

---

# Amazon Silk

## Developer Guide



## Amazon Silk: Developer Guide

Copyright © 2015 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

The following are trademarks of Amazon Web Services, Inc.: Amazon, Amazon Web Services Design, AWS, Amazon CloudFront, AWS CloudTrail, AWS CodeDeploy, Amazon Cognito, Amazon DevPay, DynamoDB, ElastiCache, Amazon EC2, Amazon Elastic Compute Cloud, Amazon Glacier, Amazon Kinesis, Kindle, Kindle Fire, AWS Marketplace Design, Mechanical Turk, Amazon Redshift, Amazon Route 53, Amazon S3, Amazon VPC, and Amazon WorkDocs. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon.

All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

AWS documentation posted on the Alpha server is for internal testing and review purposes only. It is not intended for external customers.

---

## Table of Contents

What Is Amazon Silk? .....	1
Split Browser Architecture .....	1
Machine Learning .....	2
Getting Started with Amazon Silk .....	3
Are You an Amazon Silk User? .....	3
Are You a Site Owner? .....	3
Accessing the Amazon Silk User Guide .....	4
Amazon Silk Settings Menu .....	4
Developing Web Content for Amazon Silk .....	6
Responsive Web Design .....	6
How Does Responsive Web Design Work? .....	8
Responsive Design and Amazon Silk .....	9
The Takeaway .....	11
Additional Resources .....	11
Amazon Silk User Agent Strings .....	12
User Agent String Examples .....	13
User Agent Detection .....	14
Feature Detection .....	14
How To Detect a Feature .....	15
Additional Resources .....	16
Media Handling .....	16
Images .....	16
Audio .....	17
Video .....	17
Touch .....	17
Touch Example Using jQuery Mobile .....	18
Additional Resources .....	20
Screen Resolution .....	21
Secure Connections .....	22
JavaScript .....	22
JavaScript Loading .....	22
Caching .....	23
Remote Debugging .....	24
Do Not Track .....	25
Additional Resources .....	26
HTML5 Support .....	27
Support for HTML5 and Related Features .....	27
HTML5 APIs .....	29
Animation Timing API .....	29
Application Cache API .....	30
Cross-Origin Resource Sharing .....	31
File API .....	32
File System API .....	32
Geolocation API .....	34
Indexed Database API .....	35
Server-Sent Events .....	36
Touch Events .....	36
XMLHttpRequest Level 2 .....	37
Web SQL Database .....	37
Web Storage .....	38
Web Workers API .....	39
WebGL .....	39
WebSocket API .....	40
HTML5 Elements and Attributes .....	40
Audio Element .....	41

Canvas Element .....	41
contenteditable Attribute .....	42
Input Types .....	43
Keygen Element .....	43
Meter Element .....	44
Output Element .....	44
Progress Element .....	44
SVG Element .....	45
Video Element .....	45
CSS3 Support .....	46
Media Queries .....	46
Transform Property (2D) .....	48
Transitions .....	48
viewport Meta Element .....	49
Tutorials .....	51
Detect the Silk User Agent .....	51
Create Drop-down Menus for a Touch Screen .....	53
Troubleshooting .....	56
How do I determine the Amazon Silkbuild version? .....	56
Does Amazon Silk support inline playback of HTML5 video? .....	57
Why is the mobile (or desktop) version of my site being delivered? .....	57
How do I determine the client's IP address? .....	57
Obtaining the Client IP Address .....	57
Why won't my Flash content play? .....	58
Cookie Issues with ASP.NET 4.0 .....	58
I own an ASP.NET 4-based site. How can I tell if my customers are encountering the cookie problem? .....	59
I own an ASP.NET 4-based site. How do I fix the cookie problem? .....	59
I'm a Amazon Silk user encountering the cookie problem. What should I do? .....	59
Why is Amazon Silk 2.0 affected, but not Amazon Silk 1.0? .....	59
Resources .....	61
Sites .....	61
Video Talks and Tutorials .....	62
Books .....	62
Document History .....	63

# What Is Amazon Silk?

---



## Topics

- [Split Browser Architecture \(p. 1\)](#)
- [Machine Learning \(p. 2\)](#)

Amazon Silk is a next-generation web browser available only on Fire tablets and phones. Built on a split architecture that divides processing between the client and the Amazon cloud, Amazon Silk is designed to create a faster, more responsive mobile browsing experience.

Unlike other browsers, Amazon Silk has the vast capacity of Amazon Web Services (AWS) behind it. Using AWS, Amazon Silk analyzes aggregate web traffic patterns, preprocesses pages, and applies predictive algorithms to determine the fastest way to deliver content to the device.

