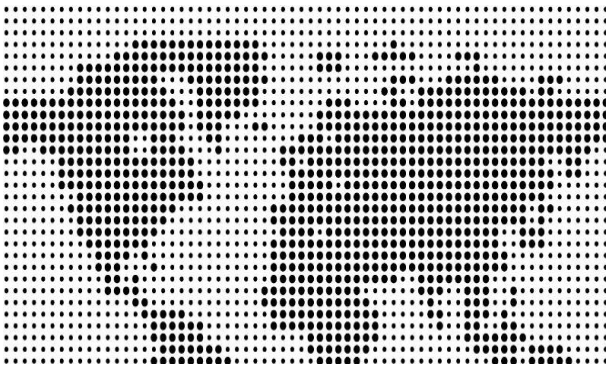
THE TRULY MOBILE WEB

Part 3: Developing Richer Web-based Applications



Qualcomm Incorporated

Qualcomm Innovation Center, Inc.

May 2011

Table of Contents

Summary

Did you know that browsers are not one-size-fits-all?

Did you know that it's possible – in fact, encouraged – to modify them for better performance on specific chipsets?

Did you know that you can extend JavaScript™ – the programming language that runs Web pages and applications from inside every browser – with functionality that reaches from the Web, over the air, through the browser and into device hardware?

The key to developing richer Web-based applications lies in getting three components to work together more smoothly:

- the browser
- JavaScript
- the underlying platform/chipset

As described in other papers in this series, Qualcomm has invested heavily in the mobile-Web-readiness of its Snapdragon™ mobile processor, while Qualcomm Innovation Center, Inc. (QuIC) has invested heavily since 2009 in developing optimizations to the browser and JavaScript. While those optimizations enhance Web page performance and users' ability to interact with web services, the next step will see the browser, JavaScript and the mobile processor combine for Web applications that perform on par with native applications.

This paper explains the vocabulary and main concepts of Web applications (especially vis-à-vis native applications) in the terms that wireless carriers, device manufacturers and application developers need to know. Readers will find insights into how both the industry and QuIC are setting the stage for the user experience that the Truly Mobile Web demands.

MAIN MESSAGE:

THE NEXT STEP WILL SEE THE BROWSER, JAVASCRIPT AND THE MOBILE PROCESSOR COMBINE FOR WEB APPLICATIONS THAT PERFORM ON PAR WITH NATIVE APPLICATIONS.

The Browser and the Mobile Web - Fundamentals

The browser remains the container for the Web.

The browser is the enabling application on the device that leads to the wide variety of rich sites and resources on the Web. It is the device's window onto the Web. But like any window, it still separates the device's internal hardware from the Web, as depicted in Figure 1. Advanced functionality – video conferencing, multimedia, peer-

to-peer sharing – requires a plug-in, which separates the device from the Web even further and hampers both performance and security.

