HTML5 Developer's Guide

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: March 30, 2015

Contents

About This Guide

This guide explains how to develop user interfaces for apps created with the QNX SDK for Apps and Media using HTML5.

This table may help you find what you need in this guide:

To find out about:	Go to:
An overview of using the HTML5 SDK to develop apps for QNX SDK for App and Media	*Overview of the QNX SDK for HTML5* (p. 9)
The HTML5 Development environment for QNX SDK for App and Media	*The QNX SDK for HTML5 and where to download it* (p. 13)
What's supported for HTML5, such as the HTML5 elements (audio, video, etc.) support, CSS3 support, etc.	*HTML5 support in the Browser Engine* (p. 23)
How applications (or apps) run in complete isolation from each other using a *sandbox*	*Web Sandbox Model* (p. 21)
The process for creating, installing, and running an HTML5 app	*Developing HTML5 Apps* (p. 27)
Creating custom Cordova Plugins	*Creating Custom Cordova Plugins* (p. 71)
Using PPS in an HTML5 app	*Example: Using the PPS interface* (p. 74) *Example: Adding a permission to access a PPS object* (p. 50)
Creating plugins to extend your app's functionality	*Creating Custom Cordova Plugins* (p. 71)
HTML5 Reference samples	*HTML5 Application Samples* (p. 17)
Best practices for optimal performance	*Enhancing Performance* (p. 70)

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	NULL
Data types	**unsigned short**
Environment variables	*PATH*
File and pathnames	**/dev/null**
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

WARNING: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1
HTML5 SDK Overview

The QNX SDK for HTML5 (called the HTML5 SDK in this document) provides a framework for developing and executing web-compatible applications, specifically using technologies such as HTML5, CSS3, and JNext.

You *install the QNX SDK for HTML5 package* (p. 13) to develop HTML5 apps. The HTML5 SDK is based on Apache Cordova 3.6.0. Apache Cordova is a framework used to develop apps for mobile devices. The HTML5 SDK exposes the aspects of Cordova most useful for building the user interface and for building plugins to access native functionality for the QNX SDK for Apps and Media. The HTML5 SDK is necessary for you to develop apps that run on target images running QNX SDK for Apps and Media. You don't need to install the QNX SDK for Apps and Media unless you want to build a target image, or develop your own native service and expose it to HTML5.

Before you use this guide, you should be familiar with developing Cordova apps, in particular, the Platform-centered workflows. For more information about developing Cordova apps, see the *Cordova documentation*.

HTML5 apps and plugin development

You can write your HTML5 apps and plugins using Cordova APIs for Apps and Media. *Cordova JavaScript Plugins* and Webplatform APIs are provided as part of the HTML5 SDK to help you with development.

The following diagram maps the various APIs that are available to you. Use it as a roadmap to choose what to use when you develop your HTML5 app.

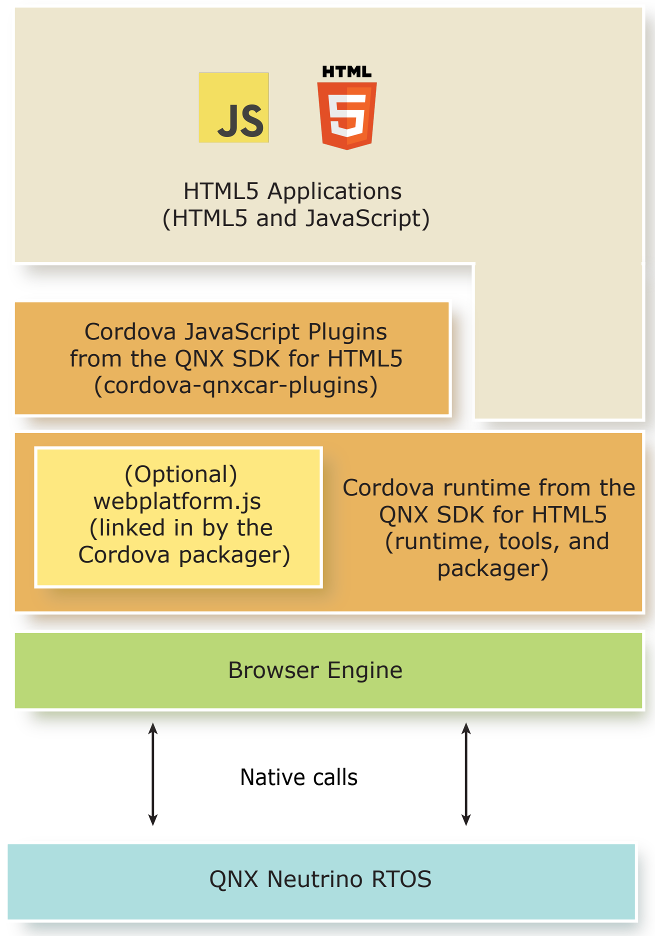


Figure 1: APIs available with the QNX SDK for HTML5

Typically, HTML5 and JavaScript with the provided Cordova JavaScript Plugins are sufficient for most development requirements. In some situations, however, you may find it useful to write your own custom plugin to access native functionality that the HTML5 SDK doesn't expose. If you determine that you require custom plugins to access native functionality on the platform, you have the option to use the Webplatform APIs. The Webplatform APIs are useful to access the WebView and simplify working with Persistent Publish/Subscribe (PPS) objects on the platform. In rare situations, you may also find the *WebLauncher APIs* (p. 81) useful for accessing functionality that you require.

Overview of components to develop HTML5 apps

There are several components available as part of the HTML5 SDK. You can use some or all the components for development, according to your requirements:

Apache Cordova (HTML5, JavaScript, and CSS)

Apache Cordova is a framework for application development that allows you to use web technologies such as HTML5, JavaScript and CSS to create apps for mobile devices. You can develop HTML5 apps, and with the tools provided in the HTML5 SDK, you can:

- package an HTML5 app to deploy it to the target hardware
- create JavaScript APIs (Cordova Plugin) to access native services on the platform
- use *webserver emulation* so that your app can run without actually running a webserver on the target hardware
- provide security features to your app

HTML5 is useful for building responsive interfaces because it allows you to modify CSS files to easily customize the look and feel of your apps. You can use various tools, such as Compass and Sass to modify your CSS. For information about Compass and Sass, see "*Install CSS development tools* (p. 16)".

HTML5 apps can use JavaScript plugins to interact with underlying services. JavaScript APIs provide interfaces to various feature components. The Netscape Plugin Application Program Interface (NPAPI) provides extensions to the HTML5 engine through a dynamically linked library. PPS, SQL, and the UI core APIs are implemented through a NPAPI interface. These APIs provide access to native DLLs that give apps access to services such as the composition manager and the launcher. If more such services are required, you can add additional plugins to extend your apps.

Cordova JavaScript Plugins and Cordova Runtime

The HTML5 SDK contains the *Apache Cordova* runtime, custom tools, and plugins to build apps for Apps and Media. You can also use the HTML5 SDK as an example to create custom plugins for your apps that follow the Cordova plugin specifications.

Included as part of the HTML5 SDK are Cordova JavaScript Plugins that you can use to build your apps. These plugins expose functionality that's available on Apps and Media. The version of the Cordova runtime included with the HTML5 SDK provides the development environment you need to create, build, and run HTML5 apps on Apps and Media. Webplatform (**webplatform.js**), an optional API included with the runtime, makes it easy to work with WebViews and PPS.

Persistent Publish/Subscribe (PPS) API

The PPS service provides a simple, filesystem-based API for client applications. PPS is used to share information between processes and apps on a target. There are various PPS objects available that you may find useful for accessing data for your HTML5 apps. For more information about PPS objects, see the *Persistent Publish/Subscribe Developer's Guide* and *PPS Object Reference*.

Node.js

Node.js is a platform built on Chrome's JavaScript runtime for building fast, scalable network apps. Node.js (available from ***http://nodejs.org/***) is required—you installed Node.js when you installed the HTML5 SDK. You can use Node.js to install Plugman, which is a required tool to manage plugins in your Cordova project.

Web Graphics Library (WebGL)

This graphics library is integrated into the *browser engine* (p. 23), so you don't need to download and install any binaries to use it. Based on OpenGL ES 2.0, WebGL is a cross-platform JavaScript API. As a Document Object Model (DOM) API, it runs in the HTML5 `canvas` element to render interactive 3D graphics in compatible browsers.

Chapter 2
The HTML5 Development Environment

The HTML5 SDK is required to build and package HTML5 apps to **.bar** files. These **.bar** files can be built or installed into a system built with the QNX SDK for Apps and Media. image.

The HTML5 SDK contains the *Cordova* framework, runtime, tools, and APIs that you can use to create plugins and HTML5 apps. The HTML5 SDK packages HTML5 apps as `.bar` files. You, or another developer, can then integrate these **.bar** files into a target image so your apps are available by default. You can also use these **.bar** files to install your apps on the target at a later time.

You can download the QNX SDK for HTML5 (**.zip** file) from the *QNX Download Center*. To install the package, follow the installation notes for the QNX SDK for HTML5 corresponding to this release.

Managing plugins

To manage plugins, use *Plugman*. Plugman was installed into your environment using the **Node.js** package manager (npm) when you installed the HTML5 SDK (i.e., `npm install -g plugman`). You use `plugman` to manage Cordova plugins from within your HTML5 projects. If you want multiple apps to use a plugin, you must run the `plugman` command for each app that needs to use the plugin. The `plugman` command is on a per project (or app) basis.

To add a plugin from your project, use the following command:

```
plugman install --platform blackberry10 --project folder to project
                --plugin idlpathlurl [--variable NAME=name]
```

For example, from within your Cordova project directory, you can use the `plugman` command to add the demo plugin from the HTML5 SDK:

- On Windows:

  ```
  plugman install --platform blackberry10 --project .
                  --plugin HTML5 SDK install location\html5sdk\demo\com.qnx.demo
  ```

- On Linux:

  ```
  plugman install --platform blackberry10 --project .
                  --plugin HTML5 SDK install location/html5sdk/demo/com.qnx.demo
  ```

To remove a plugin from, your project, use the following command:

```
plugman uninstall --platform blackberry10
                  --project folder to project --plugin id
```

To list installed plugins for your project:

```
plugman ls
```

For information about Plugman, see *Using Plugman to Manage Plugins* in the *Cordova Documentation*.

HTML5 SDK files overview

After you unarchived the ZIP file, the following folders appear under the ***Path you installed the QNX SDK for HTML5*/html5sdk**:

cordova

Contains the library of plugins developed for Cordova, as well as the QNX Cordova runtime and packager. These are the common tools that you require:

`create`

Creates a new HTML5 project. The project includes a skeleton project and the required tools for building and packaging an HTML5 app. For more information about the `create` command, see "*Creating an HTML5 project* (p. 30)"

`whereis.cmd`

Returns the path of the file you specify as a parameter based on a search of the directories in your *PATH* variable. This command is available only for Windows. For example:

```
whereis.cmd index.html
```

`templates`

Contains the folder structure and file templates for new projects. The default template creates a Cordova app that handles the `deviceready` event and displays the Cordova logo on the screen.

demo

A sample app that you can build and deploy to your image. You can see more sample HTML5 apps in the QNX SDK for Apps and Media. Ensure that you use the tools from this HTML5 SDK to build and package the tools. The tools are located in the **tools** folder.

tools

The tools used by the Cordova runtime to create and build Cordova projects for a target.

To use the tools, you add the path to either the Windows or Linux version of the tools to your *PATH* variable on your host computer where *installation path* is the location you extract the HTML5 SDK.

For Windows, the path is:

```
HTML5 SDK install location\html5sdk\tools\packaging\win32\bin
```

For Linux, the path is:

```
HTML5 SDK install location/html5sdk/tools/packaging/linux/bin
```

You don't use these tools directly, but the Cordova tools have a dependency on these tools:

`blackberry-nativepackager`

Packages your compiled binaries for your app as a **.bar** file.

`blackberry-debugtokenrequest`

Creates debug tokens on your computer. Debug tokens are not necessary for packaging your apps.

`blackberry-signer`

Signs the app if signing keys are provided. This version of the QNX SDK for Apps and Media doesn't support signed apps. Even though the tools support signing, it isn't necessary to sign your app or provide signing keys in this release.

`blackberry-deploy`

Deploys the package created for your app.

Other tools

You can edit files using your favorite HTML5 development tools. Here are some tools that can help you with HTML5 development:

- For CSS development, use Compass and Sass. For more information, see "*Install CSS development tools* (p. 16)."
- For debugging your apps, install a Webkit-based browser (such as Google Chrome), which work with Web Inspector. For more information about Web Inspector, see "*Debugging Web Apps* (p. 53)" in this guide.

Install CSS development tools

(Optional) In addition to installing and setting up the QNX SDK for HTML5, consider downloading Compass and Sass, which are useful tools for CSS modification and development.

You can modify existing or create CSS files (.scss) that reside on your target using these tools:

- *Compass* — an open-source CSS authoring framework
- *Sass* — a CSS extension language, which gets installed when you install Compass

Before you install Compass or Sass, you must have *Ruby* installed on your host computer. On Linux, you can use `rbenv` and RVM to install Ruby. But, if you are using a Windows host, you can use RubyInstaller or `pik`.

To install Compass and Sass and create Compass project files:

1. Type one of the following commands to install Compass and Sass:
   - On Windows, type `gem install compass`.
   - On Linux, type `sudo gem install compass`.
2. To verify that compass was installed, type `compass version`. If your installation was successful, a version number is returned.
3. Navigate to your project or working directory. You can add Compass project files to your working directory by typing `compass init`.

After you complete the steps above, you should see a file called **config.rb** in your working directory. You can use the **config.rb** file to configure the output for your project. For more information about configuring Compass, see the *Compass Configuration Reference*.

You may also find the following information useful for learning more about Compass and Sass:

- *Compass tutorials* that cover configuration, commands, best practices, and more.
- The *Sass site*, which has a comprehensive documentation covering variables, mixins, and so on.
- *Syntactically Awesome Stylesheets: Using Compass in Sass*

Chapter 3
HTML5 Application Samples

These are the HTML5 samples that are available:

- AudioDemo
- Browser Lite
- CordovaPPSDemo
- PeaksAndValleys
- Shutdown
- VideoDemo
- VideoDemo480 (same as the VideoDemo sample, but configured for targets that use 480p resolution)

The HTML5 samples are described in the *User's Guide* and included as part of the QNX SDK for Apps and Media.

The source code for the samples is also available:

- The sample code for the PPS demo is located at *Path you installed the QNX SDK for HTML5***/html5sdk/demo**.
- All the other samples are found in an archive file located at *base_directory***/qnx660/source/appsmedia_html_source_v1_1.zip** where *base_directory* represents where you installed the QNX SDK for Apps and Media. You can extract the contents into a separate directory to work with.

To build and deploy the samples, use the HTML5 SDK. For example, you can build the Cordova PPS Demo app that's included with the HTML5 SDK. You can also modify the project to further understand how to build an HTML5 app or use it as a basis for your own apps. For information about how to build the Cordova PPS Demo app, see *Building and deploying the Cordova PPS Demo* (p. 18).

Building and deploying the Cordova PPS Demo

The Cordova PPS demo is delivered as a separate Cordova component. You can build the demo to learn how to create a project, install a plugin, and use Persistent Publish/Subscribe(PPS) objects with an HTML5 app.

About the Cordova PPS demo

If you are using a QNX SDK for Apps and Media image, the steps regarding configuring the **sys.acl**, **sys.res**, and **pps.conf** files are already completed for you. In fact, the image contains the Cordova PPS Demo app installed. If you choose, you can recreate the application and deploy it onto the image. Before you deploy the app you build, it's a good idea to remove the previous version of the Cordova PPS Demo using the `bar-uninstall` command.

To run this demo, ensure that you have installed both the QNX SDK for Apps and Media, and the HTML5 SDK. The code for the HTML5 SDK is located at: *HTML5 SDK install location*/**html5sdk/demo**.

For more information, see *The HTML5 Development Environment* (p. 13).

The instructions below describe how to create, deploy, and launch the Cordova PPS Demo app. After you have completed these tasks, you can use the instructions you followed as a model for other HTML5 projects.

Host system

On your host system, use the files in the **demo** folder as a template to create the demo app:

1. In a command-line window, navigate to the directory where you installed the HTML5 SDK:
   - On Windows:

     `cd` *HTML5 SDK install location*\html5sdk

   - On Linux:

     `cd` *HTML5 SDK install location*/**html5sdk**

2. Create the app template using the `create` command:
   - On Windows:

     ```
     cordova\cordova-qnxcar\bin\create CordovaPPSdemo
     ```

   - On Linux:

     ```
     ./cordova/cordova-qnxcar/bin/create CordovaPPSdemo
     ```

3. Copy all the files from the **www/** directory into the newly created app's **www** directory:

- On Windows:

```
 xcopy /E demo\www CordovaPPSdemo\www
```

- On Linux:

```
cp -R ./demo/www/* CordovaPPSdemo/www
```

4. Add the required plugins using the `plugman install` command:
   - On Windows:

```
plugman install --platform blackberry10 --project ./CordovaPPSdemo
                --plugin ./demo/com.qnx.demo
```

   - On Linux:

```
plugman install --platform blackberry10 --project ./CordovaPPSdemo
                --plugin ./demo/com.qnx.demo
```

5. Build and package the app using the `build` command with the `debug` option.
   - On Windows:

```
.\CordovaPPSdemo\cordova\build debug
```

   - On Linux:

```
./CordovaPPSdemo/cordova/build debug
```

The generated BAR file is located at **.\CordovaPPSdemo\build\device\qnxcarapp.bar** (on Windows) and **./CordovaPPSdemo/build/device/qnxcarapp.bar** (on Linux).

