FirefoxOS Certification Testsuite Documentation

Release 1.0

Mozilla Corporation and Individual Contributors

June 25, 2014

CONTENTS

Tests and tools to verify the functionality and characteristics of Firefox OS on real devices.

Contents:

CHAPTER

ONE

REQUIREMENTS

The test suite is designed to run on unprivileged devices and does not rely on high level instrumentation. Instead human interaction is needed throughout the test suite to perform various instructions in order to assert that conditions are met on the device.

The certification test suite is intended to run on a host computer attached to the device via USB cable. Currently the host requires the Linux or Mac OS operating systems with *adb* (Android Debug Bridge) installed.

If you need to install adb, see https://developer.mozilla.org/en-US/Firefox_OS/Debugging/Installing_ADB.

Once installed, adb must be on your PATH. If you use the bash shell emulator you can modify your *~/.bashrc* or equivalent file, by adding the following line to the file (replacing $SDK_HOME with the location of the Android SDK:

```
export PATH=$SDK_HOME:$PATH
```

The device must have a SIM card with a functioning phone subscription to receive SMS messages for a subset of the tests to pass.

1.1 Network

The device must be connected to WiFi and must have network access to the host machine.

On the host machine, the following ports are required, and must not have any existing servers running on them:

- 2828
- 8000
- 8001
- 8888

Any network firewall must be configured to allow the device to access the host computer on the above ports.

Additionally, if you run the test suite on Mac you need to disable the system firewall so that the servers used as part of the test suite can listen to the ports mentioned above. To do this, head to the *Security & Privacy* pane in *System Preferences* and click the *Turn Off Firewall* button if present.

CHAPTER

TWO

SETUP AND USAGE

There are two methods for setting up and running the test suite: The “quick” method and the “virtualenv” way. For either to work you must enable adb access to the device.

2.1 Enabling ADB

For Firefox OS version 1.3: Launch *Settings*, and navigate to *Device Information → More Information → Developer*, then check *Remote Debugging*.

For version 1.4: Launch *Settings*, and navigate to *Device Information → More Information*, then check *Developer Options*. Next, hold down the *Home* button, and close the *Settings* app (press the *X*). Finally, launch *Settings* again, and navigate to *Developer*, then select *ADB only* in *Remote Debugging*.

Once this is done, go to *Settings → Display* and set the *Screen Timeout* to “never”. You need this because adb will not work when the device is locked.

2.2 Quick Setup and Usage

You can setup your environment and run the tests by running:

```
./run.sh --version=<some version>
```

The *–version* argument is optional. If passed, *–version* must be one of our supported release versions, either 1.3 or 1.4. If you don’t pass a version, 1.3 will be assumed.

This command sets up a virtual environment for you, with all the proper packages installed, activates the environment, runs the tests, and lastly deactivates the environment.

You may call *run.sh* as many times as you like, and it will run the tests using its previously set up virtual environment.

Some of the tests for Web APIs require manual user intervention. At this point a browser will open on your host computer. Simply follow the instructions given on screen.

2.3 Setup and Usage With virtualenv

If the quick setup doesn’t work, then follow these instructions. You can set up and run this tool inside a virtual environment. From the root directory of your source checkout, run:

```
virtualenv .
./bin/pip install -e .
```

Then activate the virtualenv:

```
source bin/activate
```

Once the virtualenv is activated, the certification test suite can be run by executing:

```
runcertsuite
```

To get a list of command-line arguments, use:

```
runcertsuite --help
```

For example it is possible to list logical test groups and to run filtered test runs of only a subset of the tests:

```
runcertsuite --list-test-groups
```

2.4 Submitting Results

Once the tests have completed successfully, they will write a file containing the results to disk; by default this file is called *firefox-os-certification.zip* and will be put in your current working directory. Please e-mail this file to fxos-cert@mozilla.com.

2.5 Known Issues

- None

CHAPTER

THREE

TESTS

3.1 WebAPI Tests

These tests can be run without user interaction.

<insert documentation for tests>

3.2 Semi-automated WebAPI tests

These tests require some user interaction to run.

CHAPTER

FOUR

INTERPRETING RESULTS

After running the FxOS Certification Suite, a result file will be generated (firefox-os-certification.zip by default) in the current directory. Inside this file are several logs; you need to review two of these to understand the cert suite's results.

4.1 The results.html file

This file contains the results of the web-platform-tests. To see these results, click the 'web-platform-tests' link. You will see a list of all the tests run, and their status. Any test failures with have a status which begins with 'UNEXPECTED'.

4.2 The cert/results.json file

4.2.1 omni_results

This section contains the output of the omni_analyzer tool. The omni_alayzer compares all the JS files in omni.ja on the device against a reference version. If any differences are found, the entire file containing the differences is base-64 encoded and included in the result file.