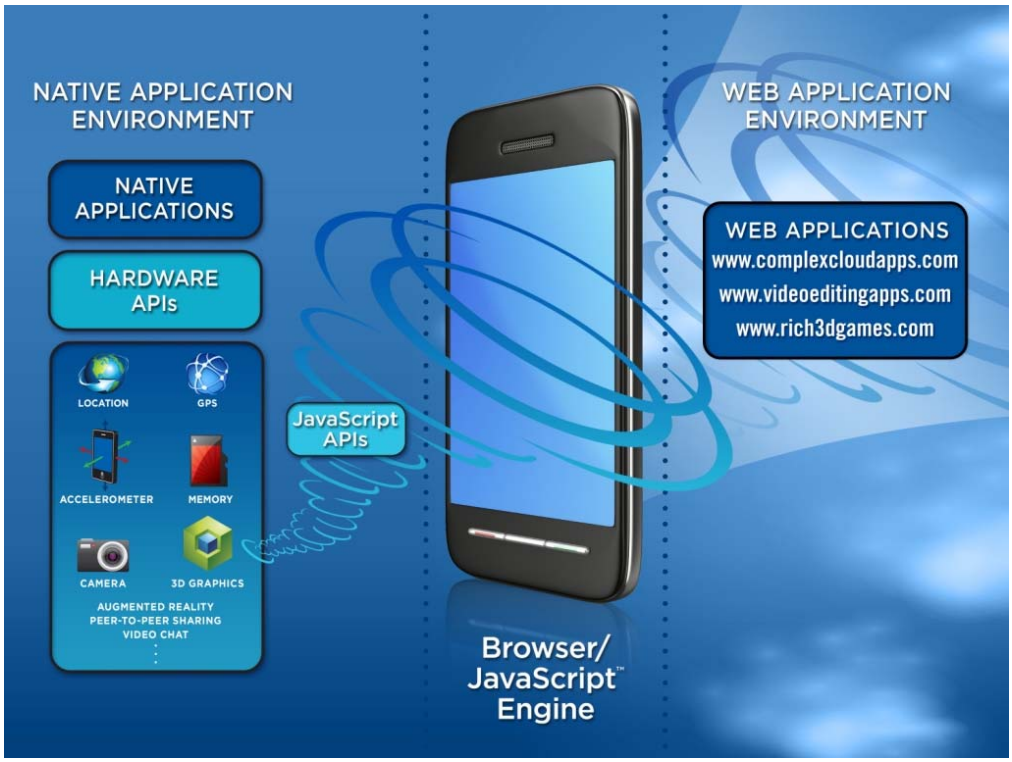


Figure 1 – The goal: Web applications access hardware resources in the device

Conversely, native applications like games, productivity software and even address books actually live inside the device. These applications are bound to the operating system of the device and developers must rewrite (port) them to run on other devices. Yet despite their limited portability, these native applications enjoy the full advantages of their proximity to the hardware's application programming interfaces (APIs): audio, memory, CPU, bus, file system. This proximity contributes to generally higher performance and better user experience.

The future of the browser, as described in "Handling Mobile Web Pages Better" and "Streaming Multimedia More Smoothly" of this series, lies in making the container smarter and faster, to move bits back and forth through the window more efficiently. But the real potential of Web applications running in the browser lies in making them more capable so that, from the Web side of the window, they can access hardware more effectively and run on par with native applications.

Complex Web applications for the real world

What would that look like? Here are some examples:

- An auction site could include a “take photo” button so that the user could write a description of the object for sale, photograph it with an in-browser camera and post it to the Web – entirely within the browser.
- An online meeting application could allow participants to use the camera for video conferencing – without downloading and installing native software.
- A rich, standalone game incorporating both 3D graphics and augmented reality could be developed right in the browser, with almost no performance penalty.
- Location-based services that annotate maps and real-time views with restaurants and other businesses could help users interact with their surroundings.
- Standalone Web-based applications for audio mixing and photo editing could synchronize users’ photos and ringtones whenever their device is connected.

THE KEY TO THIS FUTURE LIES IN EXPOSING THE DEVICE’S HARDWARE APIS AS AMPLY IN THE BROWSER AS IN THE NATIVE OPERATING SYSTEM THAT RUNS THE DEVICE.

The key to this future lies in exposing the device’s hardware APIs as amply in the browser as in the native operating system that runs the device. That way, even though the Web is on the other side of the window, it can still see, hear and access the device’s resources.

Native environment and Web environment – More overlap ahead

Different users want to use different types of mobile application, but they all expect parity in user experience. The argument is compelling enough in the desktop world: consider the rising popularity of Web-based email applications compared to native email client software. Yet this phenomenon is even more compelling in the mobile world, where hardware resources like GPS, sensors, accelerometers and the wireless modem can greatly deepen the user experience.

While Web applications may run and perform differently from one browser to another, the effort to deal with the differences will pale in comparison to the porting/test/distribution/support effort required with native applications.