The **.bar** file is called `qnxcarapp` by default. If you manage multiple projects, consider renaming the **.bar** file to match your app name . For more information, see *Packaging an HTML5 app* (p. 45).

Target system

On your target system:

1. Deploy the app on the target using one of the following options:
   - You can use a USB stick to deploy the **.bar** file to the target. Copy the **qnxcarapp.bar** file from the previous step to your USB stick, and insert the stick into your target. Then, on your target, run the following command to deploy from your USB stick:

```
/scripts/bar-install /fs/usb0/qnxcarapp.bar
```

- If your target and host are on the same network, use a tool such as FTP or SCP to transfer the **qnxcarapp.bar** file to the **\tmp** directory on the target. Then, run the `bar-install` command as follows:

  ```
  /scripts/bar-install /tmp/qnxcarapp.bar
  ```

When the `bar-install` command is finished, it will display output like the following:

```
>> Added PPS entry
CordovaPPSdemo.testDev_dovaPPSdemod339185a::native/default-icon.png,
Cordova PPS Demo,,,auto,,
```

Make sure you record the filename (in this example: `CordovaPPSdemo.testDev_dovaPPSdemod339185a`). You will need it when you *specify the access* (p. 20).

2. Add the following entries to the **/base/etc/pps.conf** file to initialize the PPS object:

   ```
   qnx/demo
   0:0:0660:O_CREAT
   user::rw
   group::rw
   other::rw
   mask::rw
   ```

3. Set up the Authorization Manager (`authman`) configuration. To do this, in the **/etc/authman/sys.acl** file declare the `access_demo` capability, and the permissions to the relevant PPS objects:

   ```
   access_demo
   ACL opt rwx:rw /pps/qnx/
   ACL opt rwx:rw /pps/qnx/demo
   ```

4. Add the following entry to specify the access for the capability you declared in the previous step in the **/etc/authman/sys.res** file. Use the filename you recorded in *Step 1 above* (p. 19):

   ```
   access_demo
   deny *
   allow CordovaPPSdemo.testDev_dovaPPSdemod339185a
   ```

5. After you finish modifying the **pps.conf**, **sys.acl**, and **sys.res** files, reset your target. For more information about Authorization Manager, see Authorization Manager (`authman`) in the *System Services Reference*.
6. On your target, start and run the app. If you have an HMI running, tap **Cordova PPS demo**; otherwise type the following command at the command prompt of the target:

   ```
   # launch CordovaPPSdemo
   ```

For information about using the Cordova PPS App, see “Cordova PPS Demo” in the *User's Guide*.

Chapter 4
Web Sandbox Model

The QNX SDK for Apps and Media uses *sandboxing*. Sandboxing allows you to runs apps in their own, separate instances of the HTML5 engine. It lets you isolate apps so that incorrect behavior in one app won't affect other apps. For example, without sandboxing, a blocked or stalled JavaScript thread in one app could prevent other HTML5 apps from running.

You can use sandboxing to separate your core, trusted apps from all the other apps. With this architecture, multiple WebViews (equivalent to a tab in a desktop browser or window) can either share a common engine instance (core apps) or run in their own engine instances (other apps). While the sandbox model does increase the system's memory footprint, the ability to isolate untrusted apps in their own HTML5 engine instances increases your system's dependability.

Chapter 5 Browser Engine

The browser engine is based on the WebKit and supports features such as canvas, WebSocket, Document Object Model (DOM) improvements, session storage, offline apps, worker threads, and WebGL.

About the browser engine

The browser engine provides support for HTML5, associated technologies (including CSS3 and JavaScript), and standards, such as AJAX, JavaScript Object Notation (JSON), and XML. The browser engine is based on the WebKit (*www.webkit.org*) open-source web browser engine. The browser engine has also been extended and optimized for embedded systems in the following ways:

- improved user interaction (complex touch event handling, smooth zooming/scrolling, fat-finger touch target detection, etc.)
- improved performance and battery life for mobile devices
- enhanced user operations such as fast scrolling and zoom (e.g., zooming in on a webpage) to reduce RAM utilization
- enhanced JavaScript execution that improves performance and reduces CPU utilization
- reduced power consumption (e.g., by throttling background threads) and reduced battery drain
- added support for multimodal input (e.g., trackpad, keyboard, and virtual keyboard)
- improved overall speed (e.g., by selective image down-sampling)

By default, the browser engine implements the most basic browser functionality: the ability to follow links and to download and display content. You can use the engine's functionality at the most basic level to display web content in your app, or you can use Webplatform APIs to create your own full-featured, customized, web-based app.

The great variety of content and encoding types used on the Internet makes the simple tasks of browsing and downloading content from the web a daunting task for a browser—and the browser designer. The SDK's browser engine handles different content types transparently. It creates and manages the objects necessary to render the incoming content, and provides the view classes used to display content. Each view class (called a *WebView*) contains frames (called *WebFrames*); each frame implements its own scroll bar. You don't need to implement custom views or custom frames to display content in your app.

Sandboxing permits each app runs in its own instance of the browser engine, so that bad behavior in one app doesn't affect all other apps. For more information about sandboxing, see *Web Sandbox Model* (p. 21).

Sample Browser apps

The QNX SDK for Apps and Media includes two reference apps (Browser and Browser Lite). Both reference apps are browsers that utilize the browser engine. The Browser Lite app provides a sample that shows you how to build a web browser. The app is implemented with HTML5, CSS3, and JavaScript. The Browser Lite app provides basic components, such as a URL address entry, back and forward buttons, while the Browser app provides components (bookmarks, toolbars, etc.) that you would expect from a desktop browser. A browser without these components is called a *chromeless browser*. You can customize or replace the Browser Lite app because the source code is available to you. For more information about reference apps, see “ Browser” and “Browser Lite” in the *User's Guide*.

Browser engine components and technology support

The browser engine includes components such as native and JavaScript plugins, and supports technologies such as the HTML5 application framework, CSS, and standard APIs.

HTML5 apps

The HTML5 application framework provides the additions the browser engine needs to support full-fledged apps. This environment allows you to create and deploy apps built from web technologies (HTML5, CSS3, and JavaScript) with plugins that provide access to the underlying device hardware and native services, just like native C/C++ apps.

Native plugins

The browser engine includes, through a dynamically-linked library, plugins based on the Netscape Plugin API (NPAPI). These native plugins provide access to PPS (Persistent Publish/Subscribe), SQL, and Screen services. You can add more plugins as required. For example, you can add the native SQLite 3 plugin, which provides SQLite database access, including a complete API for opening, querying, and modifying a database.

JavaScript plugins

JavaScript Cordova plugins use the browser engine plugins to provide HTML5 apps with access to middleware-layer services. The reference image for the SDK for Apps and Media includes the browser and power plugins. For example, the Browser Lite app uses the browser plugin to build a simple web browser.

Web Inspector tool

Included as part of WebKit , *Web Inspector* is a useful debugging and profiling development tool for web content. You can use this tool to troubleshoot and optimize your web content for your apps. The tool includes features and capabilities such as inspection, profiling, and console integration. For details, see "*Debugging Web Apps* (p. 53)" in this guide.

CSS support

The browser engine supports CSS3 properties. You can modify CSS files to customize the appearance of your app. Consider using Sass and Compass to help you customize CSS files. For a complete list of supported CSS3 properties for WebKit-based browsers, see the *CSS3 Browser Support Reference* at the following W3Schools site:

www.w3schools.com/cssref/css3_browsersupport.asp

HTML5 elements

The HTML5 SDK supports the use of HTML5 elements in your apps. The table below describes some of the more common elements used to develop HTML5 apps. For more information about these elements, see the corresonding W3Schools references:

Element	Description
HTML5 Audio	Represents a sound or audio stream.
HTML5 Canvas	Provides a container for JavaScript to draw graphics on a webpage.
HTML5 Geolocation	Scripts use this object to programmatically determine the location information associated with the hosting device.
HTML5 Web Storage (`localStorage`)	Provides functions to access a list of key/value pairs for *local storage objects* (i.e., objects that persist after a browser session has ended).
HTML5 Web Storage (`sessionStorage`)	Lets you save a large amount of key/value pairs and text for *session storage objects* (i.e., objects that are valid only for the current browser session).
HTML5 Video	Represents a video or video stream.
HTML5 Web Workers	Allows JavaScript code to be executed in a background thread.

Browser API and offline support

HTML5 includes several features that address the challenge of building web apps that work offline. These features include SQL, offline app-caching APIs, `online`/`offline` events, status, and the `localStorage` API. For more information about creating web apps that work offline, see the W3C document *Offline Web Applications*.

The browser engine supports various standard APIs. For information about these APIs, see the following W3C resources:

API Support	Description
Web SQL Database	A set of APIs for manipulating client-side databases using SQL.
WebSocket API	An API that allows webpages to use the WebSocket protocol. This protocol enables web apps to maintain bi-directional communications with a remote host.
Web Workers	An API that allows authors of web apps to spawn background workers running scripts in parallel to their main page. This process allows for thread-like operation with *message passing* as the coordination mechanism.
Geolocation API Specification	An API that provides scripted access to geographical location information associated with the hosting device.

Chapter 6
Developing HTML5 Apps

The HTML5 app environment lets you create and deploy apps built from web technologies (HTML5, CSS3, and JavaScript) with Cordova plugins. These plugins can access the underlying device hardware and native services, just like native C/C++ apps.

Overview

An HTML5 app created with the *Apache Cordova* framework can be targeted to run on different devices, or ported from a different OS, such as iOS, BlackBerry 10, or Android.

You can take advantage of the available mobile web frameworks, such as Sencha Touch and jQuery Mobile. In addition to the Apache Cordova plugins and the plugins provided in the HTML5 SDK, these mobile frameworks provide APIs are useful for creating HTML5 apps. There is one caveat about using mobile frameworks: you may find that some frameworks or features don't function on the QNX SDK for Apps and Media, because the target board doesn't support the feature. For example, the Vibrate API used on many phones won't work on your target unless the hardware supports vibration.

The process for creating an HTML5 app is similar to mobile development. You can use an iterative development approach to add functionality, testing as you go along. The following diagram shows the main development workflow:

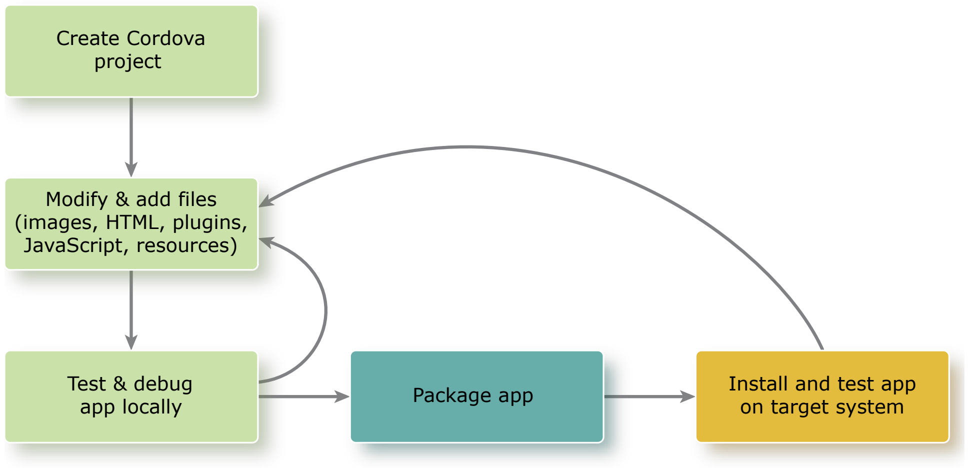

Create Cordova project
Modify & add files (images, HTML, plugins, JavaScript, resources)
Test & debug app locally
Package app
Install and test app on target system

Figure 2: The HTML5 app development workflow

For a walkthrough of the entire process, consider looking at "*Example: Hello World!* (p. 47)."

Creating a Cordova project

To start a Cordova project, use the *`create`* (p. 14) command to create a skeleton HTML5 project. For more information about the `create` command and the folder structure that it creates, see *Creating an HTML5 project* (p. 30).

Modifying and adding files

After you create a Cordova project, you modify and add files to your project. You might add images, HTML files, plugins, and JavaScript.

Here are a few guidelines to follow during the development of your app:

- To ensure that apps function correctly, verify that the Cordova project has the proper permissions, and disable pinch and zoom. For more information about these tasks, see "*Creating an HTML5 project* (p. 30)."
- You can use Cordova plugins and plugins provided in the HTML5 SDK to incorporate functionality as required for your app. To manage any plugins that you want to use in your project, use Plugman. After you add a plugin to your project, it's bundled into the **.bar** file during the build process. For more information about managing plugins, see "*Managing plugins* (p. 13)."
- If you need to access native functionality, you may find it useful to create your own custom plugin. For more information about creating custom plugins, see *Creating Custom Cordova Plugins* (p. 71). If you require access to PPS objects or want to limit permissions, look at the example code in the "*Example: Adding a permission to access a PPS object* (p. 50)."
- Modify the **config.xml** configuration file to tell the framework which HTML file to launch your app with, configure your app's behavior, and list plugins you have added to the Cordova project. For more information about modifying the **config.xml** file, see "*Working with the configuration file* (p. 32)."
- As you develop and test your app, ensure that your app performs and functions well. If you find that you need to improve performance, follow the guidelines at *Improving HTML5 app performance* (p. 70).

Testing and debugging locally

The HTML5 standard ensures compatibility between browsers, making the environment compatible with mobile, desktop, and web environments. Therefore, you can test an HTML5 app locally using a web browser. If you are using a WebKit-based browser, you can debug your app using Web Inspector. For more information about using Web Inspector and debugging apps, see *Debugging Web Apps* (p. 53).

The testing you do is useful to test interactions and transitions in your app. You can test functionality that you incorporate from JavaScript libraries (provided your browser supports the plugin). To run an app, select the starting HTML file for your app and run it in a web browser on your host computer. It's important to recognize that some functionality may not work because:

- it is native functionality that's only available on the target
- it requires interaction with PPS objects
- the functionality is hardware-specific (e.g. camera)

For example, any functionality that requires communication with PPS objects or calls to Cordova JavaScript Plugins won't work in your local browser because these require a target that runs Apps and Media. For this reason, your final testing should be on a target or VMware image running Apps and Media.

Packaging the app

To fully test functionality (especially native functionality that your HTML5 app uses), use the `build` command to package your HTML5 app into an archive called a **.bar** file. For more information about packaging a Cordova app, see *Packaging an HTML5 app* (p. 45).

Installing and testing app on target system

After you build a **.bar** file, transfer the **.bar** file to your target to install, launch, and test your app. You should ensure that you do final testing on a target board or VMware running Apps and Media—especially if you need to test native functionality or functionality specific to Apps and Media.

For more information about installing, launching, stopping, and uninstalling apps, see the *Application and Window Management* guide.

If you need to debug your application, you can use a WebKit-based browser and connect to your target. For more information about debugging an app on a target board, see *Debugging Web Apps* (p. 53).

Creating an HTML5 project

You create an HTML5 app through an HTML5 project. When you create an HTML5 project, you first create a skeleton project, then look after the configuration to ensure that your HTML5 app will work correctly on your target system.

The process for creating HTML5 apps is similar to the process for creating Cordova apps. You create the project with the default folders and files, then add and modify files for your scripts, HTML5, images, etc., and modify the **config.xml** file to configure the app's behavior.

To create an HTML5 project, use the *`create`* (p. 14) command. This command creates a skeleton project and pre-populates the files required to run an HTML5 app on the platform. If you build the app without modifications, by default, the app is set up to handle the `deviceready` event, and displays the **index.html** page with the Cordova logo.

An HTML5 application created with Apache Cordova is a standalone app, which means you aren't required to point to a remote server to load a webpage. The app runs on the target and can run in a standalone fashion.

Configuring apps for QNX SDK for Apps and Media

Your HTML5 app doesn't run on a mobile device, such as a tablet or a smartphone. Instead, it runs in an HMI on your target board. Therefore, you must make the following changes to your app so that it functions correctly:

- Disable pinch and zoom on the HTML5 app. This incorrect resizing can cause your app to behave incorrectly. In the **index.html** file (or other starting page for your app), set the `height` and `width` to match your screen dimensions, and set the `user-scaleable` to `no` as shown here:

  ```
  <meta name="viewport" content="width=800, height=480,
        initial-scale=1.0, user-scalable=no"/>
  ```

- Whitelist any required URIs, such as URLs or network locations, that aren't local to the app. To whitelist a URI, use the *<access>* (p. 33) element in the **config.xml** file. For example:

  ```
  <access subdomains="true" origin="www.qnx.com" >
  <access subdomains="true" origin="file:///SDCard" />
  ```

  For improved security, it's best to whitelist only the locations you require and use HTTPS for sensitive information. For more information about whitelisting, see the *Whitelist Guide* in the *Cordova Documentation*.

