



MobiWebApp

Mobile Web Applications for Future Internet Services

Deliverable D3.2

Test Suites Report Year 2

Version: 18th September Final

PROJECT PERIODIC REPORT

Name, title and organisation of the scientific representative of the project's coordinator¹:

Dr Philipp Hoschka Tel: +33-4-92385077 Fax: +33-4-92385011 E-mail: ph@w3.org

Project website² address: <http://mobiwebapp.eu/>

Project	
Grant Agreement number	257800
Project acronym:	MobiWebApp
Project title:	Mobile Web Applications for Future Internet Services
Funding Scheme:	Coordination & Support Action
Date of latest version of Annex I against which the assessment will be made:	March 17, 2010
Document	
Period covered:	1 September 2011 to 31 August 2012
Deliverable number:	D3.2
Deliverable title	Test Suites Report Year 2
Contractual Date of Delivery:	Project month M24
Actual Date of Delivery:	September 2012
Editor (s):	Robin Berjon
Author (s):	Robin Berjon
Reviewer (s):	Dominique Hazael-Massieux
Participant(s):	ERCIM/W3C
Work package no.:	3
Work package title:	Interoperability
Work package leader:	Robin Berjon
Distribution:	PU
Version/Revision:	1.1
Draft/Final:	Final
Total number of pages (including cover):	20

¹ Usually the contact person of the coordinator as specified in Art. 8.1. of the grant agreement

² The home page of the website should contain the generic European flag and the FP7 logo which are available in electronic format at the Europa website (logo of the European flag:

http://europa.eu/abc/symbols/emblem/index_en.htm ; logo of the 7th

FP: http://ec.europa.eu/research/fp7/index_en.cfm?pg=logos). The area of activity of the project should also be mentioned.

DISCLAIMER

This document contains description of the MobiWebApp project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the MobiWebApp consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 27 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (<http://europa.eu.int/>)



MobiWebApp is a project funded in part by the European Union.

TABLE OF CONTENTS

Disclaimer	3
Table of Contents	4
1- Summary	5
2- Introduction	6
3- Testing Framework	8
4- Testing Activity At W3C	15

1- SUMMARY

This report provides an overview of the achievements of the Interoperability Work Package during the second year of MobiWebApp.

MobiWebApp built atop the work carried out in the first year to continue establishing the constitutive element of a generic testing activity within W3C, targeting mobile interoperability as a primary goal. This activity encompasses groups working specifically on testing technology and is coordinated with the many W3C groups that develop tests as part of their own deliverables.

Over this period, MobiWebApp has developed and deployed a complete testing framework using state-of-the-art mobile-friendly Web technology that enables the management of test suites across the entire W3C organisation in coordination with the broader community and industry, and has contributed to the development of test suites as well as to their integration into the framework through continuous education and outreach. This report delves into both aspects.

2- INTRODUCTION

Web technologies are increasingly being used to develop full-fledged applications, notably on mobile devices where they are commonly perceived as the upcoming primary application development technology. The scope and variety of these technologies keep expanding, adding new features to HTML, to CSS, and a host of new APIs specifically tailored to the creation of applications.

Such an expansion, spread across a broad community spanning multiple working groups, and being deployed by a multiplicity of browser vendors, device manufacturers, and Web application developers naturally creates a number of interoperability issues. While W3C working groups are required to produce test material as part of their deliverables, these efforts have long happened primarily independently from one another, without reuse of testing components, with insufficient common infrastructure, and often with varying degrees of quality. Results obtained from these diverse testing efforts were difficult to find and to combine together in order to gain visibility on the broader interoperability of the platform and therefore on the overall usability of the technologies that compose it.

The W3C thus made the decision to launch a generic testing activity that would be able to produce common infrastructure for the complete testing ecosystem involved in standards development and the usage of standards technologies by Web developers.

MobiWebApp's Interoperability Work Package funds work to create a common test framework within W3C along with its documentation, and work on integrating the test suites that working groups produce into this shared infrastructure.

The Testing Activity at W3C includes the Web Testing Interest Group³ which is in charge of developing and deploying testing infrastructure and their paraphernalia. It is in collaboration with this group that the work described here has been carried out.