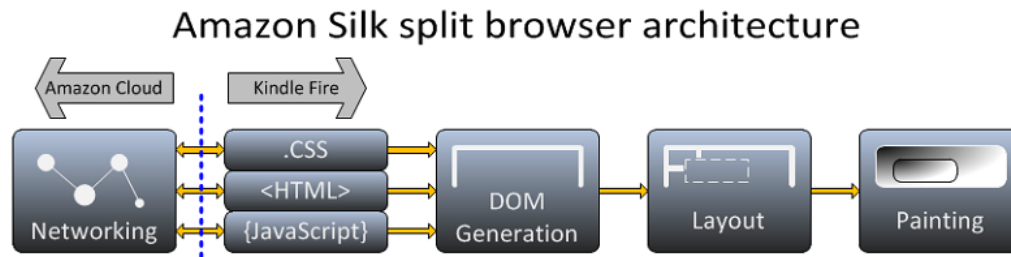
To minimize latency and page load times, Amazon Silk routes requests through remote proxy servers powered by [Amazon EC2](#). These cloud servers provide high-performance connection speeds and computing power not normally available to a mobile form factor.

## Split Browser Architecture

When a traditional browser requests a web page, the browser has to complete a series of resource-intensive, client-side processes before it can display the page. For a typical page request, a browser might perform a DNS look-up, open a series of TCP/IP connections to the server, retrieve resources (HTML, CSS, JavaScript, media files), and use these resources to generate a document object model (DOM). Then the browser's layout engine renders a layout, and the browser paints the page.

All of these client-server interactions can add up to slow page loads, especially given the limited resources of a mobile device and the increasing size and complexity of modern websites. To load a news page that features many media components and scripts, the browser might need to make several hundred requests. And the initial page load isn't the only aspect of browsing that consumes system resources. Dynamic modern web pages invite many additional interactions. Users scroll, zoom, and initiate on-page events that result in JavaScript invocations or style recalculations. More and more, web pages are turning into long-running applications.

That's where Amazon Silk comes in. The architecture is designed for both rapid page loading and extended engagement with the page. With Amazon Silk, much of the processing behind page rendering can be performed in the Amazon cloud.



Instead of relying on the limited client-side processing power of a mobile device, Amazon Silk leverages the processing power of Amazon EC2 virtual servers. The EC2 servers provide high-performance CPUs, expansive memory, and fat network pipes, all of which help minimize page load times.

## Machine Learning

All browsers make an effort to optimize performance over time. A simple example is device-side caching, wherein the browser temporarily stores requested resources so that subsequent requests can be fulfilled more efficiently. However, many optimizations are limited because they're based only on individual browsing behavior and because they tend to be ephemeral. For example, clearing browsing data cancels the performance gains made by caching.

By combining machine learning with the resources available in the Amazon cloud, Amazon Silk realizes new possibilities in optimization. Amazon Silk can preprocess and analyze web sites and make dynamic decisions about how to deliver pages to provide the best possible browsing experience.

Because so much of the work of delivering web pages happens on the Amazon cloud, Amazon Silk's optimizations are informed by massive amounts of anonymous, aggregated web traffic data. In effect, Amazon Silk can learn from web traffic patterns and use this learning to improve the individual user experience.

# Getting Started with Amazon Silk

---

## Topics

- [Are You an Amazon Silk User? \(p. 3\)](#)
- [Are You a Site Owner? \(p. 3\)](#)
- [Accessing the Amazon Silk User Guide \(p. 4\)](#)
- [Amazon Silk Settings Menu \(p. 4\)](#)

## Are You an Amazon Silk User?

If you're an Amazon Silk user, and you'd like a more general introduction to Silk, see [Accessing the Amazon Silk User Guide \(p. 4\)](#).

## Are You a Site Owner?

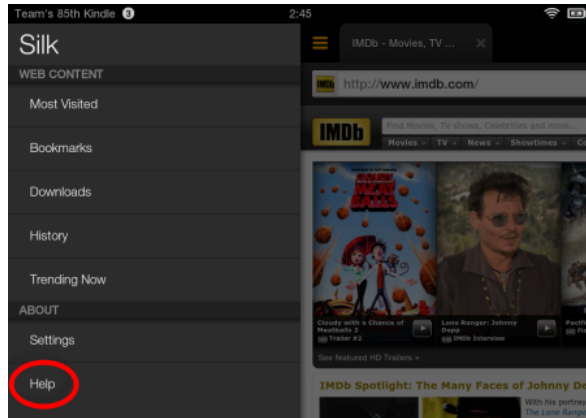
This guide provides an overview of the Amazon Silk web browser, with a focus on features and behaviors that are relevant to site owners and web developers. If you're wondering how to optimize your content for Silk, this is the place to start. If you're wondering about supported features and standards, you can find that information here too.

For tips on developing and optimizing web content for Silk, see the sections below:

- This section, [Getting Started with Amazon Silk \(p. 3\)](#), describes the **Silk Settings** menu and some of the configuration options available. Here you can learn about the options Silk users have for customizing the browsing experience.
- At [Developing Web Content for Amazon Silk \(p. 6\)](#), you'll find information on media handling, screen resolution, Flash support, the Silk user agent string, and other topics relevant to web development and content optimization.
- At [HTML5 Support \(p. 27\)](#), you'll find an introduction to many of the HTML5 and CSS3 features supported by Silk. You'll also find markup and examples to give you an idea of how Silk implements W3C standards.