In the current mobile application paradigm shown in Figure 2, one side’s advantage is the other side’s disadvantage, and the common ground of always-on-always-connected has more to do with the device than whether the application was written to run on the device or from the Web:

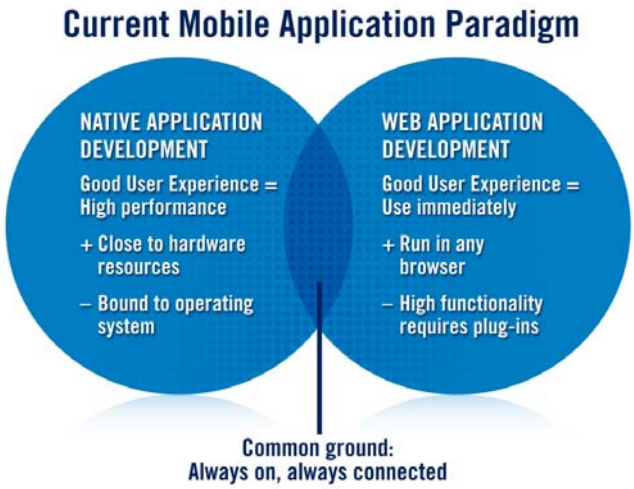


Figure 2 - Current Mobile Application Paradigm

The advent of richer Web-based applications and everything that comes with them has the potential to broaden the common ground between native application development and Web application development, as shown in Figure 3:

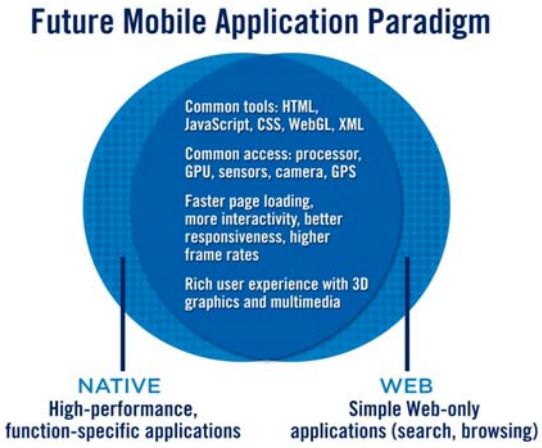


Figure 3 - Future Mobile Application Paradigm

There will certainly continue to be a role for high-performance, function-specific, native applications, and for simple Web development such as pages and even Web-based email portals. But the common ground will burgeon as access to tools and resources benefits both sides.

Making Web Applications Work - Fundamentals

There are three areas in which to set the stage for these Web applications:

1. Browsers and JavaScript
2. HTML5 technologies
3. Infrastructure for Web applications

1. Browsers and JavaScript

No sooner did interest in desktop browsers begin to subside than the mobile Web opened a second front, keeping the industry scrambling for more features. Table 1 lists today's top-tier players:

Browser engine	JavaScript engine (virtual machine)	Distribution
Trident	Chakra	Internet Explorer®
Gecko	Spidermonkey, JägerMonkey	Firefox™
Presto	Carakan	Opera
WebKit	Nitro	Safari, RIM, Nokia
WebKit	V8 JIT	Chrome, Android™, webOS, Brew® Mobile Platform

Table 1 - Browser/JavaScript landscape

THE BROWSER/ JAVASCRIPT ENGINE COMBINATION OF WEBKIT/V8 ALLOWS OPTIMIZATION IN TWO OF THE AREAS MOST IMPORTANT FOR WEB APPLICATIONS.

Several of these combinations – especially WebKit/V8 – allow optimization in two of the areas most important for Web applications.

Hardware-accelerate specific tasks

Instead of relying on the operating system on which they sit, the browser can hand off the work of graphics processing from the CPU, where it takes place with relatively slow software libraries, to dedicated hardware. An example is the GPU-accelerated frame composition in both Internet Explorer 9 and some of the latest WebKit-based browsers.

Execute JavaScript faster

JavaScript contains all the programming capability that designers weave into a Web page. Examples include mousing over an advertisement to start it playing, entering information into an online order form, uploading a picture and cropping it directly on a website, and accessing a device's location data from a Web page.

JavaScript is extensible. It's possible to extend its capabilities through new JavaScript APIs that let Web applications access device resources through the window that separates Web applications from native applications. Consequently, there has been a recent trend of investment by Apple, Google, Microsoft, Mozilla and Opera in improving JavaScript performance, such that it has become an important metric in the browser wars. Current JavaScript engines (or virtual machines) are 10-15 times faster than those available only a couple of years ago, setting the stage for running complex Web pages and Web applications, even on lower-powered smartphones and feature phones.