³ <http://www.w3.org/testing/ig/>

⁸ <http://www.w3.org/wiki/Testing/Requirements>

3- TESTING FRAMEWORK

During the first year of this project, the MobiWebApp project contributed to the creation in W3C of the basic building blocks of a testing framework. This included deploying a specific test server and putting together the initial version of the testing framework, which is the core component in this project and what the MobiWebApp team kept building atop of. In order to develop this, an initial set of over 70 requirements⁸ was gathered.

Of those requirements, all save one have now been implemented in the online Test Framework¹⁰. (The one missing requirement is currently under scrutiny and is likely to be dropped due to actual lack of interest for it.)



test case name	Gecko	Presto	WebKit
et-childElement-null.html	1/1/1	././1	././1
et-childElement-null.svg	1/1/1	1/./.	1/./.
et-childElement-null.xhtml	1/1/.	1/./.	1/./.
et-childElementCount-nochild.html	2/./.	././1	././1
et-childElementCount-nochild.svg	2/./.	1/./.	1/./.
et-childElementCount-nochild.xhtml	2/./.	1/./.	1/./.
et-childElementCount.html	2/./.	././1	././1
et-childElementCount.svg	2/./.	1/./.	./1/.
et-childElementCount.xhtml	2/./.	1/./.	1/./.
et-dynamic-add.html	2/./.	././1	././1
et-dynamic-add.svg	2/./.	1/./.	1/./.
et-dynamic-add.xhtml	2/./.	1/./.	1/./.
et-dynamic-remove.html	1/./1	././1	././1
et-dynamic-remove.svg	2/./.	1/./.	1/./.
et-dynamic-remove.xhtml	2/./.	1/./.	1/./.
et-entity.svg	3/./.	1/./.	./1/.
et-entity.xhtml	2/./.	1/./.	1/./.
et-firstElementChild.html	1/./1	././1	././1
et-firstElementChild.svg	2/./.	1/./.	1/./.
et-firstElementChild.xhtml	2/./.	1/./.	1/./.
et-lastElementChild.html	1/./1	././1	././1
et-lastElementChild.svg	3/./.	1/./.	1/./.
et-lastElementChild.xhtml	2/./.	1/./.	1/./.
et-namespace.html	1/./1	././1	././1

Figure 1: W3C Test Framework listing test results

¹⁰ <http://w3c-test.org/framework/app/>

The screenshot shows the 'Specifications' tab of the W3C Test Framework. At the top, there are navigation links for 'Test Suites', 'Specifications', and 'Reports', along with a 'Logout' button. Below this is a table with three columns: 'Actions', 'Name', and 'URL'. The table lists six specifications, each with 'edit' and 'sections' buttons in the 'Actions' column. Below the table is a green 'New Specification' button. At the bottom, there is a text box containing information about the project, including a link to the Bugzilla, a note about source availability, developer credits, the MobiWebApp logo, and funding information from the European Union.


Actions	Name	URL
edit sections	HTML5 Whatever	http://www.w3.org/TR/html5/Overview.html
edit sections	Vibrator API	http://www.w3.org/TR/vibration/
edit sections	Drawing Dahut High Circles	http://www.w3.org/TR/battery-status/
edit sections	The Real Vibration API	http://www.w3.org/TR/vibration/
edit sections	Element Traversal	http://www.w3.org/TR/ElementTraversal/Overview.html
edit sections	DeviceOrientation Event Specification	http://www.w3.org/TR/orientation-event/

[New Specification](#)

Use the [W3C Test Framework Bugzilla](#) for comments, questions, and error reports about this system.

The sources for this system are [available online](#). Patches welcome!

This project was developed by [Dominique Hazael-Massieux <dom@w3.org>](mailto:dom@w3.org), [David M. Berfanger <david.berfanger@hp.com>](mailto:david.berfanger@hp.com), [Peter Linss <peter.linss@hp.com>](mailto:peter.linss@hp.com), and [Robin Berjon <robin@berjon.com>](mailto:robin@berjon.com).

 **mobiwebapp**

This project is funded by the European Union through the Seventh Framework Programme (FP7/2010-2012) under grant agreement n°257800 - [Mobile](#)

Figure 2: Specifications tested in the W3C Test Framework

The Test Framework is now fully operational, and is increasingly being used by the Web testing communities. The initial stab at this framework which was carried out during the first year constituted a good starting point, but as a more traditional server-side Web application, it suffered from a number of limitations that did not put it on par with expectations for Web applications developed today. Over the past year, MobiWebApp therefore largely re-architected it in order to build a more flexible, more powerful system that is both more open to future development and easier to manipulate for its end-users.