Running Cordova `create`

To run the Cordova `create` command, navigate to the **`[location you installed HTML5 SDK]`/html5sdk/cordova/cordova-qnxcar/bin** folder on your host computer. You run the command with the following parameters:

```
create project_path package_name app_ID
```

where:

- *project_path* — The location of an empty directory.
- *package_name* — (Optional) A reverse domain name, such as `com.company.appname`.
- *app_ID* — (Optional) A unique name. The name should be unique on the target system.

For example:

```
create hello com.example.hello helloWorld
```

Attention The *package_name* and *app_ID* must be alphanumeric characters and can't be more than 50 characters long.

Project folders

The `create` command creates the following subfolders in the folder you specified as the *project_path*:

www

This folder contains all your project resources, including the **index.html** file, which is the starting page for your app. It also contains subfolders for various resources, including CSS and JavaScript files, as well as any other resources you might add.

When you start to add code and resources to your app, save any project resource files used by the app in the **/www** folder (or in the relevant subfolder within **/www**).

native

The location where plugins native to the platform exist. Each HTML5 project includes plugins required for target boards and VMware images.

lib

The Cordova Framework JavaScript libraries required to run on the target.

cordova

The tools to build and work with the project:

`build`

Interprets your HTML5 and JavaScript code, builds the binaries, and packages the web assets to a **.bar** file. For more details, see "*Test and package* (p. 28)."

`clean`

Removes any files, such **.bar** files, that the `build` command created.

`version`

Returns the version of the Cordova that's being used for your HTML5 app.

`whereis.cmd`

Searches the directories in your *PATH* environment to locate the file that you typed. This command is available only for Windows.

Working with the configuration file

The configuration file is a platform-agnostic XML file that contains information about your HTML5 app.

The configuration (**config.xml**) file is located in your HTML5 project's **www** folder. This file is used to control many aspects of an app's behavior. The **config.xml** file is based on the Cordova's **config.xml**, but the HTML5 SDK provides additional elements to support specific app behavior. If you are familiar with working with *BlackBerry Webworks*, you may notice some of the element names look familiar, but it isn't a one-to-one mapping and functionality differs slightly in the HTML5 SDK.

During the build process, the contents of the **config.xml** are used to generate the **bar-descriptor.xml** file. It's common to modify the **config.xml** to add permissions for an app. There are default permissions available you can use, but you can also create your own custom permission. A custom permission allows you to control access to resources and Persistent Publish/Subscribe(PPS) objects that you create. For more information about creating custom permissions, see *Example: Adding a permission to access a PPS object* (p. 50).

The following table lists the elements that are applicable for **config.xml** when you use the HTML5 SDK.

Element	Description
`<access>` (p. 33)	Specifies that the app can access external network resources. If you do not specify an `<access>` element, the app can access only local resources. Local access includes the resources packaged with the app in the **.bar** file.
`<author>` (p. 34)	Specifies the name (typically the company or developer name).
`<config-file>` (p. 35)	Specifies configuration settings to add to a configuration file.
`<content>` (p. 36)	Specifies the start page that the Cordova app uses when it runs. The start page can contain the web address of a file that is located outside of the application archive.
`<description>` (p. 37)	Specifies a human-readable description for the Cordova app.
`<feature>` (p. 37)	Specifies the plugins to use with your app.
`<name>` (p. 39)	Specifies a human-readable name for the Cordova app.
`<icon>` (p. 38)	Specifies that image that appears on the screen. The image should be 86x86 pixels.
`<rim:permissions>` (p. 40)	Specifies permission access to various features.
`<rim:permit>` (p. 41)	Specifies permission access to various features. The list of features is available in the **sys.acl** and **sys.res** files on your target system. For more information, see “Authorization Manager” in the *System Services Reference*.
`<widget>` (p. 42)	The top-level parent component and follows the Cordova specification. See *Project Settings* in the *Cordova Documentation*

<access>

Specifies the network resources that the application can access. Use this element to specify cross-domain URIs which your application requires but which are not local.

Syntax:

```
<access origin="string" subdomains=["true" | "false"] />
```

Description:

The `<access>` element specifies that a Cordova app can access external network resources. By default, if you do not specify an `<access>` element, an app has access to local resources only. Local resources include all resources packaged in the app's **.bar** file.

When you specify more than one `<access>` element, the most specific definition is used. For example, if you use `http://somedomain.com` and `http://specific.somedomain.com`, the `<access>` element that uses the first definition (and any features defined under it) is ignored.

As a good practice, use HTTPS to expose sensitive APIs and to protect your communication channel. For more information about whitelisting, see the *Whitelist Guide* in the *Cordova Documentation*.

Occurrences:

Zero or more.

Parent elements:

`<widget>` (p. 42)

Child elements:

None.

Content:

None.

Attributes:

You can define the following attributes for this element:

Attribute	Description
`origin`	Optional. The `origin` attribute defines the web address for the access request. For domains that access data through `XMLHttpRequest`, explicitly specify the domain for the `origin` attribute. For domains that don't access content through `XMLHttpRequest`, you can specify a wildcard (`*`) for the `origin` attribute to whitelist any domain.

Attribute	Description
`uri`	Deprecated. Instead, use the `origin` attribute. The `uri` attribute defines the web address for the access request. This attribute is supported only for backwards compatibility.
`subdomains`	Optional. The `subdomains` attribute is a Boolean value that specifies whether the host component in the access request applies to subdomains of the domain specified by the `origin` attribute. By default, if you don't specify the value of the `subdomains` attribute, the value is set to `false` and no access to subdomains is requested.

Examples:

The following example shows how to white-list an external resource, in this case, the domain `somedomain`, as well as any subdomains it might have.

```
<access origin="https://somedomain.com" subdomains="true"/>
```

The following example shows how to white-list a URI to mountpoint named **SDCard**:

```
<access subdomains="true" origin="file:///SDCard" />
```

<author>

Specifies the name of the creator of the app.

Syntax:

```
<author>
    string
</author>
```

Description:

The `<author>` element specifies the name of the app's creator (typically a developer or a company).

Occurrences:

One.

Parent elements:

`<widget>` (p. 42)

Child elements:

None.

Content:

A string value representing the name of the person or organization that created the app.

Example:

```
<author>My Corp</author>
```

<config-file>

The `<config-file>` specifies where child elements should be placed.

Syntax:

```
<config-file target="bar-descriptor.xml" parent="/qnx">
    <category>media</category>
</config-file>
```

Description:

The `<config-file>` specifies the configuration file and XPath where child elements should be placed.

Add the `<config-file>` element's child elements to the specified configuration file. The location where elements should be added is specified using an XPath value, such as the first node of the file. For example, in the configuration file called **bar-descriptor.xml**, the first node is called `qnx`.

Occurrences:

Zero or more.

Parent elements:

`<widget>` (p. 42)

Child elements:

Any elements that you want to add after the specified `parent` node.

Content:

None.

Attributes:

You can define the following attributes for this element:

Attribute	Description
`target`	The name of the configuration file.

Attribute	Description
`parent`	The parent node where to put the configuration specified by child elements. You can specify the location using XPath.

Examples:

The following example shows how to add a category element to the `qnx` node in the **bar-descriptor.xml** file.

```
<config-file target="bar-descriptor.xml" parent="/qnx">
        <category>media</category>
</config-file>
```

<content>

Specifies the first page to load when the app starts.

Syntax:

```
<content src="string" />
```

Description:

The `<content>` element specifies the first page the HTML5 app displays when it runs. The start page can contain the web address of a file that is located outside of the application archive.

Occurrences:

One.

Parent elements:

`<widget>` (p. 42)

Child elements:

None.

Content:

None.

Attributes:

You can define the following attributes for this element:

Attribute	Description
`src`	Required. The `src` attribute specifies the source HTML file in the application archive.

Example:

The following example shows how to specify a start page called **startpage.html**.

```
<content src="index.html" />
```

<description>

A human-readable description of the HTML5 application.

Syntax:

```
<description>string</description>
```

Description:

The `<description>` element specifies a human-readable description for an HTML5 app.

Occurrences:

One.

Parent elements:

`<widget>` (p. 42)

Child elements:

None.

Content:

A string value representing a human-readable description of the app.

Example:

```
<description>
   This app displays "Hello World" on the screen.
</description>
```

<feature>

The names of the plugins to be included with the HTML5 app.

Syntax:

```
<feature  name="string" value="string"/>
```

Description:

The `<feature>` element specifies the names of the plugins to include with the HTML5 app.

Occurrences:

Zero or more.

Parent elements:

`<widget>` (p. 42)

Child elements:

None.

Content:

A string value.

Attributes:

You can define the following attributes for this element:

Attribute	Description
`name`	The name of the feature.
`value`	The plugin ID, as defined in its **plug.xml** file.

Example:

```
<feature name="com.qnx.demo" value="com.qnx.demo" />
```

<icon>

Specify a custom icon for an HTML5 app.

Syntax:

```
<icon src="string" />
```

Description:

The icon specified by the `<icon>` element's `src` attribute identifies your app on the Home screen of your target system.

The specified icon must meet the requirements for custom or default icons. The icon image should be 24-bit PNG format with an alpha channel. If you use the icon on the HMI (Home Screen), it should be 86x86 pixels in size. If you don't specify an icon for your app, the app appears with only the text specified by its `<name>` element.

Occurrences:

Zero or more.

Parent elements:

`<widget>` (p. 42)

Child elements:

None.

Content:

None.

Attributes:

You can define the following attributes for this element:

Attribute	Description
`src`	Specifies the path for an image file packaged in the **.bar** file. This attribute is required.

Example:

In the **config.xml** file, we include the following entries:

```
<icon src="icon-86x86.png" />
```

<name>

The human-readable name of the HTML5 application.

Syntax:

```
<name>string</name>
```

Description:

The `<name>` element specifies a human-readable name for an HTML5 app. You can use this name in, for example, an application menu.

If you don't specify a `name` element, the app is not valid. The name must be 25 characters or less.

Occurrences:

One.

Parent elements:

<widget> (p. 42)

Child elements:

None.

Content:

A string value representing a human-readable name for the HTML5 app.

Example:

```
<name>Hello World! app</name>
```

<rim:permissions>

This element is a container element for `<rim:permit>` element, which specifies the permissions for HTML5 app features.

Syntax:

```
<rim:permissions>
    <rim:permit>permission_string</rim:permit>
</rim:permissions>
```

Description:

The `<rim:permissions>` element is a container element for the `<rim:permit>` element, which specifies the permissions for various features in an HTML5 app.

See the *<rim:permit>* (p. 41) element for information about permission features.

To use the `<rim:permissions>` element, you must include the following namespace declaration in the parent `<widget>` element:

```
xmlns:rim="http://www.blackberry.com/ns/widgets">
```

Occurrences:

One or none.

Parent elements:

<widget> (p. 42)

Child elements:

<rim:permit> (p. 41)

Content:

None.

Attributes:

None.

Example:

The following example demonstrates how to set a custom permission called `access_demo`.

```
<widget xmlns:rim="http://www.blackberry.com/ns/widgets">
...
...
  <rim:permissions>
     <rim:permit>access_demo</rim:permit>
      </rim:permissions>
  </platform>
<widget>
```

<rim:permit>

Specifies the permission for specific HTML5 app features.

Syntax:

```
<rim:permissions>
    <rim:permit>permission_string</rim:permit>
</rim:permissions>
```

Description:

The `<rim:permit>` element specifies permission access to various features in a HTML5 app.

Occurrences:

One or more.

Parent elements:

`<rim:permissions>` (p. 40)

Child elements:

None.

Content:

A string value representing a valid permission value. For a list of permissions and their descriptions, see Authorization Manager.

Attributes:

None.

Permissions:

The permissions (or capabilities) you can use are defined using the Authorization Manager (`authman`). For information about the available permissions, see Authorization Manager in the *System Services Reference*.

Example:

The following example shows how to give an app access to geolocation.

```
<widget xmlns:rim="http://www.blackberry.com/ns/widgets">
...
...
    <rim:permissions>
        <rim:permit>read_geolocation</rim:permit>
    </rim:permissions>
...
...
<widget>
```

<widget>

Specifies the root element for the **config.xml** file.

Syntax:

```
<widget xmlns="http://www.w3.org/ns/widgets"
        xmlns:rim="http://www.blackberry.com/ns/widgets"
        version="string"
        id="string"
        xml:lang="string"
        rim:header="string"
        rim:userAgent="string">
</widget>
```

Description:

The `<widget>` element is the root element of the **config.xml** file. The **config.xml** file must contain a single instance of `<widget>`.

Occurrences:

One.

Parent elements:

None.

Child elements:

Name	Occurrences
`<access>` (p. 33)	zero or more
`<author>` (p. 34)	one
`<config-file>` (p. 35)	zero or more
`<content>` (p. 36)	one
`<description>` (p. 37)	zero or more
`<feature>` (p. 37)	zero or more
`<icon>` (p. 38)	one
`<name>` (p. 39)	one
`<rim:permissions>` (p. 40)	zero or more
`<rim:permit>` (p. 41)	zero or more

Content:

None.

Attributes:

You can define the following attributes for this element:

Attribute	Description
`xmlns`	Required. Defines the namespace for the HTML5 app. The value must be `xmlns="http://www.w3.org/ns/widgets"`. If this namespace is missing, the **.bar** file isn't valid.
`xmlns:rim`	Required. Defines the namespace for the Cordova extensions (that is, those elements with the `rim:` prefix). The value must be `xmlns:rim="http://www.blackberry.com/ns/widgets"`.
`version`	Required. Specifies a valid version for the app in one of the following formats: • X.X.X • X.X.X.X If you specify a version number that's not valid, the **.bar** file isn't valid.

Attribute	Description
`id`	Required. Specifies a unique identifier for the app. Unless you are repackaging an app from another platform, use a reverse DNS format (e.g., `id="com.somedomain.HelloWorld"`).
`xml:lang`	Optional. Specifies the language that is used in the element. For more information about this attribute, visit *www.w3.org/TR/html401/struct/dirlang.html*.

Example:

```
<widget xmlns="http://www.w3.org/ns/widgets"
        xmlns:rim="http://www.blackberry.com/ns/widgets"
        version="1.0.0.1"
        id="com.sample.HelloWorld">
</widget>
```

Packaging an HTML5 app

To run an HTML5 app on a target running QNX Neutrino, you must package it as a **.bar** file.

Build options

The `build` command interprets your HTML5 and JavaScript code, builds the binaries, and packages the web assets into a **.bar** file.

You can use the following options with the `build` command:

`--debug`

Build in debug mode. This option allows Web Inspector to connect to your application on the target for debugging purposes and console logs to appear.

`--release`

Build in release mode and signs the app if signing keys are provided. This version of the QNX SDK for Apps and Media doesn't support signing. Even though the tools support signing, it isn't necessary to sign your app or provide signing keys for apps running on this release.

`--web-inspector`

Enables web-inspector for debugging apps on the target. This option is included when you use the `--debug` option.

It's a good idea to use the debug option when you are testing, because it allows you to run Web Inspector.

Building the app

To create a **.bar** file for an HTML5 app, run the `build` command in your HTML5 project folder. The **.bar** file can be deployed to your target using these steps:

1. Navigate to the folder where your HTML5 project was created when you ran the `create` command.
2. Ensure that all the resources and permissions your application needs are specified in the **config.xml** file.

   The **create** command produces a basic **config.xml** file that defines the app's properties, features and functionality, and behavior. You may need to edit this file depending on the features or permissions required by your app. For more information, see "*Working with the configuration file* (p. 32)."

3. Navigate to the **cordova** folder.
4. Run the `build` command:

```
build --debug
```

5. The default name of the **.bar** file is **qnxcarapp.bar**. If you will work with multiple **.bar** files on a target system, rename the file to a unique name.

The output from an HTML5 app build looks like the following. The warnings about not having a debug token when you build is expected because debug tokens aren't used for apps on this platform:

```
[WARN]    Using legacy signing keys
[INFO]    Populating application source
[INFO]    Parsing config.xml
[INFO]    Generating output files
[WARN]    Failed to find debug token. If you have an existing debug token,
          please copy it to C:\Users\qnxuser\.cordova\blackberry10debugtoken.bar.
          To generate a new debug token, execute the 'run' command.
[INFO]    Package created:
          C:\AppsMedia_html5sdk_141022_1646\html5sdk\CordovaPPSdemo\build\simulator\qnxcarapp.bar
[WARN]    Failed to find debug token. If you have an existing debug token, please
          copy it to C:\Users\qnxuser\.cordova\blackberry10debugtoken.bar.
          To generate a new debug token, execute the 'run' command.
[INFO]    Package created:
          C:\AppsMedia_html5sdk_141022_1646\html5sdk\CordovaPPSdemo\build\device\qnxcarapp.bar
[INFO]    BAR packaging complete
```

After the `build` command is finished, the **.bar** file is found in the following folders:

- ***location of your HTML5 project*/build/device**
- ***location of your HTML5 project*/build/simulator**

You can use copy the file to a USB stick and mount it to your target board or use a tool to transfer the file to the target. If you choose the latter, your target must be connected to the same network as your computer. You must get your file to the target to deploy it.

For information about creating a simple HTML5 app using Cordova, see "*Creating an HTML5 project* (p. 30)." For information about installing the app on your target, see "Installing packaged apps on the target" in *Application and Window Management*.

Example: Hello World!

This example walks you through the end-to-end process you follow when using the HTML5 SDK tools to create, edit, and package and application, then install and run it on a target.

