Mozilla automated testing ✓

---

this is a text anotation

You've just written a feature and (hopefully!) want to test it. Or you've decided that an existing feature doesn't have enough tests and want to contribute some. But where do you start? You've looked around and found references to things like "xpcshell" or "mozmill" or "talos". What do they all do? What's the overlap? In short, where should your new tests go?

This document aims to answer that question. There's a very short summary of each framework, and a bit of Q&A to help you pick your framework. This may only narrow down your choices, however, in which case you should read more about the frameworks and/or hop on #ateam, #qa, or one of the development forums and ask questions.

Generally, you should pick the lowest-level framework that you can. If you are testing JavaScript but don't need a window, use xpcshell. If you're testing page layout, try to use reftest. The advantage is that you don't drag in a lot of other components that might have their own problems, so you can home in quickly on any bugs in what you are specifically testing.

In production ✓

buildbot automation

These tests are found within the mozilla-central tree, along with the product code. They are all run when a changeset is pushed to mozilla-central, mozilla-inbound, or try, with the results showing up on tbpl ey can also be run on their own.

The letters in parentheses are the abbreviations used by tbpl.

compiled-code (B)

Written in C++, compiled-code tests can test pretty much anything but are difficult to write properly. In general, this should be your last option for a new test, unless you have to test something that is not exposed to JavaScript.

xpcshell (X)

xpcshell are console JavaScript tests. There is no chrome, no content, no window. xpcshell is useful for testing low-level things, such as XPCOM components. If you don't need a window, use this. xpcshell is particularly useful for testing low-level objects that *are* exposed to JavaScript.

JS shell tests (J)

Tests specifically for the JavaScript engine. They test every piece of the engine.

crashtest (C)

Really simple: open a web page and see if it causes a crash. If you've found pages that crash Firefox, add a test here to make sure future versions don't experience this crash again.

reftest (R)

A reftest verifies that two web pages are rendered identically to test layout and graphics correctness, taking advantage of the fact that there is generally more than one way to achieve any given visual effect in a browser. For each test, Reftest will take two sample pages that try to produce the same effect (normally one with a simple markup, and one using more complex markup) and verify that they produce the same visual construct.

Mochitest (M)

Mochitest uses JavaScript to test features. Anything piece that has its functionality exposed in JavaScript can theoretically be tested with Mochitest. "Plain" mochitest should be used to test DOM APIs and other pieces of functionality exposed to web content (i.e. requiring no special permissions).

Mochitest-other (Moth)

Mochitest-other are mochitests with higher privileges, logically split into a few sections:

- IPC: tests plugin APIs, particularly out-of-process plugins.
- a11y: tests accessibility interfaces.
- chrome: tests running with high privileges that can test a lot of the browser's functionality. Tests that verify JavaScript interactions with chrome-level objects should go here. These tests do not exist for mobile Firefox.
- browser-chrome: running in the scope of the browser window, this is a rough UI automation tool testing how the browser interacts with itself and with content. Since these are moving away from the rest of mochitest's functionality, they will eventually be split into their own category, "b-c".

Mochitest-Robocop (Mrc)

Mochitest-Robocop tests run on Native Android builds only marked with an 'rc' in tbpl. These are Java based tests which run from the mochitest harness and generated similar log files. These are designed for testing the native UI of Android devices by sending events to the front end.

Talos (T)

Talos is a framework for performance testing. If you're measuring performance, Talos is the place to go.

Talos tests on buildbot are split into a few categories. Some test suites are run in several categories but with different configurations or metrics. The following are the codes as found on tbpl:

- tp: Measures the load time of a set of test web pages taken from the Alexa top 500.
- s: SVG rendering performance.

Desktop only

- c: chrome. A set of suites with chrome (i.e. the full UI) enabled.
- di: dirty. Uses a "dirty" places.sqlite that more closely resembles that of an average user.
- dr: Dromaeo. A suite of JavaScript performance tests.
- n: nochrome. A set of suites with chrome disabled.