A number of new features have been added. Of particular note, data from WURFL has been integrated into the framework so as to improve its ability to produce more useful analysis as part of its reporting features. WURFL is a database that makes it possible to identify browsers and devices based on the headers they set in the HTTP protocol.

Prior to WURFL integration, reports could be generated for specific browsers or rendering engines, but such information did not provide the ability to differentiate for instance mobile platforms from desktop ones (especially since in most cases the same browsers or rendering engines are now available on both). With the integration of WURFL, it is now possible to query test results gathered by the system to know if they were on a mobile device, a tablet, or a TV for instance. This enables **more powerful data mining** by making it possible to categorise test results to take the specifics of a **multi-device world** into account.

This contributes to another new feature: the production of report-generating tools as part of the Framework. It is difficult to anticipate what different users may want as part of a report generated by the Framework and therefore to produce all the potentially useful reports.

As detailed below, the Framework now exposes a REST API that is very useful to ensure that third-party developers can both reuse our open data and contribute to it. But it is not realistic to expect everyone who might need such data to be able to write a program to process it and generate a report tailored to specific needs.

In order to address this, the framework now offers a “Reports” section. One of these reports is in fact a **report-making tool**. The user is invited to choose which target platforms she wants a report for (e.g. a specific browser, or browsers on a specific device class such as “all TVs”) and the specifications for which results are desired. Once the report type is specified, the system generates a matrix of support based on test results that fills out the report.

Additionally, the framework can now produce a report on itself, such as how many test suites, test cases, results, and so forth are available. Since the data is continuously available, no human intervention is required in order to keep such reports up to date.

Mobile Friendly Framework

One of the objectives for this second year was to build a mobile-friendly test runner and improve overall navigation in the Test Framework, as well as improve the Test Framework’s user interface overall to make it more fluid to use.

It was already possible to use the Test Framework Web application on mobile devices, but the experience was at best suboptimal. This has now been strongly improved.

First, the code has been reorganised in order to be more easily optimised, which has enabled us to trim down the weight of resources that are being transmitted over the network – an important consideration for mobile devices.

Overall the client-side code has been written to take today’s Web development best practices into account, which translates to a highly maintainable Web application that adapts well to new constraints and requirements.

Second, the way in which the interface is presented has been completely overhauled to be responsive to device size. Notably, when on a smaller screen the navigation switches from horizontal to vertical, the tables that make it possible to select a suite or specification to test fold down into simpler boxes, and the test running user interface reorganises itself to be more usable within the allotted space (see Figure 3).

Only the results tables remain large on a small screen, as there does not seem to be a better way of presenting a large data set of results, and that most users interested in these data would in any case be more likely to use a wide screen to browse through it

Thanks to this the Test Framework is now **perfectly usable on a mobile device**, which is a key aspect in obtaining test results for small-screen devices.

Beyond the mobile improvements, the new user interface has also been largely improved in many places. It is now more fluid and easier to navigate where the first version often confounded users. It also provides more details about the underlying data that the system gathers.

Switching between pages is now much faster as only the requisite data is loaded, while ease of navigation is maintained. This was achieved by reusing modern Web development techniques and libraries. The feedback from end-users received after these changes has been overwhelmingly positive.

Finally, the initial iteration on the Test Framework only made it possible to run tests inside of an <object> element, which worked well for desktop browsers but caused a variety of issues with mobile ones. This has been changed so that embedded tests can now run either using the <object> element or the <iframe> element. This small preference can make a large difference in the behaviour of embedding on some implementations, and therefore helps with the test running experience. The preference has been made sticky for a given user agent on a given device, and can therefore persist across multiple test runs and test suites which makes the experience of users **running tests much more fluid**.