To create a basic HTML5 app template:

1. In a command prompt, navigate to the `[location where you installed the HTML5 SDK]`**/html5sdk/cordova/cordova-qnxcar/bin** directory.
2. Run the Cordova `create` command to create an HTML5 project. For more information about using the `create` command, see *Creating an HTML5 project* (p. 30). For example, on a Windows machine, type the following in the command-line prompt.

   On Windows:

   create ***[directory to put your project]\myfirstapp myfirstapp com.mycompany.myfirstapp***

   Linux:

   create ***[directory to put your project]/myfirstapp myfirstapp com.mycompany.myfirstapp***

3. Navigate to your project directory specified in the previous step, edit the app's start page, and change the existing `<meta name="viewport"...>` line to the following line to prevent pinch and zoom on the app. Usually the start page is the **www/index.html** file.

   ```
   <meta name="viewport" content="width=800, height=480,
         initial-scale=1.0, user-scalable=no"/>
   ```

4. Below the `<title>Hello Word</title>` line, add an event to display simple text on the *onReady()* event occurs. For example, add the following code:

   ```
   <script type="text/javascript">
               function onReady() {
                   document.body.innerHTML = "HELLO AWESOME HTML5 WORLD!";
               }
               window.addEventListener("load", function (e) {
                   if (window.cordova) {
                       document.addEventListener("deviceready", onReady);
                   }
               }, false);
    </script>
   ```

5. (Optional) Navigate to the parent directory of your project folder and add any additional Cordova plugins to your project using the `plugman install` command for any features you want. The `Plugman` command is required to manage plugins for your project. For more information, see "*Managing plugins* (p. 13)." For example, add the provided demo project plugin.

On Windows:

```
plugman install --platform blackberry10
                --project .\myfirstapp --plugin .\demo\com.qnx.demo
```

On Linux:

```
plugman install --platform blackberry10
                --project ./myfirstapp --plugin ./demo/com.qnx.demo
```

6. Modify the **config.xml** to configure app behavior and add access permissions. In the **config.xml** file, add the following text. You can also put a human-readable string (e.g., `myfirstapp`) for the `<name>` element. The string that you provided is what is shown on the screen when the HMI loads your app. If you added the plugin above, the plugin should be added as a `<feature>` entry in your **config.xml**:

```
<name>myfirstapp</name>
...
<access subdomains="true" origin="www.qnx.com" >
<access subdomains="true" origin="file:///SDCard" />
```

7. As you code and modify your project, you can build your project. To build, run the `build` command from the **cordova** folder in your project. Optionally, when you build use the `debug` option:

   To build using debug command: `build debug`

   To build without the debug command: `build`.

   The generated **.bar** file for a target is located at:

   On Windows:

   `[location of project]`\build\device\qnxcarapp.bar

   On Linux:

   `[location of project]`/build/device/qnxcarapp.bar

   To find **.bar** files to install on VMware images, look in the **`[location of project]`/build/simulator** directory .

8. Transfer the **.bar** file to your target and install it using one of these options:

   - You can use a USB stick to deploy the file to target. Copy the **.bar** file from the previous step to your USB stick, and insert it into the target. Then, on your target, run the following command to deploy from your USB stick:

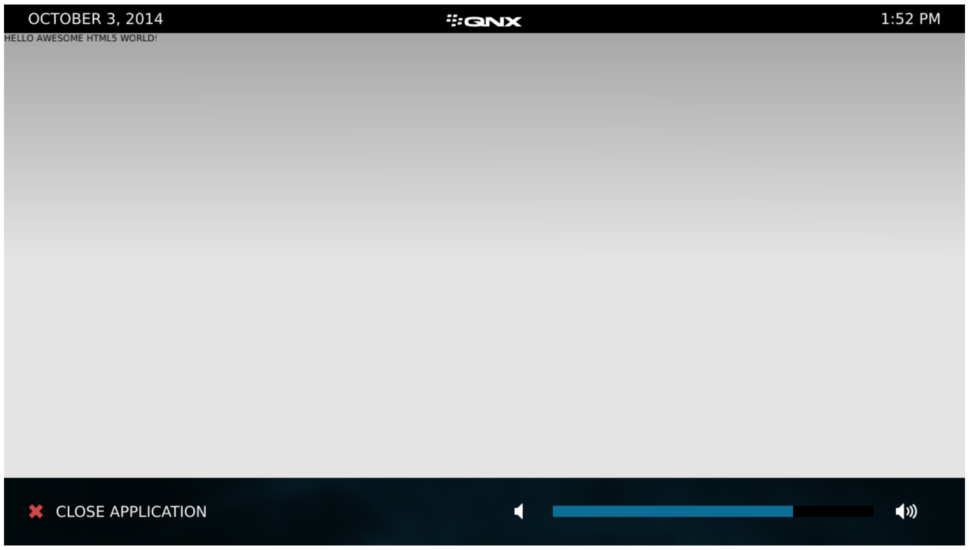
   ```
   #bar-install /fs/usb0/qnxcarapp.bar.
   ```

- Use a tool to transfer the **.bar** file to the **\tmp** folder on the target. You can use FTP or SCP if your target is connected to same network as your host computer. Then, on your target, run the following command:

  ```
  # bar-install /tmp/qnxcarapp.bar
  ```

9. If your target has an HMI, then you should see the app appear. To run it, tap the icon representing your app. Otherwise, you can launch the app from command line using the `launch` command:

   ```
   # launch myfirstapp
   ```



HELLO AWESOME HTML5 WORLD!

Figure 3: A screen showing the text output by the HTML5 "Hello World!" program

The `create` command generates the folder structure for your project at the specified location (for the example above, the **hello** directory). For more information about the `create` command, see *Creating an HTML5 project* (p. 30).

Tip

The app is essentially an arrangement of web assets that are packaged into a container. You can use a browser to view your app.

If you find that you require functionality not available, such as accessing a PPS object you created, you can build a custom Cordova plugin. For information about creating a simple Cordova plugin that uses PPS, see "*Creating Custom Cordova Plugins* (p. 71)."

Example: Adding a permission to access a PPS object

You can use Authorization Manager (`authman`) to restrict access to Persistent Publish/Subscribe (PPS) objects.

Permissions and capabilities

As well as letting you persist information across device restarts, PPS is useful for sharing information between apps in a secure manner. You can use the Authorization Manager (`authman`) to restrict access to a PPS object and to limit the data one more multiple apps can access.

To create a permission, you edit the **pps.conf** file to create a *capability*. You can then configure access and restrictions to the resources and services for that new capability in the **/etc/authman/sys.acl** and **/etc/authman/sys.res** files.

After you have created a capability, use the `<rim:permit>` to specify the name of the capability in the **config.xml** in your HTML5 project. The `<rim:permit>` element in the **config.xml** file grants the HTML5 app access to the resources and services defined by the capability.

Before you begin, you should have a good understanding of `authman`, of PPS, and how to use PPS with an HTML5 app.

- For information about `authman`, see Authorization Manager in the *System Services Reference*.
- For more information about the **pps.conf** file, see "ACL configuration file format" in the *Persistent Publish/Subscribe Developer's Guide*.
- For information about using PPS with HTML5, you can look at the Cordova PPS Demo sample code and see "*Using the PPS interface* (p. 74)".

Setting a PPS object capability

The instructions below show you how to:

- use `authman` to create and configure access to a capability called `access_demo`
- configure the system to define a new PPS object
- grant access to your HTML5 project to use the new the capability

To perform these tasks:

1. Modify the **config.xml** in your HTML5 project to grant access to a new permission called `access_demo`. To do this, add the `<rim:permit>` element beneath a `<rim:permissions>` element. The capability `access_demo` is defined in the next step. Here's how your `config.xml` might look after you have made this addition (shown in **bold**):

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="CordovaPPSdemo" version="1.0.0.0"
        xmlns="http://www.w3.org/ns/widgets"
        xmlns:rim="http://www.blackberry.com/ns/widgets">
    <name>Cordova PPS Demo</name>
    <author>QNX</author>
    <description>Cordova PPS  Demo</description>
    <content src="index.html" />
    <rim:permissions>
        <rim:permit>access_shared</rim:permit>
        <rim:permit>access_internet</rim:permit>
        <rim:permit>run_native</rim:permit>
        <rim:permit>access_demo</rim:permit>
    </rim:permissions>
    <feature name="com.qnx.demo" />
    <feature name="com.qnx.demo" value="com.qnx.demo" />
</widget>
```

2. On the target system, define the PPS object to the **/etc/pps.conf** file. This file lists the PPS objects to create. For example, to create a PPS object called `qnx/demo` and provide read-write access for everybody to the object, you add the following entries to the **pps.conf** file:

```
qnx/demo
    0:0:0660:O_CREAT
    user::rw
    group::rw
    other::rw
    mask::rw
```

3. On your target, create the capability. For example, to create the `access_demo` capability, add the name of the capability, and then list the PPS objects that are accessible in the **/etc/authman/sys.acl** file.

```
access_demo
    ACL opt rwx:rw /pps/qnx/
    ACL opt rwx:rw /pps/qnx/demo
```

4. On your target, configure the restrictions for the new capability in the **/etc/authman/sys.res** file. In this scenario, one app called CordovaPPSdemo is configured to access the PPS object. To restrict other apps from accessing the PPS object, but allow the CordovaPPSdemo app to access the PPS object, add the name of the capability, `deny *`, and `allow [name of app]` entries to the **/etc/authman/sys.res** file. For example:

```
access_demo
    deny *
    allow CordovaPPSdemo.testDev_dovaPPSdemod339185a
```

The string "CordovaPPSdemo.testDev_dovaPPSdemod339185a" is determined when you build your app using the `build debug` command. The name is a combination of the package name and the package identifier. There are two ways to determine the string to use.

- You can see the string that appears in the message after you use the `bar-install` command to install your **.bar** file.
- You can unarchive the **.bar** file, then determine the string that's used from the **META-INF/MANIFEST.MF** file based on this algorithm:

  ```
  Package-Name+ '.' + Package-Id
  ```

  For example, based on the following **MANIFEST.MF** file, the name would be **CordovaPPSDemo.testDev_dovaPPSdemod339185a**:

  ```
  Archive-Manifest-Version: 1.5
  Archive-Created-By: BlackBerry WebKit BAR Packager 1.10

  Package-Type: application
  Package-Author: QNX
  Package-Author-Id: testUU5YICAgICAgICAgICAgICA
  Package-Name: CordovaPPSdemo
  Package-Id: testDev_dovaPPSdemod339185a
  Package-Version: 1.0.0.0
  Package-Version-Id: testMS4wLjAuMCAgICAgICAgICA
  ...
  ...
  ```

After the CordovaPPSdemo app is deployed, no other app may access the PPS object named **qnx/demo**. If you want another app to access this PPS object, add the generated name of the app to **/etc/authman.sys** using another `allow` entry as described above.

Debugging Web Apps

Included as part of WebKit, *Web Inspector* is a useful debugging and profiling tool for web content.

You can use Web Inspector to troubleshoot and optimize your web content for your apps. This tool includes various features and capabilities, such as inspection, profiling, and console integration.

WebKit-enabled browsers use a client-server architecture to make Web Inspector functionality available. They act as webservers, so you can inspect content remotely on a browser. You can inspect your app's code by using any WebKit-based browser on the same Wi-Fi network as your app. Simply launch Web Inspector and navigate to the IP address and port number used by the browser.

Enabling Web Inspector

Web Inspector functionality is disabled. You can enable it when you compile your HTML5 app.

To use Web Inspector in the browser, you enable it in the browser options. For a Cordova or an HTML5 application, you enable Web Inspector by specifying a command-line flag at compile time.

To begin debugging your app, you can install the Google Chrome browser or another WebKit browser for your platform. Once Web Inspector is enabled, you can access it using a supported device with the browser.

To enable Web Inspector for your app:

1. Navigate to your **cordova** apps directory.
2. Run the following command:

   ```
   build debug
   ```

   To launch your compiled app, you need to know the port number your app is running on. You can use `netstat -a` on your target to see the open ports. The listening port numbers start at 1337. For example:

   ```
   # netstat -a
   Active Internet connections (including servers)
   Proto Recv-Q Send-Q  Local Address          Foreign Address        State
   tcp        0      0  *.1337                 *.*                    LISTEN
   tcp        0      0  *.ssh                  *.*                    LISTEN
   tcp        0      0  *.telnet               *.*                    LISTEN
   tcp        0      0  *.8000                 *.*                    LISTEN
   udp        0      0  *.*                    *.*
   udp        0      0  *.bootpc               *.*
   Active Internet6 connections (including servers)
   Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
   tcp6       0      0  *.ssh                  *.*                    LISTEN
   Active UNIX domain sockets
   Address  Type   Recv-Q Send-Q    Inode      Conn     Refs  Nextref Addr
    81556d8 stream      0      0        0  8155680        0        0
    8155680 stream      0      0        0  81556d8        0        0
   ```

Launching Web Inspector

To use Web Inspector, you must launch and manually connect it.

Before you begin to use Web Inspector, verify that:

- your computer is connected to your target board via Ethernet
- your computer has a WebKit-based desktop browser (e.g. Google Chrome or Apple Safari)
- you have launched your application on the target (e.g. by manually starting **weblauncher**, or by sending a command to the launcher's PPS object via a command console).

After you've launched the app you want to inspect, you'll need to manually connect Web Inspector.

To launch Web Inspector to begin inspecting web content:

1. On your host computer, open your WebKit-based browser.
2. In the address bar, type the IP address of the device, specifying the port number used by the application serving the content. Note that different instances of the browser will be at different ports.

   After you complete this step, you will be prompted with a list of page titles for content that Web Inspector has loaded into memory (e.g. browser content or an HTML5 app).
3. Click any of the page titles to begin using Web Inspector to debug and profile your web content. Web Inspector opens and displays the **Elements** panel.

In the **Sources** view, after you enable debugging, to ensure that you have the latest source files in the list you can use the shortcut key (**CTRL+R**) to refresh the current page.

In the **Console** view, you can browse objects, test extensions, and test APIs without having to build and deploy your app.

Debugging and profiling using Web Inspector

Web Inspector contains a number of panels that provide different functionality you can use to help improve the appearance and performance of your webpage.

Using the WebKit-based browser on your desktop, you can navigate to the IP address and port number used by the server application and begin inspecting the code. You can use Web Inspector to manipulate the Document Object Model (DOM), edit and debug JavaScript code, analyze resource requests, and audit the performance of web content and web apps in near real time.

You can work from different panels, depending on what you need to do:

Panel	Description
Elements (p. 55)	Inspect the DOM of the current webpage and adjust settings for attributes and CSS properties. Changes you make are reflected in the browser.
Resources (p. 58)	Display information about all the resources used by the current webpage.
Network (p. 59)	Display information about each HTTP request made as resources are requested, received, and displayed in the browser or in your HTML5 app.

Panel	Description
Sources (p. 61)	Debug JavaScript code. You can set breakpoints and step through your code to locate and correct issues.
Timeline (p. 63)	View how much time it takes for the browser to load and render the webpage and its resources.
Profiles (p. 65)	Examine how your JavaScript code utilizes memory. With the Profiles panel, you can pinpoint programmatic inefficiencies.
Audits (p. 68)	Examine the network utilization and webpage performance. The tool suggests ways to improve performance.
Console	A command-line utility that lets you debug JavaScript or HTML parsing errors.

Optimizing layout and style

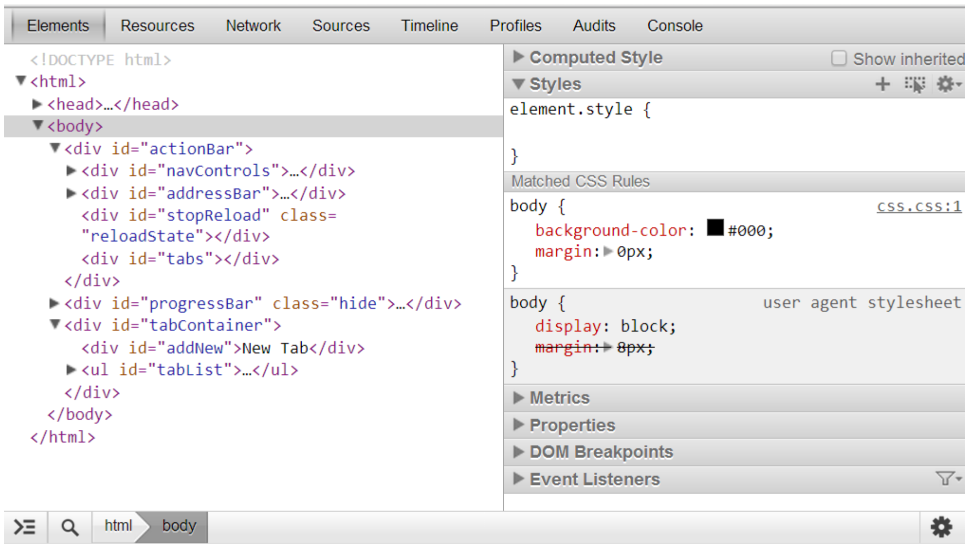
The **Elements** panel can be a very useful tool when you need to find the optimal layout for small screens.

The **Elements** panel allows you to view the DOM and trace style values for an element. By tracing style values, you can control where they are applied, how they have been inherited, and where they have been superseded. You can then adjust style settings to tweak the appearance of the webpage components to achieve the desired result.

The changes you make to the webpage are applied in near real time in the browser, so you can see how your changes affect the appearance of the content. Once you achieve the results you want, you can propagate the updated values into the source file.

The Elements panel

You can click the **Elements** icon on the toolbar to display the **Elements** panel. This panel is divided into two sections. On the left is the document pane, which displays the DOM tree of the HTML source document. On the right is a set of collapsible panes that display various pieces of information related to the element currently selected in the document pane.



