FirefoxOS Certification Testsuite Documentation

Release 1.0

Mozilla Corporation and Individual Contributors

January 14, 2016

Contents

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

Contents:

CHAPTER 1

Requirements

The test suite is designed to run on devices with adb root access 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 and phone calls for a subset of the tests to pass.

The device must have an SD card inserted with some free space available 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

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 2

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 and Disabling Screen Lock

Launch *Settings*, and navigate to *Device information* → *More Information*, then check *Developer Menu*. 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
```

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-tests
```

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

- Bug 1026259 - web-platform tests sometimes lose wifi connection

CHAPTER 3

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 4

Interpreting results

After running the FxOS Certification Suite, a result file will be generated (firefox-os-certification_timestamp.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 all PASS/FAIL tests run by the cert suite, including the webapi tests, the web-platform-tests, and the webIDL tests.

4.2 The cert/cert_results.html file

This file contains informative test output that needs to be interpreted by a human engineer. It contains the following sections:

4.2.1 omni_result

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, they are displayed here.

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 [web|privileged|certified]_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 [web|privileged|certified]_added_webidl_results

This section, if present, represents new, unexpected APIs which are exposed to applications in an (unprivileged|privileged|certified) context on the test device, but which are not present on a reference device.

4.2.9 [web|privileged|certified]_missing_webidl_results

This section, if present, represents APIs which are missing in an (unprivileged|privileged|certified) context on the test device, but which are present on a reference device.

4.2.10 [web|privileged|certified]_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.

4.2.11 [web|privileged|certified]_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.

CHAPTER 5

Test Coverage

5.1 Security

Security testing checking the process authority and ssl file checking

filesystem check the file system permission

ssl check the CA validatation

kernel check the authority of kernel processes

5.2 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

5.3 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.

apps able to retrieve manifest of apps.

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

device_storage Accessibility of read/write device storage such as SD card.

devicelight Lets you detect changes to ambient light using the device's light sensor.

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

geolocation Provides information about the device's physical location.

idle idle mode for power saving

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.

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.

5.4 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/

5.5 Permissions

The permissions model is tested by first retrieving the PermissionsTable from the device. Then for each app type (hosted, privileged, certified) an app is generated and installed on the device. For each permission in the PermissionsTable, the value is queried in the app and compared to an expected result. This comparison will detect added or removed permissions, as well as any changes to the default permission value (deny, prompt, allow) for each app type.

Once this is complete, a second round of testing determines the effect of setting each permission to 'allow'. A baseline result is generated by recursively walking the window object with no permissions set. Then each permission is individually set to 'allow' and then window object is again walked and the result compared to the baseline. This will detect any changes to the window or navigator object that result from the permission being set.

This is not sufficient to detect the effect of all permissions. For instance, to detect whether the mozbrowser permission is granted, it is necessary to create an iframe with mozbrower and then look for additional properties on it. A small number of hand written tests are run to accomodate these cases, and again compared to expected results.

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)

5.6 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.

5.7 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 6

Making a new release

This page describes how to create a new release of the certification suite.

6.1 github branching

If you are creating a release for the first time for a new version of Firefox OS, you should create a new branch in github for it, e.g.,:

```
git branch v2.1
git checkout v2.1
git push origin v2.1
```

The master branch should be retained as a development branch; releases should not be made directly from it.

6.2 versioning

You should verify that *certsuite/config.json* has the correct two-digit version number this release supports. This version number is passed to the individual components of the suite as they're run.

The package version in *setup.py* should be bumped with each release. For a release tracking Firefox OS 2.1, for example, the valid package versions are 2.1.0, 2.1.1, etc.

6.3 archive generation

The contents of the *fxos-certsuite* directory are copied to another directory with a name in the format *fxos-MCTS-2.1.0*, and the hidden *.git* directory is removed. The *documentation.pdf* file in the root directory is updated using command:

```
make latexpdf
```

in the *docs* folder. The resulting PDF file is moved from *_build/latex/FirefoxOSCertificationTestsuite.pdf* to *documentation.pdf* in the root folder, and the *docs/_build* folder is removed.

Finally, the directory is archived using the command:

```
zip -r fxos-MCTS-2.1.0.zip fxos-MCTS-2.1.0/
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search