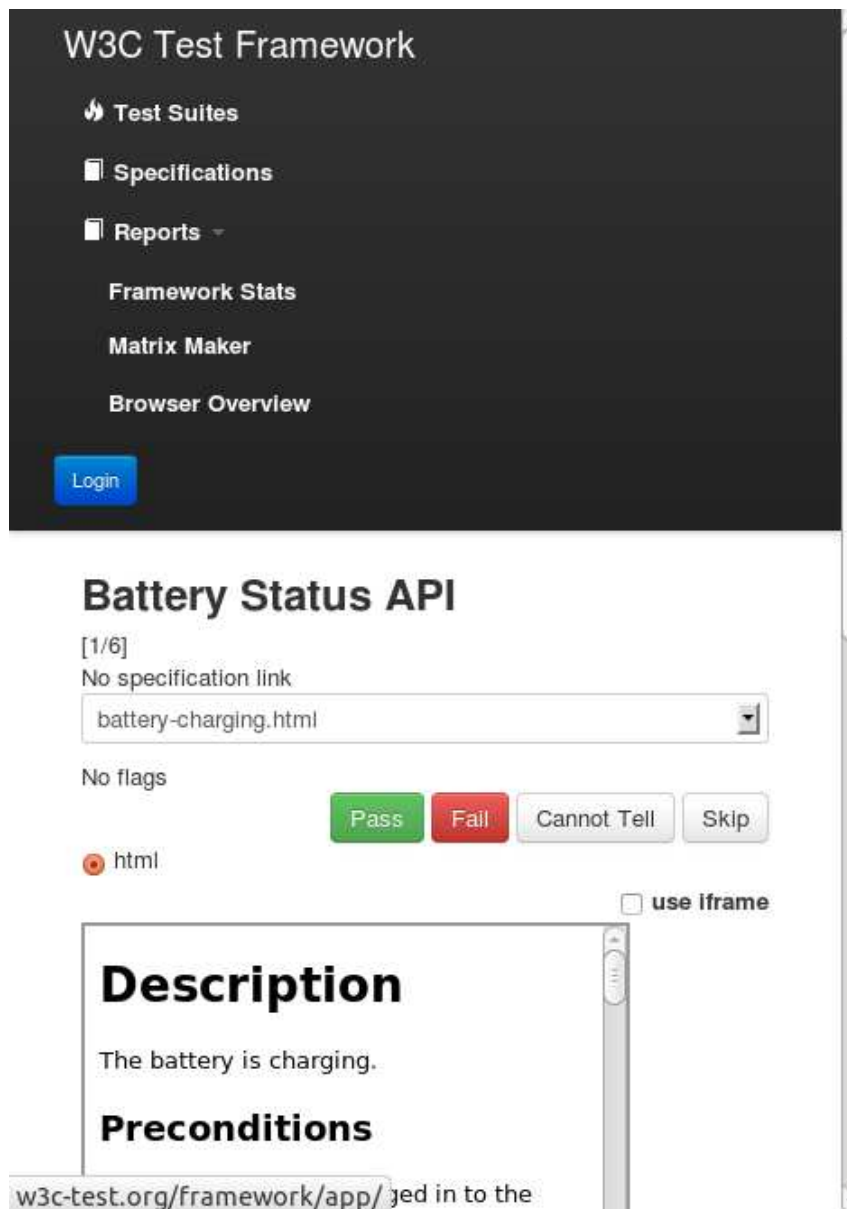


Figure 3: Screenshot of the test runner on a smaller screen

Automated Test Results Collection

By default, the test runner present tests to the tester in simple alphabetical order — which is fine, but not always the most helpful.

In order to address this, the MobiWebApp project added to the system a way of running tests in “most-needed” order. Using this, the framework will find out which **tests need more runs** for the specific user-agent that the current user is running and place those at the beginning of the list.

MobiWebApp has also worked on adding **two other smarter test running modes**. The first is the ability to run tests in a random order. Indeed, in some cases running a test may cause the user agent to enter a state that has a side-effect on another, subsequent test. Due to this, always running tests in the same order can hide problems that should ideally be surfaced. Adding a random order addresses this issue.

The other smarter ordering is the ability to run **automated tests first**. This makes it Ivention, and therefore gather more information faster.

While developing this functionality it appeared that some of the metadata, notably the parts that can help ascertain that a test can run in an automated fashion, were of unequal quality. In order to address that, we **reviewed and improved metadata** throughout the system.

Having now shown the usefulness of that metadata, we now expect that test contributors will pay more attention to it and provide higher quality metadata in the long run.

Documentation for the Framework

While MobiWebApp made efforts to ensure that using the framework is as intuitive as possible, some parts naturally require documentation. The MobiWebApp project therefore provided documentation that describes the manifest format that the framework uses so that it can more easily be manipulated outside of the Manifest Generator tool¹¹.