To see the diffs between the files on the device and the reference versions, use the omni_diff.py tool inside the certsuite package. To run this tool:

```
    source certsuite_venv/bin/activate # this will exist after you run the tests cd certsuite python omni_diff.py
    /path/to/cert_results.json expected_omni_results/omni.ja.1.3 results.diff
```

You can then view results.diff in an editor.

Differences in omni.ja files are not failures; they are simply changes that should be reviewed in order to verify that they are harmless, from a branding perspective.

4.2.2 application_ini

This section contains the details inside the application.ini on the device. This section is informative.

4.2.3 headers

This section contains all of the HTTP headers, including the user-agent string, that the device transmits when requesting network resources. This section is informative.

4.2.4 buildprops

This section contains the full list of Android build properties that the device reports. This section is informative.

4.2.5 kernel_version

This section contains the kernel version that the device reports. This section in informative.

4.2.6 processes_running

This section contains a list of all the processes that were running on the device at the time the test was performed. This section is informative.

4.2.7 [unpriv|priv|cert]_unexpected_webidl_results

This section, if present, represents differences in how interfaces defined in WebIDL files in a reference version differ from the interfaces found on the device in an (unprivileged|privileged|certified) context. For example:

> **{** “message”: “assert_true: The prototype object must have a property “textTracks” expected true got false”, “name”: “HTMLMediaElement interface: attribute textTracks”, “result”: “FAIL”
>
> },

This means that the HTMLMediaElement interface was expected to expose a textTracks attribute, but that attribute was not found on the device.

4.2.8 [unpriv|priv|cert]_added_window_functions

This section, if present, lists objects descended from the top-level ‘window’ object which are present on a reference version, but not present on the device, in an (unprivileged|privileged|certified) context.

4.2.9 [unpriv|priv|cert]_missing_window_functions

This section, if present, lists objects descended from the top-level ‘window’ object which are present on the device, but not on a reference version, in an (unprivileged|privileged|certified) context.

4.2.10 [unpriv|priv|cert]_added_navigator_functions

This section, if present, lists objects descended from the top-level ‘navigator’ object which are present on a reference version, but not present on the device, in an (unprivileged|privileged|certified) context.

4.2.11 [unpriv|priv|cert]_missing_navigator_functions

This section, if present, lists objects descended from the top-level ‘navigator’ object which are present on the device, but not on a reference version, in an (unprivileged|privileged|certified) context.

4.2.12 [unpriv|priv|cert]_added_navigator_unprivileged_functions

This section, if present, lists objects descended from the top-level ‘navigator’ object which are reported as null on a reference version, but reported as not-null on the device. This could indicate a permissions problem; i.e., the object belongs to an API which a reference version reports as null because the API is only available to privileged contexts, and the test is run in an unprivileged context, but which is available in an unprivileged context on the device. This test is performed in an (unprivileged|privileged|certified) context.

4.2.13 [unpriv|priv|cert]_missing_navigator_unprivileged_functions

This section, if present, lists objects descended from the top-level ‘navigator’ object which are reported as not-null on a reference version, but reported as null on the device. This could indicate a permissions problem; i.e., the object belongs to an API which should be available to unprivileged contexts, but which is not available to an unprivileged context on the device. This test is performed in an (unprivileged|privileged|certified) context.

CHAPTER

FIVE

TEST COVERAGE

5.1 Web Platform Tests

Tests from the W3C's web-platform-tests testsuite covering standardised, web-exposed, platform features. These tests work by loading HTML documents in a simple test application and using javascript to determine the test result. Tests are divided into groups — i.e. directories — according to the specification that they are testing. The upstream testsuite is undergoing continual development and it is not expected that we pass all tests; instead correctness for the purposes of the certsuite is determined by comparison to a reference run. At present the following tests groups are enabled:

dom Tests for the dom core specification

IndexedDB Tests for the IndexedDB specification

touch_events Simple tests for the automatically verifiable parts of the touch events specification

5.2 Guided WebAPI Tests

WebAPIs make it possible to interface between the web platform and device compatibility APIs. To test these APIs we need to assert certain physical aspects about the phone during the testrun.

E.g. to verify that the device does indeed change its screen orientation when tilted 90º, we will first ask the user to turn the device and then ask her for the perceived orientation, which is then compared with what the API reports.

For this reason the guided Web API tests, backed by the test harness *semiauto*, require a user to interact with various questions and prompts raised by the tests. This works by showing a dialogue with a question, confirmation, or input request in a web browser on the user's host computer (the computer the device is connected to).

To ensure that all facets of the various WebAPIs are covered the tests also require a number of phyiscal aids to be present when the tests are running. These involve the presence of a Wi-Fi network, a bluetooth enabled second device, a phone with SMS and MMS capabilities, &c.