Mobile only

- dh: tdhtml. Measures the time to cycle through a set of DHTML test pages.
- pn: tpan. Loads a test page and measures the time to pan to the bottom then back up to the top.
- sp: tsspider. Runs the SunSpider benchmark test.
- tpn: The tp suites with chrome disabled.
- ts: Tests the startup time of Firefox by opening the browser 20 times.
- w: twinopen. Time to open a new window.
- z: tzoom. Loads a test page and measures the time to zoom in and out.

These are Robocop based tests that are developed and running in either staging/production but have no official names on tbpl:

- (tc): tcheckerboard. Loads a test page, zooms and pans, measures the amount of checkerboarding (delayed painting)
- (tc2): tcheckerboard2: Similar to tcheckerboard, but supports pinch to zoom
- (rp): trobopan: Loads a test page and pans to the bottom and back to the top. This measures the lag time in rendering the page.
- (pr): tprovider: Fills the awesomebar database (android os db) with thousands of entries and measures the time to perform a series of queries.

Mozmill

Mozmill is an extensive framework for browser automation. It is an extension with an extensive API to help you write functional tests that simulate user interactions, as well as a full unit test API. It can be used to test configurations that are difficult to simulate in buildbot automation. QA automation uses

Mozmill to test localized builds, performance over time, and other scenarios.

Speedtests

SpeedTests is a framework for executing arbitrary tests, which report results via JavaScript calls, in several browsers. Originally this framework was designed for modified versions of Microsoft's speed demos but has now been expanded to include conformance tests such as test262 and Kraken.

SpeedTests are good for cross-browser comparisons when tests don't need to dig too deep into the guts of the browsers.

Peptest

Peptest measures responsiveness, how "snappy" Firefox/Thunderbird feels, by issuing alerts when the event loop is stuck for more than 50 ms. It will soon be available on try, to catch regressions in responsiveness. If you're helping to improve peppiness, consider writing some peptests to help you measure your improvements and to find future regressions.

Marionette harness

Marionette is a test automation framework used to drive the UI and JS/XPCOM layer of remote or local instances.

The Marionette client includes the harness which runs Marionette and mochitest-browser-chrome tests. It will give you similar functionality as Selenium for Firefox builds, but with built in chrome support as well. With it, you can send commands to chrome or content on demand, allowing you to coordinate large scope tests like communicating to multiple remote gecko processes (useful for testing things like SMS messaging for WebAPI for example).

This is currently being used to test B2G, but can work with any gecko platform.

So which do I use already?

Here's a series of questions to ask when you want to write some tests. Remember this is only a rough guide, and it may give you multiple frameworks. Try #ateam on irc.mozilla.org to get some more specific answers.

Is it low-level code?

If the functionality is exposed to JavaScript, consider xpcshell. If not, you'll probably have to use compiled-code tests.

Does it cause a crash?

If so, a crashtest could help isolate the problem. Note that this may lead to more tests once the core problem is found.

Is it a layout/graphics feature?

Reftest is your best bet, if possible.

Do you need to verify performance?

Try Talos!

Are you comparing speed or functionality between browsers?

See if you can write a Speedtest.

Want to investigate responsiveness?

Peptest provides an easy way to measure responsiveness.

Testing UI?

If it's mobile UI, look into Robocop. For desktop, try browser chrome tests, soon to be split out of Mochitest, or Mozmill

Testing Mobile/Android?

Mobile UI, look at Robocop. There are some specific features that Mochitest or Reftest can cover. Mochitest-chrome and browser-chrome do not run on Android. If you want to test performance, Talos runs just fine with a few limitations (use --noChrome options) and smaller cycles (e.g. 10 iterations instead of 20, etc...)

None of the above?

To get your tests run through buildbot, try Mochitest, or, if higher privileges are required and you don't need mobile testing, try Mochitest chrome tests.

While not in buildbot automation, Mozmill is a bigger framework that can test almost anything, on the desktop at least.

For desktop Firefox or B2G, or if you just want to see the future of Gecko testing, look into the on-going Marionette project.