Another important component is the metadata format as used inside test files themselves. This part of the metadata system makes it possible to maintain metadata as close as possible to the tests that are being described — which is to say in the tests themselves. The MobiWebApp project also wrote and published documentation for that part of the format¹².

This documentation covers multiple aspects. First, it covers the format itself, what it means, and provides a convenient template for whoever wishes to start writing a test from scratch. Second, it provides the pointers to the Test Harness documentation (of which more is said below). Finally, it documents the Manifest Generator tool that can take a test suite, extract all the metadata contained in its tests, and generate a manifest tailored to the needs of the Test Framework so that tests can be more easily imported, with the correct information.

Several users have taken to using the framework since we have documented it, and the feedback so far is that they have found it very helpful in getting up and running quickly with the system.

¹¹ <http://w3c-test.org/framework/docs/maintainer/>

¹² <http://w3c-test.org/framework/docs/maintainer/metadata-format.html>

REST API and Third-Party Integration

Interactions with the test framework were initially only available through a human-oriented user interface.

Over the past year, MobiWebApp also developed a **REST API** to the same functionalities that enables programmatic access (both for reading and writing) to the system, essentially enabling it as an Open Data platform.

This enables third parties to re-use creatively the data collected by the system: anyone in the Web community can make use of this information and interact with it without prior permission. This makes it possible for such third parties to mine the same information and enrich it with new views and new services.

Many creative uses of this data are now entirely open, from building existing services such as Can I Use¹³ atop real-world, continuously gathered information to novel exploitations of this data that we haven't envisaged yet. The flexibility of the API is there precisely so that anyone can put it to work for purposes never imagined by the MobiWebApp team.

Producing specialised views on test results is useful for W3C working groups as well. Amongst other things, it makes it possible for them to more easily evaluate how well a specification is being implemented by vendors, which sections are proving most complicated, and in turn improve their specifications to address such problems.

It can also be used by developers to produce a technological baseline so that they can know when to use some technologies and when they are not yet sufficiently well deployed. A company wishing to develop a Web application known to require a specific set of features could build a view of the results for these features that would enable them to evaluate its support in deployed systems, and to adjust their product to reach a broader audience.

The API caters to many needs by having at its core a simple, REST-friendly set of actions that produce well-documented JSON data. It is accessible to any Web applications (notably via the usage of CORS and JSON-P).

In order to ensure that the API was sufficiently functional for advanced uses, the entire Test Framework user interface itself, including complete reports and test runner, are built on top of this API.

MobiWebApp also assisted the first deployment of the framework's REST API to a third party. The W3C Internationalization Working Group is very keen on testing and on publishing their test results to their own site, with explanations and details about what exactly they are testing. To date they had been carrying out this process manually, a task that required a fair amount of tedious work and that had proven error prone over the years.

By using the documentation and some code examples and with continuous oversight and assistance from the project they were able to automate this functionality and integrate

¹³ <http://caniuse.com/>

data directly from the framework straight into their report pages, notably for Ruby Markup¹⁴ and HTML Escapes¹⁵.

Simple ruby

	Gecko	Presto	Trident	WebKit	
basic rendering, with rb [Exploratory test] When simple ruby markup is used with the rb tag, the ruby text appears above the base text at approximately half the font size.	1 / 1 / .	. / 1 / .	2 / . / .	4 / . / .	ruby-001a
basic rendering, no rb When simple ruby markup is used without the rb tag, the ruby text appears above the base text at approximately half the font size.	. / 1 / .	. / 1 / .	2 / . / .	3 / . / .	ruby-002
no rb, multiple bases in one ruby When multiple ruby bases appear within a single ruby element, without the rb tag, the ruby text appears above the appropriate base text at approximately half the font size.	. / 1 / .	. / 1 / .	2 / . / .	3 / . / .	ruby-003
rp, with rb [Exploratory test] Ruby parentheses in the source are hidden if the browser supports ruby.	. / 1 / 1 / .	. / 1 / 1 / .	2 / . / .	3 / . / .	ruby-004a
rp, no rb Ruby parentheses in the source are hidden if the browser supports ruby.			2 / . / .	3 / . / .	ruby-005

Links: Section 4.6.20 • Latest results for section 4.6.20 • Submit data for section 4.6.20 • Related tests

2011-11-12: Only Internet Explorer, Safari and Chrome all handle simple ruby. They handle it equally well whether there is an `rb` tag or not. They also handle multiple base+annotations in a single `ruby` element.