As with the web platform tests, the tests are organized in logical groups divided by directories (listed below) that make up Python modules. Some of the tests may not be applicable depending on the device under test's capabilities and hardware configuration.

bluetooth Bluetooth API provides low-level access to the device's Bluetooth hardware.

fm_radio Provides support for a device's FM radio functionality.

geolocation Provides information about the device's physical location.

mobile_message Lets apps send and receive SMS text messages, as well as to access and manage the messages stored on the device.

mozpower Lets apps turn on and off the screen, CPU, device power, and so forth. Also provides support for listening for and inspecting resource lock events.

moztime Provides support for setting the current time.

notification Lets applications send notifications displayed at the system level.

orientation Provides notifications when the device's orientation changes.

proximity Lets you detect proximity of the device to a nearby object, such as the user's face.

tcp_socket Provides low-level sockets and SSL support.

telephony Lets apps place and answer phone calls and use the built-in telephony user interface.

vibration Lets apps control the device's vibration hardware for things such as haptic feedback in games.

wifi A privileged API which provides information about signal strength, the name of the current network, available WiFi networks, and so forth.

5.3 WebAPI Verifier

The WebAPI Verifier test group attempts to detect changes to the supported WebAPIs. Test apps are generated with all permissions for each app type (hosted, privileged, certified) and the tests are repeated for each condition.

Coverage is provided in two ways. The first is a simple recursive walk of the window (and so also the navigator) object which enumerates properties of each object encountered. This is compared to an expected results list, which will detect added, removed and modified properties. It is not capable of detecting behavioural or semantic modifications (for example, changes to the arguments for a method.)

The W3C WebIDL test suite [1] is used to provide additional test coverage. This suite generates tests to verify WebAPI implementations based upon the WebIDL files used to define them. It goes further than the simple recursive enumeration of properties described above. For example, given a method on an interface, it creates tests to verify the type of the method is 'function', checks that the length of the operation matches the minimum number of arguments specified in the IDL file, and verifies that function will throw a TypeError if called with fewer arguments.

The the WebIDL test suite is itself is still under development and so has bugs and does not provide complete coverage. It was originally designed to work on a desktop browser and will run out of memory on some devices. To work around these problems, a preprocessing step is performed using the in-tree WebIDL.py parser, which also limits testing to a subset of the interfaces defined in the full set of WebIDL files.

This avoids out-of-memory situations on the device as well as running tests which are guaranteed to fail. For example, the version of the test suite in use currently expects every interface defined to be accessible from the window object, which is not the case for interfaces like 'AbstractWorker'. These interfaces are made available to the test suite when testing other interfaces, but are not directly tested themselves.

The WebIDL test suite should be sufficient by itself to verify the WebAPIs have not been modified, but since it is not complete, the recursive walk of the window object is also performed to provide additional coverage.

[1] https://github.com/w3c/testharness.js/

5.4 Permissions

Permissions model testing is currently done by generating apps of each app type (hosted, privileged, certified) with either all or no permissions granted.

Each permission is then tested individually. The majority of these tests examine the window or navigator object for the existence (or non-nullness) of a property, and so are redundant with the WebAPI verification performed above. Some permissions require additional work. For example, the mozbrowser permission requires creating an iframe and then testing whether or not certain properties are available on it.

Not all permissions are not currently tested due to a variety of reasons: * background-sensors (planned feature) * background-service (planned feature) * deprecated-hwvideo (removed) * networkstats-manage (only used in Gaia) * storage (attempts to test this result in OOM) * audio-capture (triggers known bug on some devices) * video-capture (triggers known bug on some devices) * network-events (requires phone to be on data network, but the testharness requires wifi) * wappush (requires source of wappush events)

Based upon feedback on the initial set of tests, we are in the process of moving to a test where the list of permissions to test is created by examining the PermissionsTable.jsm file on the device, and the permissions are tested individually. This will allow the detection of added or removed permissions (although the omni.ja tests will also provide coverage here) as well as detecting whether setting one permission allows more than it should (e.g. if setting systemXHR to 'allow' also granted access to Contacts.)

In this case, one app will be created for each app type with no permissions granted. The permissions will then be read from the permissions table and each one will be toggled to 'allow' individually. The test app will then recursively walk the window object (as done in the WebAPI verification tests) and report the results.

This does not provide coverage for the permissions that require special handling such as the mozbrowser permission. These will be tested using individual test cases as is currently done by using a separate app, and these tests will have to be maintained across different versions of B2G.

5.5 Omni Analyzer

Many of Gecko's JavaScript sources are compressed into an omni.ja file which is part of all FirefoxOS distributions. The omni-analyzer extracts these files and compares them to a relevant reference version. Any differences are logged, and the diffs between test and reference files can be viewed using the omni_diff.py tool.

The omni-analyzer does not produce pass/fail results; differences in JavaScript source files should be reviewed by an engineer to determine whether they're harmless in terms of FirefoxOS branding requirements.

5.6 User-Agent Test

The user-agent test verifies that the user agent string reported by the device conforms to the Gecko user agent specification and the device model inclusion requirements.

CHAPTER SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*