internet Explorer 6, and Android 1.5.

13.4.2 Code Coverage Collection

If your JavaScript source is instrumented using a tool like Istanbul, the coverage data collected on the browser is automatically sent to Native Runner, which stores it into `coverage/` directory in your project's root.

You can use Istanbul or any other compatible tools to generate code coverage reports, and upload the coverage data to services like https://codecov.io, https://coveralls.io and others.

13.4.3 JavaScript Unit and Selenium Testing

With versions 0.4.0+, Native Runner supports both. For Selenium testing, it abstracts various details for you so you can focus on writing test code alone.

13.4.4 Monitoring of Tunnels

BrowserStack tunnels die due to undocumented/unknown issues. Native Runner monitors tunnels created across all platforms and restarts those that die.

With 0.5.0+ this functionality has been moved to the Platform library layer, so this happens even for testing with third party test runners like Testem.

13.4.5 Test Results Reporting Retries

Native Runner's injected code includes a retry with exponential back-off mechanism that tries to recover from failures that occur when tunnels die, exploiting the tunnel monitoring mechanism described above.

Since this is done by code injected by Native Runner, this is not available with third party test runners.

13.4.6 Test Retries

Sometimes, test result reporting fails even with all above mechanisms in place. Native Runner retries tests for which no results were reported if a non-zero `retries` (See *Parameters*) parameter is provided in *Settings*.

13.5 JavaScript Unit Testing

13.5.1 Test HTML

Unlike Testem that generates the test HTML, one needs to write a test HTML file when using Native Runner. Following sections provide samples that show the structure the test HTML file needs to have.

A test can use more than test HTML files. The `test_file` (See *Parameters*) parameter in *Settings* specified test HTML file(s) to use.

Jasmine 1.x

Use the following sample and replace the annotated lines in the sample with your source and test JavaScript files.

```
<!doctype html>
<html>
<head>
  <title>Cross Browser Tests Runner</title>
  <script src="//cdnjs.cloudflare.com/ajax/libs/jasmine/1.3.1/jasmine.js"></script>
  <script src="//cdnjs.cloudflare.com/ajax/libs/jasmine/1.3.1/jasmine-html.js"></
↪script>
  <script src="/cross-browser-tests-runner.js"></script>
  <script>
    (function() {
      var jasmineEnv = jasmine.getEnv();
      jasmineEnv.addReporter(new jasmine.HtmlReporter);
      window.onload = function() {
        jasmineEnv.execute();
      }
    })()
  </script>
  <!-- start of your app and test code -->
  <script src="../../js/src/app.js"></script>
  <script src="../../js/tests/jasmine/test.js"></script>
  <!-- end of your app and test code -->
  <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/jasmine/1.3.1/jasmine.
↪css">
</head>
<body>
  <div id="jasmine_content"></div>
</body>
</html>
```

13.6 Selenium Testing

13.6.1 Test Script

For Selenium Testing, one needs to write the test code in a file and provide its path in `test_script` (See *Parameters*) parameter in *Settings*.

Sample

The following sample shows the structure of a test script file.

```
'use strict'

exports.script = (driver, webdriver) => {
  return driver.findElement({id: 'test-message'})
  .then(el => {
    return el.getText()
  })
  .then(text => {
    console.log('Selenium Test Script: text of #test-message %s', text)
    return true
  })
}

exports.decider = (driver, webdriver) => {
  return Promise.resolve(true)
}
```

Structure

A test script exports two functions:

- `script` **required**: This implements the test script functionality.
- `decider` *optional*: This decides whether the test succeeded or failed.

Arguments provided to both functions:

- `driver`: This is the `thenable` web driver instance created by Builder. See Selenium documentation.
- `webdriver`: This is the handle obtained with `javascript require('selenium-webdriver')`

Both functions must return a `Promise` or `thenable`. See the Selenium documentation.

CHAPTER 14

Troubleshooting

Add `LOG_LEVEL=DEBUG` to any of the utilities/commands on Linux/OSX, or export `LOG_LEVEL` as an environment variable on Windows before running them. For example:

```
$ LOG_LEVEL=DEBUG ./node_modules/.bin/cbtr-init
```

Supported logging levels: `DEBUG`, `INFO`, `WARN`, and `ERROR`, with `DEBUG` producing most verbose logging.

Default logging level: `ERROR`