Figure 4: Integration of Test Framework results in the W3C Internationalization Working Group pages

Working closely with the Internationalization Working Group helped MobiWebApp identify which parts of the REST API worked, and which parts were problematic. Notably, we identified and fixed two primary issues:

- The documentation was insufficiently clear in a number of places about which type of information a client script could expect to receive. This was addressed with extensive clarification of the documentation, based on the explanations that were given to the Internationalization Working Group.
- In some cases, the part of the API that provided test results could generate very large amounts of data with extremely precise granularity. While there are cases in which such granularity can indeed be useful, for many situations (such as the Internationalization group’s case) the volume of results to be transferred was excessive while at the same time causing the rendering of the results to be slow — possibly even unusable over a mobile connection. This was addressed by adding a more summarised API method that provides the same data but with far less granularity and with a number of statistical aspects pre-calculated.

Once the changes were deployed and the Internationalization Working Group was able to release its live pages using MobiWebApp’s REST API, they indicated that they were very happy with the result and plan on using it more. We anticipate to progressively see the appearance of an increasing number of such uses — notably, we expect the API to increasingly be used for the **production of implementation reports**, a critical step in W3C standardization process.

We also envision its usage directly inside draft to flag some sections as more or less stable based on the test results being gathered.

¹⁴ <http://www.w3.org/International/tests/html-css/ruby/results-interactive>

¹⁵ <http://www.w3.org/International/tests/html-css/escapes/results-html-escapes>

4- TESTING ACTIVITY AT W3C

In the first year, the MobiWebApp project created the formal environment in which the work done in the second year evolved. This comprised multiple components. First, requirements were gathered from the W3C community at large in order to ensure that the different needs of all groups developing different technologies were taken into account. Then, these requirements were prioritised in order to produce the development roadmap that was deployed in the second year.

In parallel to that, the collaboration spaces in which this work took place — namely the Web Testing Interest Group and the Browser Testing and Tools Working Group were chartered and outreach was conducted in order to drive participation in them. Having successfully lifted them off the ground, this provided a high quality setting for the continuation of this work to take place in.

Collaboration with Core Mobile Web Platform Community Group

A Community Group is a specific type of group within W3C that is more open than regular groups in order to foster greater involvement from a wider community.

The Core Mobile Web Platform Community Group (CoreMob CG)¹⁶ is one such group that aims to accelerate the adoption of the Mobile Web as a compelling platform for the development of modern mobile web applications. In order to do so, it has gathered together over 250 people from Web development, network operators, large Web companies, handset manufacturers and browser vendors. Its focus on interoperability through testing makes it a natural partner for this project.

Facebook, one of the instigator of the group, submitted their RingMark test suite¹⁷, a visualization of gaps in standards support, to the CoreMob CG. In order to avoid fragmentation in the W3C's testing toolset, we discussed the architecture of the system with the CG in order to reach consensus on a way in which the RingMark visual component could be used to represent results gathered by the W3C Test Framework so that the adherence to standards of various devices and browsers could be easily represented, and conversely to have the RingMark test runner submit new result batches to the Test Framework.

While at this point only the architecture for this integration has been sketched out, development continues in the CoreMob CG to further this idea. We expect this collaboration to continue after the end of the in order to bring this collaboration to full fruition.

Interaction with this CG also made it possible to test drive the functionalities of the framework with highly motivated users from the industry and broader community, which in turn made it possible to make multiple improvements and extensive documentation of the W3C's system.

¹⁶ <http://www.w3.org/community/coremob/>

¹⁷ <http://rng.io/>

Harness Documentation

One aspect in which our collaboration with CoreMob revealed a weakness was in the documentation of the JavaScript test harness.

The Test Harness¹⁸ is a JavaScript library that W3C test suites use in order to run tests in a specific, well-controlled manner, and to produce well-defined reports on a given test run.

It is designed to work well with the overall Test Framework, and to support many advanced features that put it on par with modern testing systems. Thanks to continuous outreach, it is now the standard test system used across W3C groups that produce APIs, such as HTML, WebApps, Device API, etc.