```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div id="actionBar">
      ▶<div id="navControls">…</div>
      ▶<div id="addressBar">…</div>
       <div id="stopReload" class=
       "reloadState"></div>
       <div id="tabs"></div>
     </div>
    ▶<div id="progressBar" class="hide">…</div>
    ▼<div id="tabContainer">
       <div id="addNew">New Tab</div>
      ▶<ul id="tabList">…</ul>
     </div>
   </body>
</html>
```

```
element.style {

}
body {                                  css.css:1
   background-color: ■#000;
   margin:▶ 0px;
}
body {                     user agent stylesheet
   display: block;
   ~~margin:▶ 8px;~~
}
```

Figure 4: The Web Inspector Elements panel

Left pane

The pane of the **Elements** panel displays the DOM tree of the HTML source document. Each element is displayed as a separate node. You can expand the nodes of the DOM tree to view the children of a container element.

The document pane is a good tool to use to view the source of a page. Since the panel displays the page as a tree, the document is easy to view and to navigate, even when the original webpage is minified, or poorly formatted and difficult to read. Within the document pane, you can edit aspects of the DOM, such as attribute values or text.

Right pane

The right side of the **Elements** panel is a set of collapsible panes that display various pieces of information related to the element currently selected in the document pane. Some of these panes, such as **Computed Style** and **Event Listeners**, are informative; you use them simply to track information about the element. Other panes are editable and let you change the styles or properties associated with the selected element. You can edit content in the following panes:

- **Styles**: A pane that's divided into sections to show each matched CSS rule and the associated style declarations. It also shows style values that have been inherited. Inherited values that have been overwritten by other style declarations are shown with strikethough text.
- **Metrics**: A visual representation of the box model, which you can edit to optimize the layout of a container element on the screen. The box model refers to the amount of space a container element occupies in a rendered webpage. You can apply styles, such as margins, borders, and padding to an element to adjust the size of the content block and improve the page layout.
- **Properties**: Provides a pane that allows you to view the page as a collection of DOM objects with associated property values. Although some of the property values are editable, in most cases it's easier to edit style values in the **Styles** pane.

Inspect and modify element styles

Use the Web Inspector **Styles** pane to inspect and modify styles.

To inspect and modify styles:

1. Click the **Elements** icon on the toolbar to display the **Elements** panel.
2. In the document pane, locate and select the element with the style you want to modify.
3. In the data sidebar, expand the **Styles** pane to display the style declarations applied to the selected element.
4. Perform any of the following actions:
    - To change the value for a style declaration, double-click the value in the **Styles** pane to make the value editable, then type the new value. You can use the **Tab** key to cycle through the declarations within a selector to modify more than one value.
    - To disable a style declaration, deselect the adjacent check box.
    - To add a new style declaration for a selector, double-click the white space below the last style declaration, then type the new declaration.
    - To modify the selector, double-click the selector, then type the new selector value.
5. When you're done, propagate your changes to the source document.

Inspect and modify the DOM

Inspect and modify the DOM from the the Web Inspector **Document** pane.

To inspect and modify the DOM:

1. Click the **Elements** icon on the toolbar to display the **Elements** panel.
2. In the **Document** pane, navigate to the node that you want to view or change.
3. Perform any of the following actions:
    - To change the value of an attribute, double-click the value in the document pane to make the value editable, then type the new value. To modify more than one value, use the **Tab** key to cycle through each of the element's attributes.
    - To change an attribute name, double-click the attribute name and type the new value.
4. When you're done, propagate the changes to the source document.

Modify the box model for an element

Use the Web Inspector **Metrics** pane to inspect and the box model.

To inspect and modify the box model for an element:

1. Click the **Elements** icon on the toolbar to display the **Elements** panel.
2. In the **Elements** panel, expand the **Metrics** pane to display the *box model* associated with the selected element.
3. Click any of the top, bottom, left, or right values, then type the new value. The changed value is propagated to the associated style declaration in the **Styles** pane.
4. When you're done, propagate the changes to the source document.

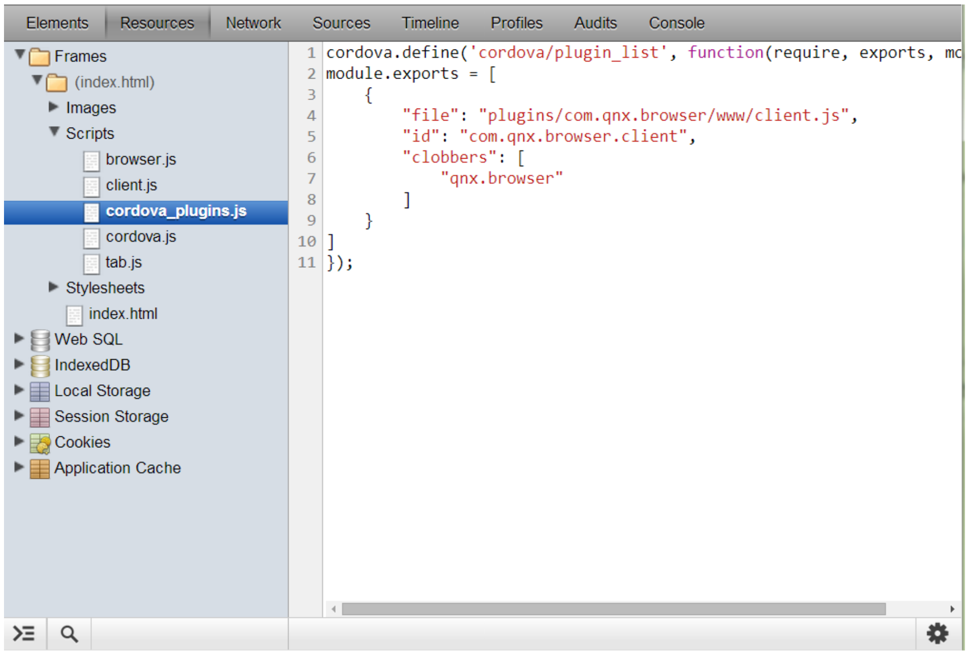
Analyzing page resources

The **Resources** panel allows you to view the complete set of resources that are loaded by a webpage.

You can use the Web Inspector **Resources** panel to view and interact with resources such as CSS and JavaScript, to check image content and information, and to view which font sets are used on the page. You can also view and interact with any client-side resources created by your page, including cookies, databases, storage objects, and application cache.

The Resources panel

You can click the **Resources** icon on the toolbar to display the **Resources** panel.



```
cordova.define('cordova/plugin_list', function(require, exports, mc
module.exports = [
    {
        "file": "plugins/com.qnx.browser/www/client.js",
        "id": "com.qnx.browser.client",
        "clobbers": [
            "qnx.browser"
        ]
    }
]
});
```

Figure 5: The Web Inspector Resources panel

The **Resources** panel shows a complete list of the resources that the WebKit engine must request and load to render the webpage, as well as any client-side resources created and used by the webpage. You can also use the **Resources** panel to view the content of any resource file. Resources are organized in the panel as follows:

- **Frames**: Contains the resources for each frame displayed in the content, including images, fonts, scripts, style sheets, and other content resources (e.g. embedded video or Flash files). Subframes within the main window are displayed as subfolders beneath the main Frames folder.
- **Databases**: Contains all the database tables that are associated with your content or app.
- **Local Storage**: Contains all *Local Storage* objects; that is, storage objects that persist after a browser session has ended.
- **Session Storage**: Contains all *Session Storage* objects; that is, storage objects that are valid only for the current browser session.
- **Cookies**: Contains all the cookies associated with the webpage or app.

- **Application Cache**: Contains the resources included in the manifest of an offline web application.

View resource content

Use the **Resources** panel to view the contents of page resources.

1. Click the **Resources** icon on the toolbar to display the **Resources** panel.
2. In the list in the left pane of the **Resources** panel, double-click a category to show the resources and subgroups. Continue to drill down until you locate the resource you want to view.
3. Double-click the resource in the left pane. The right pane shows the contents of that resource. For example, selecting an image resource displays the image itself, along with the file size and URL of the image file. Selecting a script or style sheet shows the content of that script or style sheet.

View resource network information

Use the **Network** panel to view resource information such as file size and transfer rate.

You can quickly see additional information about a specific resource by viewing the resource in the **Network** panel, which displays information such as file size and transfer rate information.

To view this information:

1. In the list in the left pane of the **Resources** panel, double-click a category to show the resources and subgroups. Continue to drill down until you locate the resource you're interested in.
2. Right-click the resource and click **Reveal in Network Panel**. Web Inspector opens the **Network** panel and briefly highlights the selected resource.

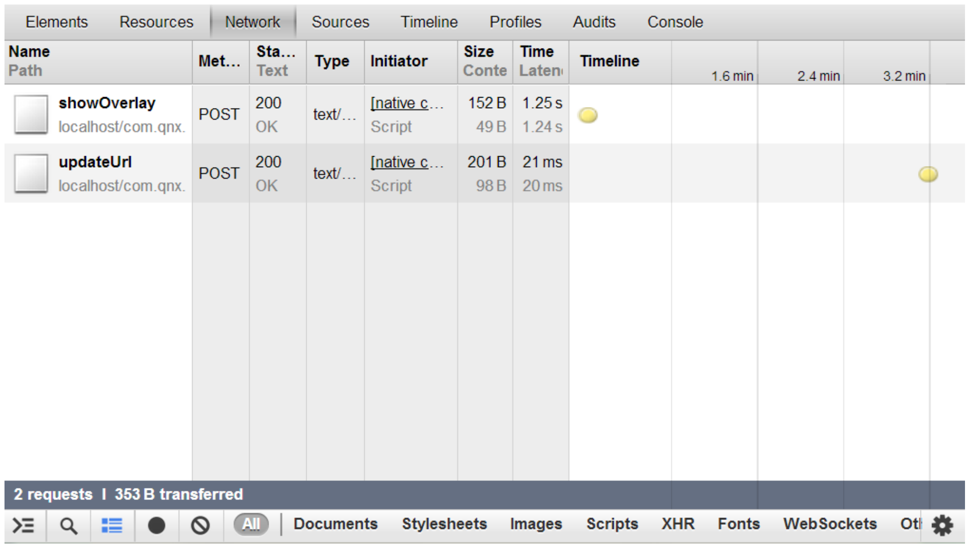
Analyzing network usage

The **Network panel** can show the time used to load resources.

The **Network** panel allows you to determine the network efficiency of your content. The panel displays information about each HTTP request made as the browser engine requests and downloads resources.

The Network panel

Click the **Network** icon in the toolbar to display the **Network** panel. Initially, the panel shows no information; you must reload the content on the device or simulator to allow Web Inspector to track the HTTP requests. When loading is complete, the **Network** panel displays a table.



Name / Path	Met…	Sta… / Text	Type	Initiator	Size / Conte	Time / Laten	Timeline
showOverlay / localhost/com.qnx.	POST	200 / OK	text/…	[native c… / Script	152 B / 49 B	1.25 s / 1.24 s	
updateUrl / localhost/com.qnx.	POST	200 / OK	text/…	[native c… / Script	201 B / 98 B	21 ms / 20 ms	

1.6 min 2.4 min 3.2 min

2 requests I 353 B transferred

Figure 6: The Network panel

By default, the table lists each of the requested resources in their requested order, and then charts the network activity as a waterfall timeline, with resources color-coded by type.

The waterfall timeline plots resources by the total time required to load the resource, from the initial request to the completion of the download. The pale segment of the resource bar in the chart represents the total latency; that is, the time the browser engine must wait from the moment it initially makes the request to the moment it receives the first packet of data for the resource. Two vertical lines on the chart show key page-load milestones:

- The blue line indicates the time when parsing of the content is complete and the `DOMContent` event fires.
- The red line indicates the time when all the resources have been loaded and the `load` event fires.

You can customize how the content is displayed in the **Network** panel by filtering based on type, or by sorting by any of the table headings. You can also reformat the chart to highlight different time measures.

Apply a filter to display a specific resource type

You can filter the resources Web Inspector displays.

By default, the **Networks** panel displays all resource requests in the table. The status bar at the bottom of the panel contains buttons that allow you to filter the resources displayed based on the resource type.

To filter the resources displayed:

1. Click the **Network** icon on the toolbar to display the **Network** panel.
2. If you haven't already done so, on the device or simulator, reload the page to allow Web Inspector to track and record network activity.
3. In the status bar at the bottom of the **Network** panel, choose the type of resource you want to display.

Change which time measure is displayed

You can reformat the network activity chart to highlight different time measures.

By default, when you measure the network activity, Web Inspector charts the network activity in a waterfall timeline. You can change this default through the **Network** panel:

1. Click the **Network** icon on the toolbar to display the **Network** panel.
2. If you haven't already done so, on the device or simulator, reload the page to allow Web Inspector to track and record network activity.
3. In the dropdown list above the chart, select one of the following:
    - **Timeline**: Displays the network activity in a waterfall timeline.
    - **Start time**: Highlights the time when each resource was requested.
    - **Response time**: Highlights the time when the resource is initially received.
    - **End time**: Highlights the time when the resource is completely loaded.
    - **Duration**: Displays the total length of time it takes to load the resource.
    - **Latency**: Displays the amount of delay between the start time value and the response time value.

Reorder the list of resources

Use the **Network** panel to reorder the resource list.

To reorder the resources list:

1. Click the **Networks** icon on the toolbar to display the **Networks** panel.
2. If you haven't already done so, on the device or simulator, reload the page to allow Web Inspector to track and record network activity.
3. Click a column heading to reorder the list based on the column data.

Debugging scripts

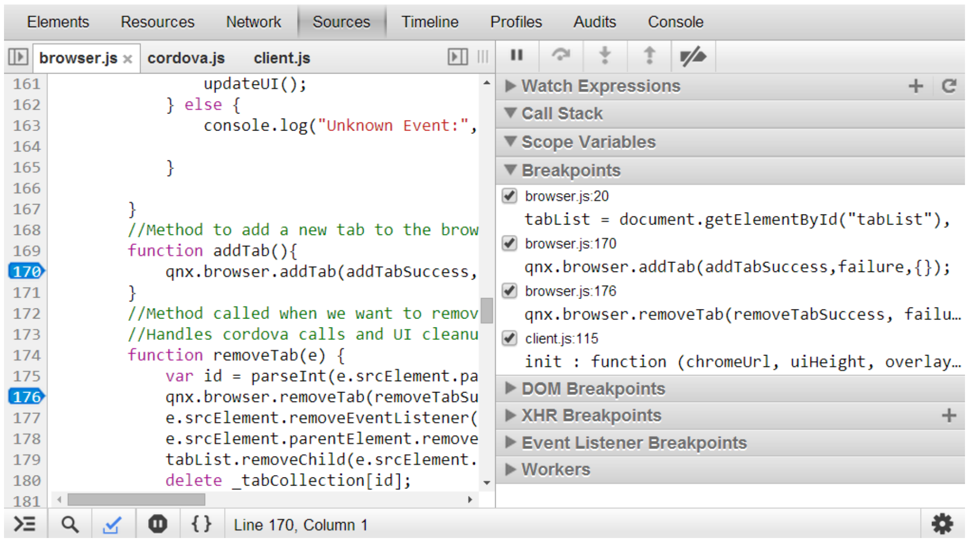
The **Sources** panel allows you to debug the JavaScript code used by your webpage.

Web Inspector lets you set breakpoints and step through your code, which can help to locate and correct problems in your code. When you determine that the script is functioning as intended, you can copy the changes back into the source file.

To use the **Sources** panel, you must first enable debugging. When you first view the **Sources** panel, Web Inspector prompts you to enable debugging for just the current session or for all sessions.

The Sources panel

You can click the **Sources** icon in the toolbar to display the **Sources** panel.