Two of the most common benchmarks for measuring relative JavaScript performance are WebKit's SunSpider and Google's V8 Benchmark. SunSpider is more commonly used and has a collection of tests with short execution times. The newer V8 Benchmark, preferred by Google's V8 team, weighs execution times more than compilation times.[1]

THE PROMISE OF HTML5 IS THAT THESE TECHNOLOGIES WILL WORK IN THE BROWSER WITHOUT PLUG-INS AND OFFER NATIVE-APPLICATION PERFORMANCE WITH WEB-APPLICATION PORTABILITY.

2. HTML5 technologies

Several browsers have already adopted technologies that put Web applications on the path to parity with native-application functionality. All of these technologies are from open-source groups and consortia; none of them is proprietary to any single company.

Some of the functionality – like playing video – already has a long history of working in a browser, but only through a browser plug-in, which represents another layer of separation between the Web application and the hardware. The promise of HTML5 is that these technologies will work in the browser without plug-ins and offer native-application performance with Web-application portability:

[1] *Qualcomm and QuIC use SunSpider, V8 and other benchmarks to measure performance.*

HTML5 <video> tag – Provides native support in the browser to play back multimedia. Currently, most of the video on the Web runs in the browser through the Adobe® Flash® Player and other plug-ins. If the <video> tag gains wide acceptance, it has the potential to displace plug-ins such as Flash Player that have been commonly used for in-browser video playback. (See "Streaming Multimedia More Smoothly" for more details.)

HTML5 <canvas> tag – Provides native support in the browser for vector graphics. Web graphics – buttons, banners, ads – are currently static, x-by-y pixel images. The <canvas> tag lets Web designers use images that the browser can scale and resize, for JavaScript-driven drawing and animations.

WebGL API – A native application can take advantage of the device's Graphics Processing Unit (GPU) for 3D graphics. Through this JavaScript API, WebGL endows Web applications with that same advantage.

Caching/offline storage – Web applications depend on a connection to the network for most of their functionality; native applications are self-contained, so they always run. Caching and offline storage allow the Web application to run offline, such as when network connectivity is limited or unavailable.

DEVICE APIS EXTEND TO COMPLEMENTARY, NON-HARDWARE TECHNOLOGIES LIKE PEER-TO-PEER (P2P), AUGMENTED REALITY, COMPUTER VISION, INDOOR NAVIGATION AND CONTEXTUAL AWARENESS.

Device APIs – In the same way that the aforementioned WebGL API gives Web applications access to the GPU, device APIs give them access to mobile-specific hardware such as sensors, GPS, camera and audio, through JavaScript. Furthermore, device APIs extend to complementary, non-hardware technologies like peer-to-peer (P2P), augmented reality, computer vision, indoor navigation and contextual awareness.

Geolocation API – This JavaScript API allows a Web page or application to determine the location of the device, and whether it is in landscape or portrait mode.

Web Workers – Native applications can run multiple threads and take advantage of all available cores on the CPU, whereas Web applications are subject to the JavaScript bottleneck of running in a single thread on a single core. The Web Workers specification defines an API for running multiple, processor-intensive threads in the background on different cores, without blocking the main page or other running scripts.

3. Infrastructure for Web applications

Just as native applications do not live in a vacuum, the most complex Web applications rely on surrounding infrastructure and rules.

Application framework

An application framework provides a consistent environment in which developers can build applications. It helps them define look and feel, behavior upon launch, use of Web components and installation procedure. Most application frameworks provide libraries to standardize methods for database access, session management and user interface components.

PACKAGING HELPS DEVELOPERS MONETIZE THEIR APPLICATIONS AND SECURITY IS DESIGNED TO PROTECT USERS FROM ROGUE APPLICATIONS.

Packaging and security

The Web application lives in a package that indicates which company developed it, the processing capabilities required to run it on a device, and installation/runtime/offline verification. Packaging helps developers monetize their applications and ensures that applications are targeted to appropriate devices.