Despite being in common use, it was not well documented beyond comments in the source JavaScript. This provided a barrier to new contributors and did not help existing one produce tests with as high quality as they ought to have. In fact, only a core set of high-quality developers were successful in deploying it properly, while too many remained puzzled as to how to use it or created test suites that features bugs due to poor understanding of the harness. This issue has now been fixed with a **complete tutorial** that covers the entire functionality of the harness system¹⁹.

The tutorial is divided into two columns — one with the functionality, and the other with the code that it corresponds to — in order to be easier to read. Furthermore, all of the code displayed in the tutorial is actually run as part of the tutorial, so that the results as reported by a real test suite can be observed at the end of the tutorial. This is a pattern that is common in tutorials found in the JavaScript community, and was therefore selected as a way of maximising outreach to that specific community.

¹⁸ <https://github.com/jgraham/testharness.js>

¹⁹ <http://darobin.github.com/test-harness-tutorial/docs/using-testharness.html>

`assert_true(actual, description)` checks that `actual` is *strictly* equal to `true`, which is to say that it *has* to be the JavaScript `true` value and not just something that evaluates as "truthy" such as `1` or `"dahut"`.

`assert_false(actual, description)` is the same as `assert_true` but in reverse. It has the same strictness about its `actual` being JavaScript's `false` and not just "falsy" (e.g. `0`, `null`)

`assert_equals(actual, expected, description)` checks that `actual` and `expected` have the same value (without necessarily being the same object). Note that this comparison is strict and that you should not rely on whatever automatic type conversions that JavaScript may perform on comparisons.

`assert_not_equals(actual, expected, description)` is the reverse of `assert_equals` and checks that its `/actual/` and `/expected/` are not the same. The same caveat on comparison strictness applies, so that values that may seem very similar are still not equal.

`assert_in_array(actual, expected, description)` checks that `actual` is in the array provided in `expected`. Any odd member will do, but note that it will not recurse into the array if it is multidimensional.

```
test(function () {
  assert_true(true, "Truth is true");
  assert_true(1 === 1, "One is really one");
}, "Simple checks on truth");

test(function () {
  assert_false(false, "Falsity is false");
  assert_false(1 === 0, "One is not zero");
}, "Simple checks on falsity");

test(function () {
  assert_equals("dahut", "da" + "hut", "String concatenation");
  assert_equals(42, 6 * 7, "The ultimate answer");
}, "Simple checks on equality");

test(function () {
  assert_not_equals("dahut", "myth", "String comparison");
  assert_not_equals(42, "42", "The ultimate answer");
}, "Simple checks on inequality");

test(function () {
  assert_in_array("dahut",
    "chupacabra dahut unicorn".split(" "),
    "Dahut hunting");
  assert_in_array(2017, [42, 47, 62, 2017], "Lottery");
}, "Simple checks on membership");
```

Figure 5: Screenshot of the Test Harness tutorial with running code

The community has provided extensive feedback on this tutorial, which has been incorporated. The Test Harness project now uses MobiWebApp's tutorial as its official primary documentation. Feedback on this tutorial from the test developers involved in the Core Mobile Web Platform Community Group has been very positive, and is helping them develop more tests.

Integrating Existing Test Suites

MobiWebApp has also been contributing to the development of test suites, notably by ensuring that they are integrated into the framework, and in providing the tooling related to the task.

While the test framework was first deployed in late 2011, it now features **3122** test cases in **40** test suites for **37** specifications. Put together they have generated **19,923** test results. The increase in test suites on a monthly basis can be seen in the framework's own reporting system:

Test Suite Additions

Increase in test suites per month.