```
            updateUI();
        } else {
            console.log("Unknown Event:",

        }

    }
    //Method to add a new tab to the brow
    function addTab(){
        qnx.browser.addTab(addTabSuccess,
    }
    //Method called when we want to remov
    //Handles cordova calls and UI cleanu
    function removeTab(e) {
        var id = parseInt(e.srcElement.pa
        qnx.browser.removeTab(removeTabSu
        e.srcElement.removeEventListener(
        e.srcElement.parentElement.remove
        tabList.removeChild(e.srcElement.
        delete _tabCollection[id];
```

```
✓ browser.js:20
  tabList = document.getElementById("tabList"),
✓ browser.js:170
  qnx.browser.addTab(addTabSuccess,failure,{});
✓ browser.js:176
  qnx.browser.removeTab(removeTabSuccess, failu…
✓ client.js:115
  init : function (chromeUrl, uiHeight, overlay…
```

Figure 7: The Web Inspector Sources panel

The **Sources** panel is divided into two sections. On the left is the document pane, which allows you to view and debug JavaScript. On the right is a set of collapsible panes that display information related to the displayed script.

A toolbar at the top of the **Sources** panel allows you to choose the script file you want to inspect and to cycle between open scripts. It also provides a set of controls that allow you to step through the script displayed in the document pane.

Set and use breakpoints

Use the **Sources** panel to set breakpoints.

1. Click the **Sources** icon on the toolbar to show the **Sources** panel.
2. In the line gutter of the document pane, click the line where you want to set a breakpoint. A breakpoint marker appears in the line gutter and the new breakpoint is added to the **Breakpoints** pane, identified by the script filename and line number. The execution of the script pauses at the specified breakpoint.
3. Perform any of the following actions:
    - To continue the execution of the script beyond the current breakpoint, click the **Continue** button in the **Sources** panel toolbar.
    - To show the line of code associated with the breakpoint in the documents pane, click the breakpoint entry in the **Breakpoints** pane. The document pane shows and highlights the associated line.
    - To disable a single breakpoint without removing it, in the **Breakpoints** pane, uncheck the breakpoint. The execution of the script no longer pauses at the disabled breakpoint.
    - To deactivate or activate all the breakpoints listed in the **Breakpoints** pane without removing them, toggle the breakpoint activation switch at the right side of the **Scripts** panel toolbar.
    - To remove a breakpoint, locate and click the breakpoint marker in the line gutter of the document pane. The marker no longer appears in the line gutter and the breakpoint is removed from the **Breakpoints** pane.

Pause script execution

You can pause the script at any time to get a snapshot of the call stack and variable values.

To pause a script:

1. Click the **Sources** icon on the toolbar to display the **Sources** panel.
2. In the **Sources** panel toolbar, click the **Pause** button.

When the script pauses, the last line of JavaScript to be executed is highlighted. The call stack and the current in-scope variable values appear in the appropriate panes at the right of the panel.

Pause script execution on exceptions

You can configure Web Inspector to pause the execution of scripts whenever exceptions are thrown.

A tri-state toggle allows you to specify whether to pause for all exceptions, for only uncaught exceptions, or for no exceptions. To configure Web Inspector to pause script execution on exceptions:

1. Click the **Sources** icon on the toolbar to display the **Sources** panel.
2. Use the **Exceptions** button in the status bar at the bottom to choose one of the following behaviors:
   - To pause on all exceptions, click the **Exceptions** button until the icon turns blue.
   - To pause only on uncaught exceptions, click the **Exceptions** button until the icon turns red.
   - To not pause on any exceptions, click the **Exceptions** button until the icon turns gray.

Analyzing loading, script execution, and rendering times

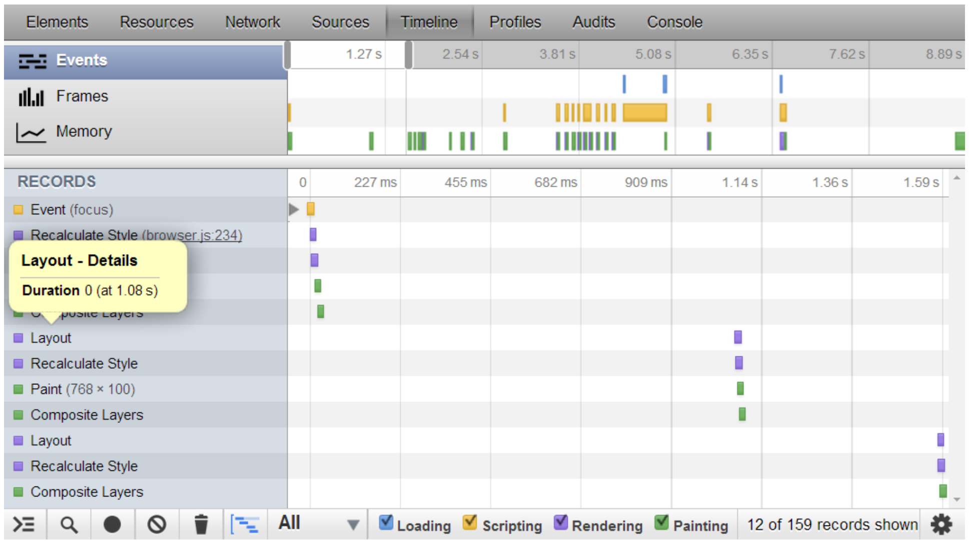
The **Timeline** panel can be used to gather information that helps you analyze browser activity and webpage loading times.

You can use the **Timeline** panel to analyze the time it takes to complete the different activities that the browser engine must perform to completely load and render your webpage. You can also contrain the view of the timeline and also filter events to better analyze your app's performance.

The Timeline panel

Initially, the panel displays no information, so you must click the **Record** button in the status bar to allow Web Inspector to record the browser engine activity.

As it records browser engine activity, the **Record** button turns red and Web Inspector adds data to the **Timeline** panel.



Event (focus)
Recalculate Style (browser.js:234)
Composite Layers
Layout
Recalculate Style
Paint (768 × 100)
Composite Layers
Layout
Recalculate Style
Composite Layers

Figure 8: The Web Inspector Timeline panel

Note that all browser engine activity pauses when the device is locked or the browser or HTML5 application is minimized. In order for Web Inspector to record any activity, the browser or HTML5 application must be the active application and the device or simulator screen mustn't be locked.

The **Timeline** panel is divided into two panes:

- In the top pane, the **Timeline** panel allows you to select which timeline view you want to show. You can choose three views:
  - **Events**: Shows the time it takes for the browser engine to complete each of the events required to completely load the content.
  - **Frames**: Shows the browser engine activity for each screen refresh.
  - **Memory**: Shows memory consumption over time.
- In the lower pane, the **Timeline** panel shows a waterfall timeline for the timespan that was selected in the top pane. The data in the timeline is determined by the mode you select in the top pane of the timeline's panel.

Constrain view to a specific time span

You can constrain the time span shown in the timeline. In the top pane of the **Timeline** panel, the portion of time displayed in the lower pane is represented by a white background. Two gray slider handles at the top left and right edges of this white background allow you to increase or decrease the selected timespan displayed in the timeline.

1. Click the **Timeline** icon on the toolbar to display the **Timeline** panel.
2. If necessary, record the browser engine activity to generate timeline data.
3. In the top pane of the **Timeline** panel, click and drag a gray slider handle to increase or decrease the time span.

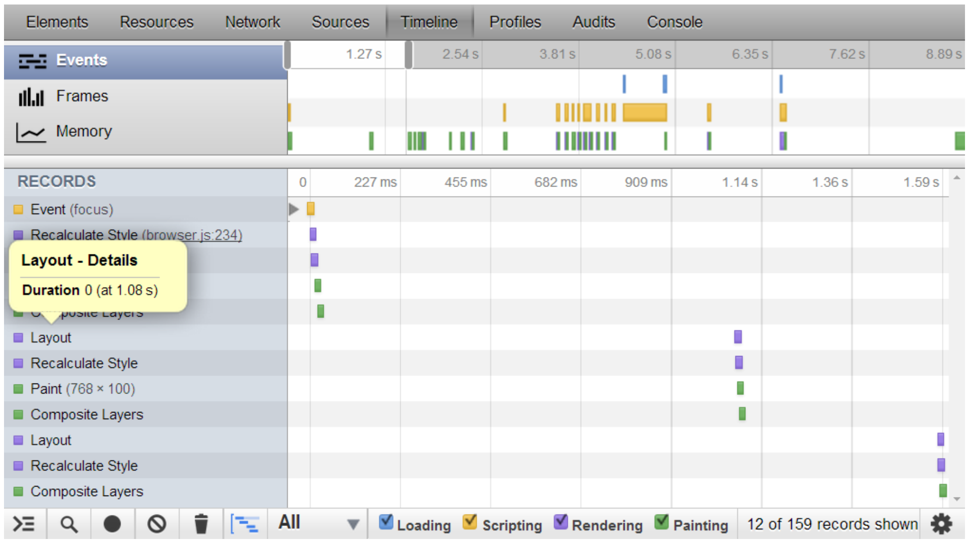


Figure 9: Changing the timespan on the Timeline panel

Filter the timeline events

By default, the **Timeline** panel shows all events in the table. The status bar at the bottom of the panel contains check boxes that allow you to show and hide events based on type.

1. If necessary, record the browser engine activity to generate timeline data.
2. In the status bar at the bottom of the **Timeline** panel, deselect the event types you want to remove from the timeline.

Analyzing memory usage and processing demands

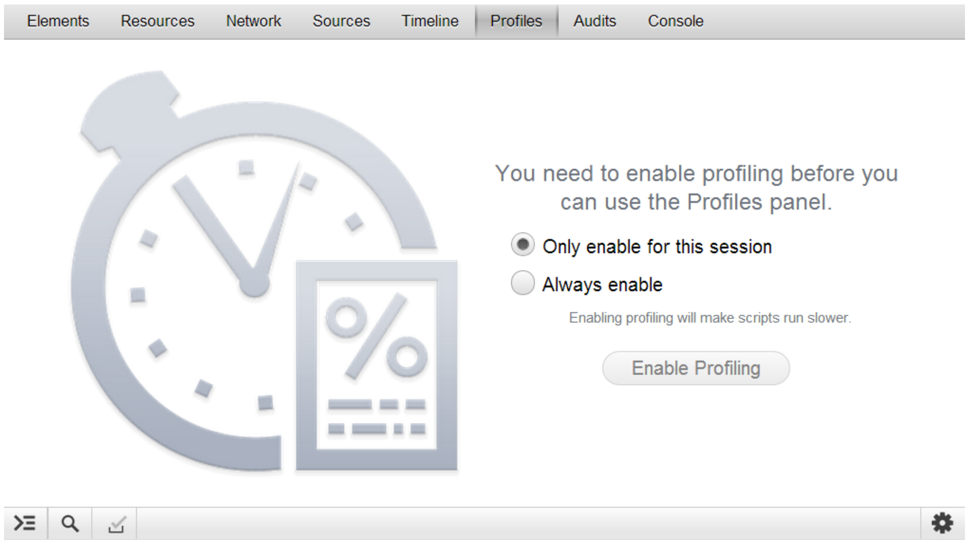
The **Profiles** panel allows you to analyze the memory usage and processing demands of your content.

You can use the **Profiles** panel to create a performance profile for your JavaScript and CSS files:

- For JavaScript files, Web Inspector examines and reports on the CPU usage for each function. You can view the CPU usage for a particular function and for the number of times that function was called.
- For CSS files, Web Inspector examines the processing demands for each selector. Web Inspector records the amount of time it took to search for matches for a particular selector and for the total number of matches for that selector.

To use the **Profiles** panel, you must first enable profiling. When you first view the **Profiles** panel, Web Inspector prompts you to enable profiling for just the current session or for all sessions.

If you haven't already enabled profiling, Web Inspector prompts you to do so.



You need to enable profiling before you can use the Profiles panel.

[✓ Only enable for this session]

Always enable

Enabling profiling will make scripts run slower.

Enable Profiling

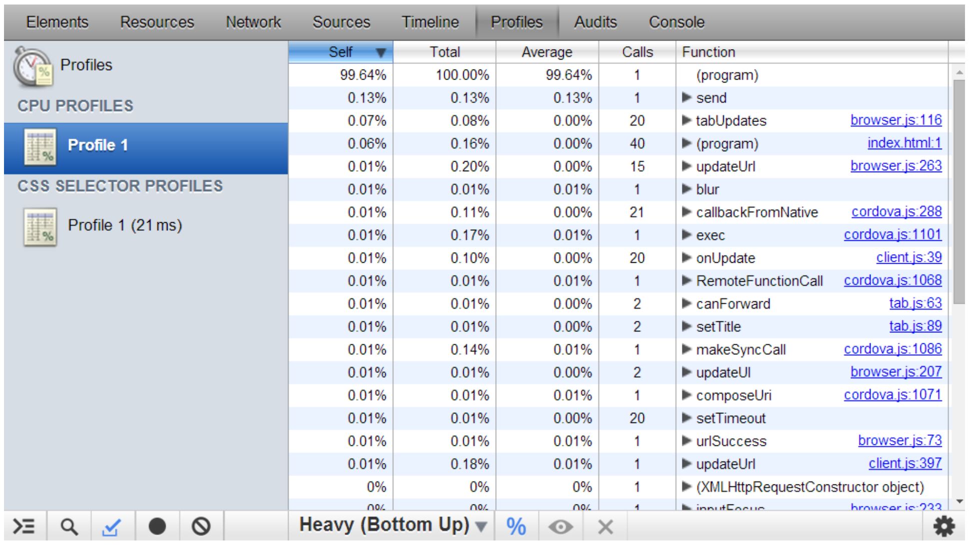
Figure 10: The Web Inspector Profiles panel

Profile the memory usage of your scripts

Use the **Profiles** panel to profile your scripts' memory usage.

To profile memory usage:

1. Click the **Profiles** icon on the toolbar to display the **Profiles** panel.
2. On the **Profiles** panel, select **Collect JavaScript CPU Profile**.
3. To start profiling your memory usage, click Start. The button turns red as the Web Inspector is recording the memory usage.
4. To stop recording, click **Stop**. When you stop recording, the new profile is added under the CPU Profiles section in the left pane and the profile's contents are displayed in the right pane:



Self	Total	Average	Calls	Function	
99.64%	100.00%	99.64%	1	(program)	
0.13%	0.13%	0.13%	1	▶ send	
0.07%	0.08%	0.00%	20	▶ tabUpdates	browser.js:116
0.06%	0.16%	0.00%	40	▶ (program)	index.html:1
0.01%	0.20%	0.00%	15	▶ updateUrl	browser.js:263
0.01%	0.01%	0.01%	1	▶ blur	
0.01%	0.11%	0.00%	21	▶ callbackFromNative	cordova.js:288
0.01%	0.17%	0.01%	1	▶ exec	cordova.js:1101
0.01%	0.10%	0.00%	20	▶ onUpdate	client.js:39
0.01%	0.01%	0.01%	1	▶ RemoteFunctionCall	cordova.js:1068
0.01%	0.01%	0.00%	2	▶ canForward	tab.js:63
0.01%	0.01%	0.00%	2	▶ setTitle	tab.js:89
0.01%	0.14%	0.01%	1	▶ makeSyncCall	cordova.js:1086
0.01%	0.01%	0.00%	2	▶ updateUI	browser.js:207
0.01%	0.01%	0.01%	1	▶ composeUri	cordova.js:1071
0.01%	0.01%	0.00%	20	▶ setTimeout	
0.01%	0.01%	0.01%	1	▶ urlSuccess	browser.js:73
0.01%	0.18%	0.01%	1	▶ updateUrl	client.js:397
0%	0%	0%	1	▶ (XMLHttpRequestConstructor object)	

Figure 11: Profile results: function execution times

The results indicate the amount of time the browser engine spent executing each function during the recording process, along with the number of times each function was called. An excessive amount of time spent in any one function can indicate a problem with the code.

5. To sort the data, perform any of the following actions:

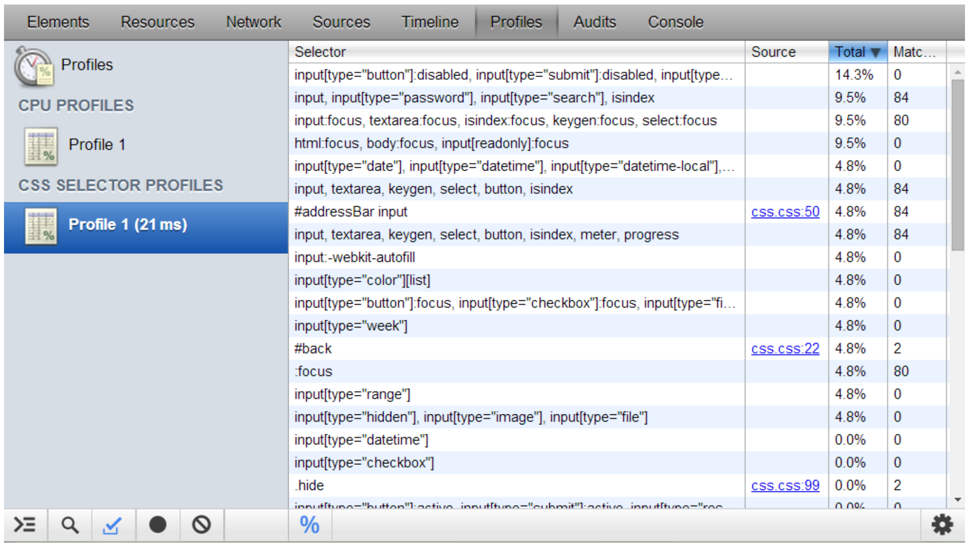
   - To sort by values in any column, double-click the column heading.
   - To display calls based on greatest impact on all exceptions or where they occurred in the call stack, in the status bar at the bottom of the panel, toggle between **Heavy (Bottom Up)** and **Tree (Top Down)**.
   - To specify whether values are presented as a time value or as a percentage of the total CPU usage required to process all the functions, toggle the percent button on or off.
   - To view a single function, select the call in the table and then click the focus button.
   - To exclude a single function from the data, select the function in the table and then click the exclude button.
   - To reload the original profile after focusing on or excluding a function, click the reload button.

Profile the performance of your CSS selectors

Use the **Profiles** panel to profile your scripts' memory usage.

To profile memory usage:

1. On the **Profiles** panel, select **Collect CSS Selector Profile**.
2. To start profiling your memory usage, click **Start**. The button turns red as Web Inspector records the memory usage.
3. To stop recording, click **Stop**. When you stop recording, the new profile is added under the **CSS Selector Profiles** section in the left pane and the profile's contents are displayed in the right pane:



Selector	Source	Total ▼	Matc…
input[type="button"]:disabled, input[type="submit"]:disabled, input[type…		14.3%	0
input, input[type="password"], input[type="search"], isindex		9.5%	84
input:focus, textarea:focus, isindex:focus, keygen:focus, select:focus		9.5%	80
html:focus, body:focus, input[readonly]:focus		9.5%	0
input[type="date"], input[type="datetime"], input[type="datetime-local"],…		4.8%	0
input, textarea, keygen, select, button, isindex		4.8%	84
#addressBar input	css.css:50	4.8%	84
input, textarea, keygen, select, button, isindex, meter, progress		4.8%	84
input:-webkit-autofill		4.8%	0
input[type="color"][list]		4.8%	0
input[type="button"]:focus, input[type="checkbox"]:focus, input[type="fi…		4.8%	0
input[type="week"]		4.8%	0
#back	css.css:22	4.8%	2
:focus		4.8%	80
input[type="range"]		4.8%	0
input[type="hidden"], input[type="image"], input[type="file"]		4.8%	0
input[type="datetime"]		0.0%	0
input[type="checkbox"]		0.0%	0
.hide	css.css:99	0.0%	2

Figure 12: Profile results: selector-style sheet matching times

The profile results indicate the amount of time the browser engine spent matching each selector in the associated style sheets, along with the total number of times the browser engine found a match for the selector.

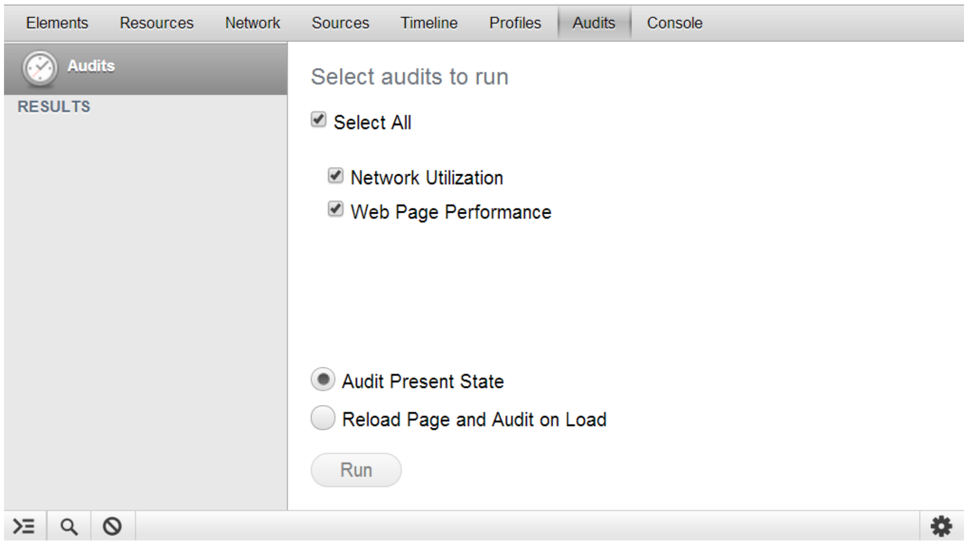
4. To specify whether the value of the **Total** column is presented as a time value or as a percentage of the total time required to process the CSS, toggle the percent button on or off.

Auditing your webpage

Web Inspector can audit your webpage for inefficiencies and, based on a set of best practices for web design, suggest changes you can make that can help improve network utilization and performance.

The **Audits** panel provides a list of perceived inefficiencies in your webpage design. For example, Web Inspector can analyze your resources and determine where you might consider combining script files or style sheets. The **Audits** panel can also inform you where you've needlessly downloaded styles that aren't used, specify resources where you haven't set cache-control directives, and suggest other optimizations.

The **Audits** panel can be especially helpful when you design pages for mobile browsers. On mobile browsers, network latency can extend download times; constrained processing power tends to increase rendering time and to slow webpage performance. As a result, eliminating inefficiencies in your webpage design can have an significant positive effect.



Select audits to run

- [✓] Select All
  - [✓] Network Utilization
  - [✓] Web Page Performance

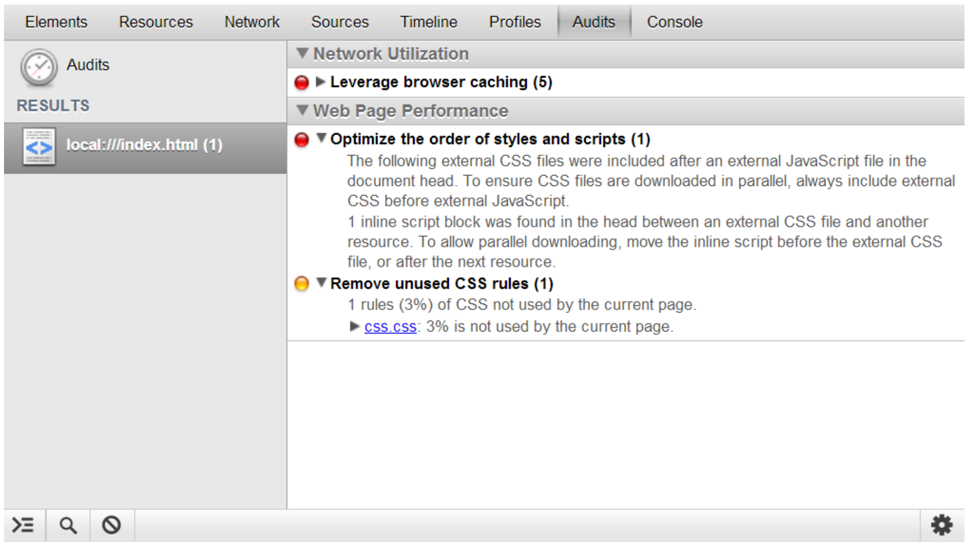
[✓ Audit Present State] Reload Page and Audit on Load

Figure 13: The Web Inspector Audits panel

The **Audits** panel lets you choose to:

- audit network utilization or page performance (or both)
- run the audit against the static page
- reload the page and run the audit as it loads.

Once you've run an audit, Web Inspector adds the report to the list at the left of the panel and shows the results in the main pane. The results suggest improvements you can make to your webpage to increase efficiencies.



▼ Network Utilization

● ▶ Leverage browser caching (5)

▼ Web Page Performance

● ▼ Optimize the order of styles and scripts (1)

The following external CSS files were included after an external JavaScript file in the document head. To ensure CSS files are downloaded in parallel, always include external CSS before external JavaScript.

1 inline script block was found in the head between an external CSS file and another resource. To allow parallel downloading, move the inline script before the external CSS file, or after the next resource.

● ▼ Remove unused CSS rules (1)

1 rules (3%) of CSS not used by the current page.

▶ css.css: 3% is not used by the current page.

Figure 14: An example of a Web Inspector audit report

Improving HTML5 app performance

These best practices can help you to develop more efficient and more responsive HTML5 apps.

1. Avoid Canvas.

   Canvas and SVG elements aren't optimal for use on mobile and embedded platforms, because proper hardware acceleration for these elements hasn't yet been finalized.

2. Avoid 2D transformations.

   Use 3D instead. For example, instead of `translateX(x)` use `translate3d(x,y,z)`. This practice forces hardware acceleration of the translation. You can use similar methods for most other transformations. **Avoid animating with JavaScript libraries!**

3. Avoid opacity, rounded corners, and gradients.

   These are a significant drain on performance when they're redrawn often. If used sparingly and mostly on static objects, they shouldn't impede performance, but if you mix these elements with animations, buttons, or anything that gets redrawn often, performance suffers. Consider using images instead of heavy CSS.

4. Remove elements from the DOM when modifying them.

   This technique is especially helpful when you're updating several DOM fields at once. For example, if you're scrolling through a list of 100 contacts and you want to refresh them, updating them one by one causes the list to be redrawn 100 times. But if you remove the entire list, update the contacts in memory, and then add the list again, this does only *two* redraws.

5. Hide elements you don't need.

   For instance, adding `display:none` to elements that don't need to be displayed prevents them from being rendered.

6. Avoid libraries intended for desktop use.

   Some JavaScript libraries are designed for use on a desktop browser with a powerful CPU. Try to limit the number of third-party JavaScript libraries included in your app or try to seek out versions optimized for mobile use.

7. Use image sprites.

   These are useful for preloading active element states (e.g., buttons with a “pressed” state).

Chapter 7
Creating Custom Cordova Plugins

Although HTML5 offers a wide range of functionality, that functionality is limited to what's provided by the SDK. There are a number of plugins available, but many of them are meant to be used on a mobile device, such as phone. You might find it useful to create your own plugins—especially if you want to access native platform functionality.

In addition to the *Cordova JavaScript Plugins* that are available with the HTML5 SDK, take a look at the *Apache Cordova Plugins Registry* to see if there's a plugin available for the functionality you require.

An app is implemented as a webpage (named **index.html** by default) that references whatever CSS, JavaScript, images, media files, or other resources are necessary for it to run. The app executes as a WebView within the native application framework. For the web app to interact with various device features the way native apps do, it must also reference a **cordova.js** file, which provides API bindings. If you want to access other features not provided by the HTML5 SDK, you have to write a *plugin*. A plugin is a bridge between the WebView the app is running in and the native layer of the platform. Plugins provide a mechanism to call into native APIs that aren't provided with the SDK.

It's presumed you know how to create Cordova plugins. For information about creating Cordova plugins, see the *Apache Cordova Documentation*.

Many of the services available with Apps and Media can be accessed through a PPS interface. In the sections that follow, a simple plugin (`com.qnx.demo`) creates a Persistent Publish/Subscribe(PPS) object and writes to it. You can use the same principles to manipulate other PPS objects.

Structure of a plugin

You can place the artifacts of the plugin in any directory structure, as long as you specify the file locations in the **plugin.xml** file. Here's a typical structure (**plugin_name**, **www**, **src**, and **blackberry10** are directory names):

- **plugin_name**
  - **plugin.xml**
  - **www**
    - **client.js**
  - **src**
    - **blackberry10**
      - **index.js**
      - **plugin_name.js**
      - other JavaScript or native files, as required (***.js**, ***.cpp**, ***.hpp**)

The JavaScript part of a plugin must contain, at a minimum, the following resources:

plugin.xml

The **plugin.xml** file is an XML document in the **plugins** namespace, **http://apache.org/cordova/ns/plugins/1.0**. The **plugin.xml** file contains a top-level `plugin` element that defines the plugin itself and child elements that define the file structure of the plugin.

You *must* name this file **plugin.xml**.

client.js

Considered the client side, this file exports APIs that a Cordova application can call. The APIs in **client.js** make calls to **index.js**. The APIs in **client.js** also connect callback functions to the events that fire the callbacks.

You *must* name this file **client.js**.

index.js

Cordova loads **index.js** and makes it accessible through the *cordova.exec()* bridge. The **client.js** file makes calls to the APIs in the **index.js** file, which in turn makes calls to JNEXT to communicate with the native side.

If your plugin needs to include events, make sure to define these events in **index.js**. Events are defined inside the *_actionMap* variable.

You *must* name this file **index.js**.

In our example, we need a file to deal with the PPS activities of the plugin. In this example, the file is named **demo.js** to reflect the name of the plugin. However, you can name this file whatever you want, but using the plugin name is a standard practice.

Depending on what your plugin needs to do, you might need to create other JavaScript files. In addition, if the native functionality you require isn't available through an existing interface (such as a PPS object or another plugin), you must write the C/C++ code to provide JavaScript access to that functionality.

The *cordova.exec()* function

You can structure your plugin's JavaScript according to your preference. However, you must use the *cordova.exec()* function to communicate between the Cordova JavaScript and the native environment.

The *cordova.exec()* function is defined in the **cordova.js** file and has the following signature:

```
exec (successFunction, failFunction, service, action, [args]);
```

The parameters are:

successFunction

Success function callback. Assuming your *exec()* call completes successfully, this function is invoked (optionally with any parameters you pass back to it).

failFunction

Error function callback. If the operation doesn't complete successfully, this function is invoked (optionally with an error parameter).

service

The name of the service to call into on the native side. This is mapped to a native class.

action

The name of the action to call into. This parameter is used by the native class receiving the *cordova.exec()* call, and essentially maps to a class's method.

args

Arguments to pass into the native environment.

The *cordova.exec()* function is the key to your plugin—it provides the link between the JavaScript and the native APIs. For more information about *cordova.exec()*, see the *Apache Cordova documentation.*

Example: Using the PPS interface

In the example, called `com.qnx.demo`, a plugin is built that provides methods and events for creating, updating, and watching a simple Persistent Publish/Subscribe (PPS) object. The source code for this project is available in the **demo** folder where you installed the HTML5 SDK.

In the example, called `com.qnx.demo`, a plugin is created that provide methods and events for creating, updating, and watching a simple PPS object. The example links to the PPS utilities file, **ppsUtils.js**, which is included as part of the **webplatform.js** file that's linked in by the Cordova packager (build command). The namespace for **ppsUtils.js** is `qnx.webplatform.pps`. The files in the demo follow the structure described in the "*Structure of a plugin* (p. 71)" section.

Files in the example

This example includes the following files:

- **plugin.xml** in the **com.qnx.demo** directory. This file declares the namespace of the plugin, and describes the file structure.
- **demo.js** and **index.js** in the **com.qnx.demo/src/blacberry10** directory. The **demo.js** file initializes the extension and defines functions for handling events and returning PPS data. The **index.js** file sets up the events that are triggered.
- **client.js** in the **com.qnx.demo/www** directory. This file defines the externally visible methods that apps can use.

Here's more information about these files.

`demo.js`

The **demo.js** file provides the core functionality of the plugin. First, you link to the `qnx.webplatform.pps` namespace to access PPS functionality and then declare some variables that are used later:

```
var _pps = qnx.webplatform.pps,
    _readerPPS,
    _writerPPS,
    _triggerUpdate;
```

Now, you can create the demo PPS object. The PPS object specified by `/pps/qnx/demo` is the location of the file where the PPS objects are stored on the target. There's more code that handles errors and so on, but for simplification, just create the PPS object:

```
_readerPPS = _pps.create("/pps/qnx/demo", _pps.PPSMode.DELTA);
_readerPPS.onNewData = function(event) {
    if (_triggerUpdate && event && event.data) {
        _triggerUpdate(event.data);
...

_writerPPS = _pps.create("/pps/qnx/demo", _pps.PPSMode.DELTA);
...
```

The preceding code creates a PPS object (**/pps/qnx/demo**) with two handles: one for reading (`_readerPPS`) and one for writing (`_writerPPS`). These handles provide access to the JavaScript APIs in `qnx.webplatform.pps` used for manipulating the PPS data. The `_triggerUpdate` method is used for handling events. Here, the method defines the action to take when new data is available in the PPS object.

The **demo.js** file also defines the functions for handling the event trigger (*setTriggerUpdate()*) and for getting (*get()*) and setting (*set()*) the PPS data. These functions are exported so that they can be called from the **index.js** file:

```
/**
 * Sets the trigger function to call when an event is triggered
 * @param trigger {Function} The trigger function to call when an event is fired
 */
setTriggerUpdate: function(trigger) {
    _triggerUpdate = trigger;
},

/**
 * Returns the demo object
 * @returns {Object} The demo object
 */
get: function(settings) {
     return _readerPPS.data.demo;
},

/**
 * Set a demo object field
 * @param {String} key The data key
 * @param {Mixed} value The data value
 */
set: function(key, value) {
     var data = {};
     data[key] = value;
     _writerPPS.write(data);
}
```

index.js

In the **index.js** file, the functions that are defined are made available to the clients through the **client.js** file:

```
/**
* Starts triggering events
* @param {Function} success Function to call if the operation is a success
* @param {Function} fail Function to call if the operation fails
* @param {Object} args The arguments supplied
* @param {Object} env Environment variables
```

```
    */
    startEvents: function(success, fail, args, env) {
        _eventResult = new PluginResult(args, env)
        try {
            _demo.setTriggerUpdate(function (data) {
                _eventResult.callbackOk(data, true);
            });
            _eventResult.noResult(true);
        } catch (e) {
            _eventResult.error("error in startEvents: " + JSON.stringify(e), false);
        }
    },

    /**
     * Stops triggering events
     * @param {Function} success Function to call if the operation is a success
     * @param {Function} fail Function to call if the operation fails
     * @param {Object} args The arguments supplied
     * @param {Object} env Environment variables
     */
    stopEvents: function(success, fail, args, env) {
       var result = new PluginResult(args, env);
       try {
           //disable the event trigger
            _demo.setTriggerUpdate(null);
            result.ok(undefined, false);

            //cleanup
            _eventResult.noResult(false);
            delete _eventResult;
        } catch (e) {
            result.error("error in stopEvents: " + JSON.stringify(e), false);
        }
    },

    /**
     * Returns system settings
     * @param success {Function} Function to call if the operation is a success
     * @param fail {Function} Function to call if the operation fails
     * @param args {Object} The arguments supplied
     * @param env {Object} Environment variables
     */
    get: function(success, fail, args, env) {
        var result = new PluginResult(args, env)
        try {
            var fixedArgs = _wwfix.parseArgs(args);
            var data = _demo.get();
            result.ok(data, false);
        } catch (e) {
            result.error(JSON.stringify(e), false);
        }
    },