Security is designed to protect users from rogue applications and verifies that the Web application has adequate privileges to, say, open the contact list or read location information.

Unfortunately, these important areas of infrastructure are marked by fragmentation. W3C, OMTP, BONDI and WHATWG have attempted to establish standards for packaging and security, but these attempts have variously been incomplete, slow or divided by stakeholder-companies. These standards are conducive and vital to an environment in which developers are comfortable marketing complex Web applications.

Developing Richer Web-based Applications

Given the role of browsers, JavaScript and HTML5 in Web applications, QuIC has made optimizations that help set the stage for richer Web-based applications:

Hardware acceleration

Although browsers are software applications running on the device's CPU, many browser functions like page composition and graphics rendering run faster and more efficiently in dedicated hardware in the mobile processor. Offloading these tasks to hardware – exp. on Snapdragon-powered devices, benefits the user through increased performance and extended battery life.

(See "Compose the page" in the paper "Handling Mobile Web Pages Better" for details on QuIC's optimization.)

JavaScript performance

Native applications enjoy ample access to in-phone components and functions like audio, memory and display through device APIs. For more Web application features, QuIC has enabled JavaScript to access the richest set of device APIs, starting with the camera. Future API sets will include GPS, sensors, augmented reality and peer-to-peer (AllJoyn) functionality.

WebGL and HTML5 <canvas>

To reach parity with the 2D and 3D graphics performance of native applications – especially important for mobile gaming – QuIC has modified browser code for WebGL and the HTML5 <canvas> tag. The result is that Web applications can render graphics at frame-rates approaching those of native applications.

THE QUIC OPTIMIZATIONS ARE FEATURE ADDITIONS TO THE SOFTWARE THAT QUALCOMM SHIPS WITH ITS SNAPDRAGON MOBILE PROCESSOR.

Packaging and security

Finally, with the growth of Web applications, the Web Technologies initiative includes defining and implementing a packaging and security mechanism that applies to Web applications running on optimized browsers and Qualcomm's mobile processors. QuIC's work on Android and Qualcomm's work on Brew Mobile Platform will initially support WebKit-based browsers. As market opportunities permit, they plan to work on other browser engines as well.

The QuIC optimizations are feature additions to the software that Qualcomm ships with its Snapdragon mobile processor. They represent extra value in hardware and competitive advantage for Qualcomm's customers.

Conclusion

Qualcomm and QuIC are investing in the development of richer Web-based applications for several reasons:

- Web applications offer developers the promise of better portability across operating systems and hardware platforms. The optimizations in the Web Technologies initiative are chipping away at the perceived disadvantages of Web applications relative to native applications – performance, device access and security – to allow developers to create richer applications and address a larger market. It will soon be a viable proposition to create a Web application with as much functionality as a native application.

- Native applications can already take advantage of the phone's hardware resources. Android's rich camera APIs, for example, allow native applications not only to take photos, but also to set to grayscale mode, use face detection, control zoom and much more. Through JavaScript APIs, Web applications can also benefit from this richness of functionality.
- Web applications are the future for a large set of application developers still learning to write with mobility in mind. Qualcomm's momentum in mobile processors and QuIC's expertise in open-source software are big steps on the path to the Truly Mobile Web.

Follow Us

How are we doing it? What does it mean? How does it apply to your devices, networks, applications and customers? See the other papers in this series from Qualcomm and QuIC, available in the Web Technologies section of the Qualcomm Developer Network:

- Six Issues We Need to Address First
- Handling Mobile Web Pages Better
- Streaming Multimedia More Smoothly
- Developing Richer Web-based Applications

Follow us on the Web:



OnQ blog

Authors

Sayeed Choudhury, Director of Product Management, Web Technologies, Qualcomm Incorporated - sayeedc@qualcomm.com

Mark Bapst, Vice President of Engineering, Web Technologies, Qualcomm Innovation Center, Inc. - mbapst@quicinc.com

Qualcomm Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Qualcomm Innovation Center, Inc.
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.