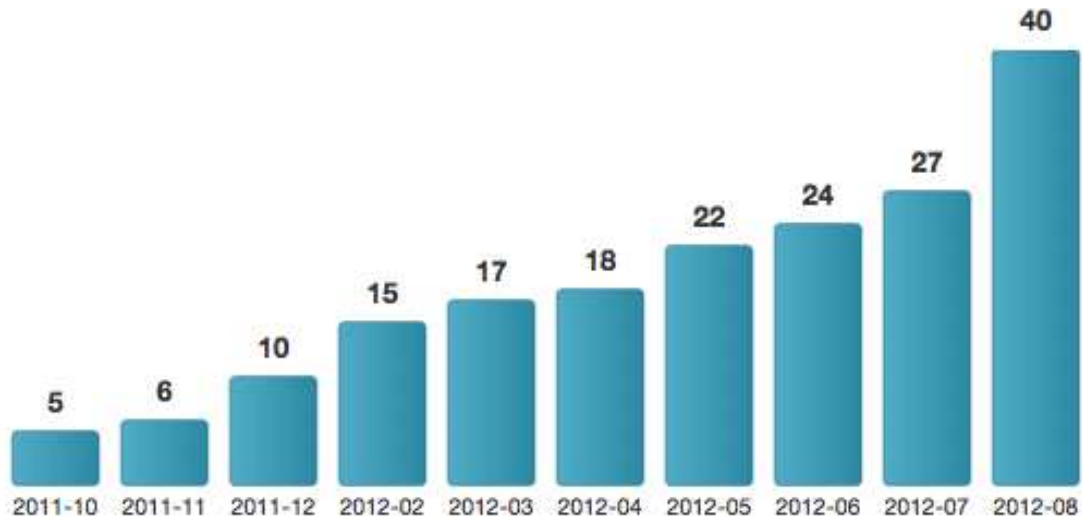


Figure 6: Evolution of the number of test cases in W3C Test Framework

These test suites are naturally generating an increasing volume of test results, which is progressively helping to paint a detailed picture of how well Web standards are implemented and deployed.

Result submissions can also be seen to increase on a monthly basis from the framework's reporting:

Results Submissions

Increase in gathered results per month.

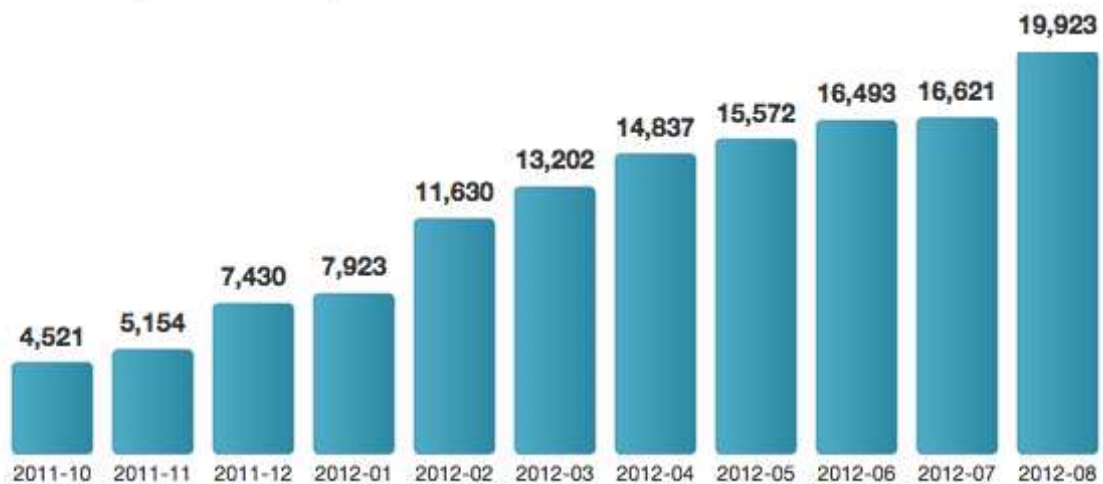


Figure 7: Evolution of number of test results gathered by the W3C Test Framework

Initially, integrating test suites into the framework was somewhat tedious. The reason for that was that one needed to produce a manifest listing all the tests in a test suite, along with metadata describing them, before the test suite could be imported.

However, since test documents contain the required metadata in a machine-readable form, MobiWebApp wrote a small tool called the Manifest Generator that is able to spider an existing test suite, extract the metadata, and generate a manifest that works for the test framework. This has proved very helpful in speeding up the integration of test suites, and most suites are now imported using this tool.

Conclusion

The availability of a strong and full-featured testing ecosystem continues to show its increasing relevance. The production of a generic testing framework and its paraphernalia has helped simplify the production of higher quality test material from Working Groups. While the test framework will continue to evolve, it has now reaches a level of capability and stability that has enabled it to prove itself useful in real-world usage contexts, outside of the community that holds testing as its primary focus.

During its second year, MobiWebApp grew the framework and its surrounding material:

- By expanding the functionality of the framework.
- By making the data available and reusable in machine-readable form.
- By ensuring that the user interface functions properly on constrained devices.
- By documenting all the important components of the framework and collaborating with various groups to ensure that they were able to use our tools.
- By helping with test development and by integrating existing test suites into the system.