    /**
     * Sets one or more system settings
     * @param success {Function} Function to call if the operation is a success
     * @param fail {Function} Function to call if the operation fails
     * @param args {Object} The arguments supplied
     * @param env {Object} Environment variables
     */
    set: function(success, fail, args, env) {
        var result = new PluginResult(args, env)
        try {
            var fixedArgs = _wwfix.parseArgs(args);
            _demo.set(fixedArgs.key, fixedArgs.value);
            result.ok(undefined, false);
        } catch (e) {
            result.error(JSON.stringify(e), false);
        }
    }
```

In the preceding code, these functions are defined:

- *startEvents()* and *stopEvents()*, which use the *setTriggerUpdate()* function that was defined in **index.js**. This function is referenced as `_demo.setTriggerUpdate()`.
- *get()*, which uses the *get()* function that was defined in **index.js** (referenced here as `_demo.get()`).
- *set()*, which uses the *set()* function that was defined in **index.js** (referenced here as `_demo.set()`).

client.js

The **client.js** file defines the public-facing interface, or client side, for the plugin. This file specifies where the *cordova.exec()* function is used to call the PPS functions that were previously defined in server-side files.

First, variables are declared. The variable `_ID` is used for calls to *cordova.exec()*:

```
var _self = {},
    _ID = "com.qnx.demo",
    _utils = cordova.require('cordova/utils'),
    _watches = {};
```

Next, a function is created to handle PPS update events. This function is also passed to *cordova.exec()*, but isn't accessible to other applications:

```
/**
 * Handles update events for this plugin
 * @param data {Array} The updated data provided by the event
 * @private
 */
function onUpdate(data) {
    var keys = Object.keys(_watches);
    for (var i=0; i<keys.length; i++) {
        setTimeout(_watches[keys[i]](data), 0);
    }
}
```

Finally, public-facing functions are declared and then and exported. For each function, *cordova.exec()* is called. The *cordova.exec()* function is passed the name of the function to call from **index.js**:

```
/**
 * Watch for PPS object changes
 * @param {Function} callback The function to call when a change is detected.
 * @return {String} An ID for the added watch.
 * @example
 *
 * //define a callback function
 * function myCallback(myData) {
 *        //just send data to log
 *        console.log("Changed data: " , myData);
 *        }
 * }
 *
 * var watchId = car.demo.watchDemo(myCallback);
 */
_self.watchDemo = function (callback) {
    var watchId = _utils.createUUID();

    _watches[watchId] = callback;
    if (Object.keys(_watches).length === 1) {
        window.cordova.exec(onUpdate, null, _ID, 'startEvents', null, false);
    }

    return watchId;
}

/**
 * Stop watching for PPS changes
 * @param {Number} watchId The watch ID as returned by car.demo.watchDemo().
 * @example
 *
 * car.sensors.cancelWatch(watchId);
 */
_self.cancelWatch = function (watchId) {
    if (_watches[watchId]) {
        delete _watches[watchId];
        if (Object.keys(_watches).length === 0) {
            window.cordova.exec(null, null, _ID, 'stopEvents', null, false);
        }
    }
}

/**
 * Get the value of the demo PPS object
 * @returns {Object} The demo object contents.
 */
_self.get = function () {
        var value = null,
        success = function (data, response) {
            value = data;
        },
        fail = function (data, response) {
            throw data;
        };

    try {
        window.cordova.exec(success, fail, _ID, 'get', null);
    } catch (e) {
        console.error(e);
    }
    return value;
}

/**
 * Set a demo object field
 * @param {String} key The data key
 * @param {Mixed} value The data value
 */
_self.set = function (key, value) {
    window.cordova.exec(null, null, _ID, 'set', { key: key, value: value });
}

module.exports = _self;
```

Applications that use this plugin must define a callback function for Cordova to call when *watchDemo()* is successful.

Chapter 8
WebLauncher's JavaScript APIs

You can use a variety of different APIs to develop your apps.

You can use the Cordova framework and the Browser Chrome APIs to customize the Browser Chrome. For most applications, the functionality exposed by this API isn't required. For HTML5 app development, the functionality you require is available in the *Cordova JavaScript Plugins*, public JavaScript libraries, or HTML5. For more information, see "*HTML5 SDK Overview* (p. 9)."

The following WebLauncher API categories are available for development work:

- `application` (p. 82)—functions and properties for the current app
- `webinspector` (p. 83)—functions related to the Web Inspector tool
- `webview` (p. 84)—functions to manage your webviews

WebLauncher `application` API

The application framework provides functionality to the browser engine to allow it to support apps.

This environment lets you create and deploy apps built from web technologies (HTML5, CSS3, and JavaScript) with plugins that provide access to the underlying device hardware and native services, just like native C/C++ apps.

The following WebLauncher `application` API is available for you to implement your own browser chrome:

- `application.adduri`
- `application.bindToNetworkDevice`
- `application.coverSize`
- `application.extendTerminate`
- `application.getenv`
- `application.isDeviceLocked`
- `application.isForeground`
- `application.lockRotation`
- `application.minimizeWindow`
- `application.newWallpaper`
- `application.notifyRotateComplete`
- `application.realpath`
- `application.requestExit`
- `application.rotate`
- `application.setKeyboardTracking`
- `application.setPooled`
- `application.setSwipeStart`
- `application.setenv`
- `application.systemFontFamily`
- `application.systemFontSize`
- `application.systemRegion`
- `application.unlockRotation`
- `application.unsetenv`
- `application.updateCover`
- `applicationWindow.flushContext`
- `applicationWindow.setSize`
- `applicationWindow.setVisible`

WebLauncher `webinspector` API

Use the `webinspector` tool to troubleshoot and optimize your web content for your apps.

Web Inspector is a useful debugging and profiling development tool for web content. It includes various features and capabilities, including inspection, profiling, console integration, and more.

The following WebLauncher `webinspector` APIs are available for you to implement your own browser chrome:

- `webInspector.enabled`
- `webInspector.enabledForWebView`
- `webInspector.port`
- `webInspector.setEnabled`
- `webInspector.setEnabledForWebView`

WebLauncher `webview` API

You can use the WebLauncher `webview` API to implement your own browser chrome.

Every HTML5 app has its own WebView. A WebView is a view that's rendered by the web engine to display an app. The browser engine provides a set of core classes that you can use to display web content in a window. By default, the browser engine implements the most basic functionality of a browser, such as the ability to follow links and to download and display content. You can use the engine's functionality at the most basic level to display web content in your app or you can use APIs to create your own full-featured, customized, web-based app.

The QNX SDK for Apps and Media provides a multiprocess architecture that allows multiple WebViews to share a common engine instance or each to run in its own engine instance. Trusted apps are run *in process*, sharing a web engine instance. Other apps are run *out of process*, protected from each other by process boundaries, each with its own web engine instance. You can implement each WebView with a separate JavaScript application framework, such as jQuery or Sencha Touch.

The following WebLauncher `webview` APIs are available for you to implement your own browser chrome:

- `webview.addKnownSSLCertificate`
- `webview.addKnownSSLCertificates`
- `webview.addOriginAccessWhitelistEntry`
- `webview.applicationSwipeInEvent`
- `webview.applicationWindowGroup`
- `webview.arguments`
- `webview.assignFocus`
- `webview.autofillTextField`
- `webview.automationLog`
- `webview.backgroundColor`
- `webview.bitmapZoom`
- `webview.canGoBack`
- `webview.canGoForward`
- `webview.cancelVibration`
- `webview.captureContents`
- `webview.certificateInfo`
- `webview.cleanupSSLCertificateDetails`
- `webview.clearAutofillData`
- `webview.clearBackForwardList`
- `webview.clearBrowsingData`
- `webview.clearCache`
- `webview.clearCookies`
- `webview.clearCredentials`
- `webview.clearDatabases`
- `webview.clearFocus`
- `webview.clearHistory`
- `webview.clearLocalStorage`

- `webview.clearWebFileSystem`
- `webview.contentRectangle`
- `webview.continueSSLHandshaking`
- `webview.convertIDNtoReadableStringByLanguage`
- `webview.create`
- `webview.createFadeColorWindow`
- `webview.currentContext`
- `webview.defaultFontSize`
- `webview.defaultTextEncoding`
- `webview.delete`
- `webview.destroy`
- `webview.destroyFadeColorWindow`
- `webview.destroyIfNotRejectedByUser`
- `webview.devicePixelRatio`
- `webview.dialogResponse`
- `webview.downloadCancel`
- `webview.downloadPause`
- `webview.downloadRemove`
- `webview.downloadResume`
- `webview.downloadRetry`
- `webview.downloadUpdate`
- `webview.downloadUrl`
- `webview.enableQnxJavaScriptObject`
- `webview.encryptionInfo`
- `webview.eventNames`
- `webview.executeJavaScript`
- `webview.executeJavaScriptFunction`
- `webview.extraHttpHeaders`
- `webview.fadeColorWindowPlatformHandle`
- `webview.favicon`
- `webview.fileSystemAPISandboxed`
- `webview.findString`
- `webview.focusNextField`
- `webview.focusPreviousField`
- `webview.forcedTextEncoding`
- `webview.fullScreenVideoCapable`
- `webview.fullScreenVideoExited`
- `webview.getCookies`
- `webview.getSSLCertificateDetails`
- `webview.goBack`
- `webview.goForward`
- `webview.handleContextMenuResponse`
- `webview.handleWebInspectorMessageToBackend`

- `webview.historyLength`
- `webview.historyPosition`
- `webview.initialize`
- `webview.isActive`
- `webview.isAllPropertyChangedEventsEnabled`
- `webview.isAllWebEventsEnabled`
- `webview.isAlwaysShowKeyboardOnFocus`
- `webview.isAnyPropertyChangedEventsEnabled`
- `webview.isAnyWebEventsEnabled`
- `webview.isAutoDeferNetworkingAndJavaScript`
- `webview.isBlockPopups`
- `webview.isDeferNetworkingAndJavaScript`
- `webview.isEnableCookies`
- `webview.isEnableCredentialAutofill`
- `webview.isEnableCrossSiteXHR`
- `webview.isEnableDNSPrefetch`
- `webview.isEnableDefaultOverScrollBackground`
- `webview.isEnableDialogRequestedEvents`
- `webview.isEnableDiskCache`
- `webview.isEnableDownloadableBinaryFonts`
- `webview.isEnableFineCursorControl`
- `webview.isEnableFormAutofill`
- `webview.isEnableGeolocation`
- `webview.isEnableInputMethodSupport`
- `webview.isEnableInputNotifications`
- `webview.isEnableJavaScript`
- `webview.isEnableLocalAccessToAllCookies`
- `webview.isEnableMediaRTSP`
- `webview.isEnableNetworkResourceRequestedEvents`
- `webview.isEnablePlugins`
- `webview.isEnableSoundOnAnchorElementTouchEvents`
- `webview.isEnableSpatialNavigation`
- `webview.isEnableTextSelectionControls`
- `webview.isEnableWebInspector`
- `webview.isEnableWebSockets`
- `webview.isLoadImages`
- `webview.isPluginFullScreen`
- `webview.isPrivateBrowsing`
- `webview.isPropertyChangedEventEnabled`
- `webview.isVideoFullScreen`
- `webview.isVisible`
- `webview.isWebEventEnabled`
- `webview.isZoomToFitWidthOnLoad`

- `webview.javaScriptInterruptTimeout`
- `webview.jsScreenWindowHandle`
- `webview.knownSSLCertificate`
- `webview.knownSSLCertificates`
- `webview.loadFile`
- `webview.loadProgress`
- `webview.loadString`
- `webview.loadStringWithBase`
- `webview.loadURL`
- `webview.location`
- `webview.lockProperties`
- `webview.log`
- `webview.maximumScale`
- `webview.minimumFontSize`
- `webview.minimumScale`
- `webview.notificationClicked`
- `webview.notificationClosed`
- `webview.notificationError`
- `webview.notificationShown`
- `webview.notifyApplicationOrientationDone`
- `webview.notifyContextMenuCancelled`
- `webview.notifyDataReceived`
- `webview.notifyDone`
- `webview.notifyHeaderReceived`
- `webview.notifyOpen`
- `webview.notifySystemLowMemory`
- `webview.notifyViewportChanged`
- `webview.openWindowResponse`
- `webview.originalLocation`
- `webview.overScrollColor`
- `webview.printToStderr`
- `webview.printToStdout`
- `webview.reload`
- `webview.removeAllKnownSSLCertificates`
- `webview.removeGeolocationFilter`
- `webview.removeKnownSSLCertificate`
- `webview.removeOriginAccessWhitelistEntry`
- `webview.requestCurrentContextUpdate`
- `webview.requestSession`
- `webview.restoreSession`
- `webview.scale`
- `webview.scrollBy`
- `webview.scrollPosition`

- `webview.secureType`
- `webview.securityInfo`
- `webview.sensitivity`
- `webview.setActive`
- `webview.setAllPropertyChangedEventsEnabled`
- `webview.setAllWebEventsEnabled`
- `webview.setAllowGeolocation`
- `webview.setAllowNotification`
- `webview.setAllowUserMedia`
- `webview.setAllowWebInspection`
- `webview.setAlwaysShowKeyboardOnFocus`
- `webview.setApplicationActivationState`
- `webview.setApplicationOrientation`
- `webview.setAutoDeferNetworkingAndJavaScript`
- `webview.setBackgroundColor`
- `webview.setBitmapZooming`
- `webview.setBlockPopups`
- `webview.setCookies`
- `webview.setDefaultFontSize`
- `webview.setDefaultTextEncoding`
- `webview.setDeferNetworkingAndJavaScript`
- `webview.setDevicePixelRatio`
- `webview.setEnableCookies`
- `webview.setEnableCredentialAutofill`
- `webview.setEnableCrossSiteXHR`
- `webview.setEnableDNSPrefetch`
- `webview.setEnableDefaultOverScrollBackground`
- `webview.setEnableDialogRequestedEvents`
- `webview.setEnableDiskCache`
- `webview.setEnableDownloadableBinaryFonts`
- `webview.setEnableFineCursorControl`
- `webview.setEnableFormAutofill`
- `webview.setEnableGeolocation`
- `webview.setEnableInputMethodSupport`
- `webview.setEnableInputNotifications`
- `webview.setEnableJavaScript`
- `webview.setEnableLocalAccessToAllCookies`
- `webview.setEnableMediaRTSP`
- `webview.setEnableNetworkResourceRequestedEvents`
- `webview.setEnablePlugins`
- `webview.setEnableSoundOnAnchorElementTouchEvents`
- `webview.setEnableSpatialNavigation`
- `webview.setEnableTextSelectionControls`

- `webview.setEnableWebSockets`
- `webview.setEnabledOutOfProcessWebInspector`
- `webview.setExtraHttpHeaders`
- `webview.setExtraPluginDirectory`
- `webview.setFadeColorWindowRect`
- `webview.setFadeWindowColor`
- `webview.setFileSystemAPISandboxed`
- `webview.setForcedTextEncoding`
- `webview.setFullScreenVideoCapable`
- `webview.setGeometry`
- `webview.setHistoryPosition`
- `webview.setJSWebViewBindings`
- `webview.setJavaScriptInterruptTimeout`
- `webview.setKeyboardVisibilityLocked`
- `webview.setKeyboardVisible`
- `webview.setLayerTilerPrefillRect`
- `webview.setLoadImages`
- `webview.setMinimumFontSize`
- `webview.setOverScrollColor`
- `webview.setPatternMatchingEnabled`
- `webview.setPopupWebView`
- `webview.setPrivateBrowsing`
- `webview.setPropertyChangedEventEnabled`
- `webview.setScreenPowerState`
- `webview.setScrollPosition`
- `webview.setScrolling`
- `webview.setSensitivity`
- `webview.setStandalone`
- `webview.setTemporaryViewportSize`
- `webview.setTextReflowMode`
- `webview.setUserAgent`
- `webview.setUserStyleSheetLocation`
- `webview.setViewport`
- `webview.setViewportHeight`
- `webview.setViewportInitialScale`
- `webview.setViewportMaximumScale`
- `webview.setViewportMinimumScale`
- `webview.setViewportTargetDensityDpi`
- `webview.setViewportUserScalable`
- `webview.setViewportWidth`
- `webview.setVisible`
- `webview.setWebEventEnabled`
- `webview.setWebInspectorEnabled`

- `webview.setZOrder`
- `webview.setZoomFactor`
- `webview.setZoomToFitWidthOnLoad`
- `webview.status`
- `webview.stop`
- `webview.submitForm`
- `webview.syncProxyCredential`
- `webview.textEncoding`
- `webview.textHasAttribute`
- `webview.textReflowMode`
- `webview.title`
- `webview.tooltip`
- `webview.unlockProperties`
- `webview.updateDisabledPluginFiles`
- `webview.updateGeolocationFilter`
- `webview.updateNotificationPermission`
- `webview.userAgent`
- `webview.userStyleSheetLocation`
- `webview.viewport`
- `webview.viewportHeight`
- `webview.viewportInitialScale`
- `webview.viewportMaximumScale`
- `webview.viewportMinimumScale`
- `webview.viewportTargetDensityDpi`
- `webview.viewportUserScalable`
- `webview.viewportWidth`
- `webview.webInspectorPort`
- `webview.windowUniqueId`
- `webview.zOrder`
- `webview.zoomFactor`

Index