## Accessing the Amazon Silk User Guide

The Silk User Guide, available within the browser, explains how to browse the web with Amazon Silk. In the user guide, you can learn about customizing the Silk view, using bookmarks, using browser tabs, reading in Reading View, clearing history, downloading content, and other topics. To access the user guide, launch the Silk browser, open the left panel menu (swipe in from the left edge of the screen), and tap **Help**.

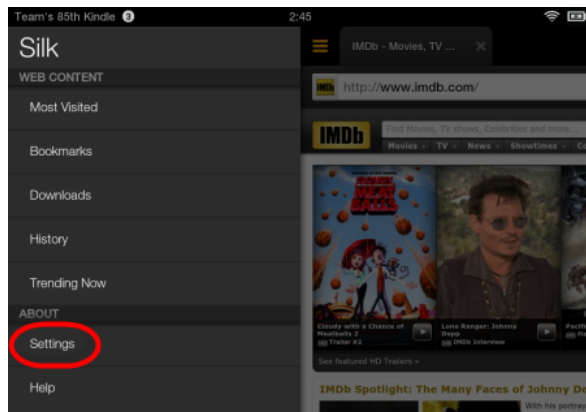


Off-device user documentation is also available on the [Fire, Kindle & Echo Support](#) site. Select your device on this page and look for a topic called *Silk Browser*, *Browse the Web* or something similar.

## Amazon Silk Settings Menu

The Silk Settings menu lets users change the default browser settings. User changes to these settings may affect the look and behavior of your site.

To get to the Silk Settings menu from the browser, open the left panel menu and tap **Settings**.



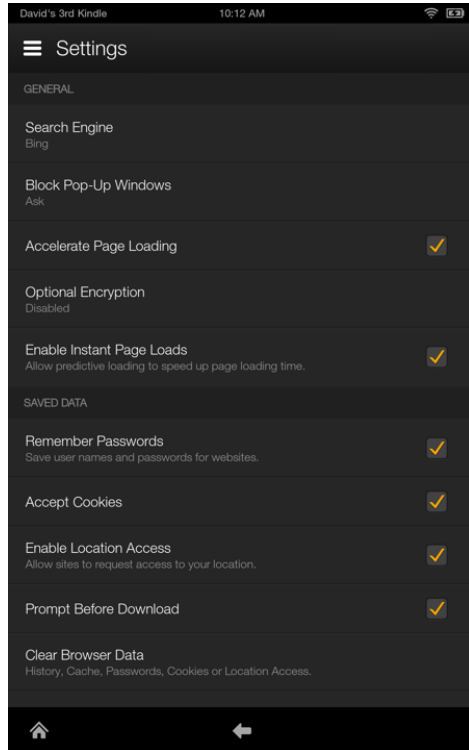
The Silk Settings menu lets users set the default search engine (**Bing**, **Google**, or **Yahoo!**), block pop-up windows, clear browsing history and cookies, disable location tracking, disable JavaScript, and make various other changes that affect how content is displayed.



## Amazon Silk Developer Guide

### Amazon Silk Settings Menu

---



Many of the Silk settings relate to standard browser functionality and will be familiar to web developers. Several settings are unique to Silk, or may be less familiar:

- **Cloud Features** – This setting can be used to disable cloud-enhanced browsing. By default, Silk accelerates page loading by routing some connections through the Amazon cloud, and syncs users bookmarks and settings.
- **Instant Page Load** – Enables Silk to use predictive loading to speed up page load times.

# Developing Web Content for Amazon Silk

---

Web browsers render content in different ways, depending on the platform, content type, browser settings, and other factors. The topics below discuss Amazon Silk features and functionality that can affect the delivery or display of content. You'll find tips on optimizing your website for Silk and also best practices that are applicable to web development in general. To learn about Silk support for HTML5, see [HTML5 Support](#) (p. 27).

## Topics

- [Responsive Web Design](#) (p. 6)
- [Amazon Silk User Agent Strings](#) (p. 12)
- [Feature Detection](#) (p. 14)
- [Media Handling](#) (p. 16)
- [Touch](#) (p. 17)
- [Screen Resolution](#) (p. 21)
- [Secure Connections](#) (p. 22)
- [JavaScript](#) (p. 22)
- [Caching](#) (p. 23)
- [Remote Debugging](#) (p. 24)
- [Do Not Track](#) (p. 25)

## Responsive Web Design

### Topics

- [How Does Responsive Web Design Work?](#) (p. 8)
- [Responsive Design and Amazon Silk](#) (p. 9)
- [The Takeaway](#) (p. 11)
- [Additional Resources](#) (p. 11)

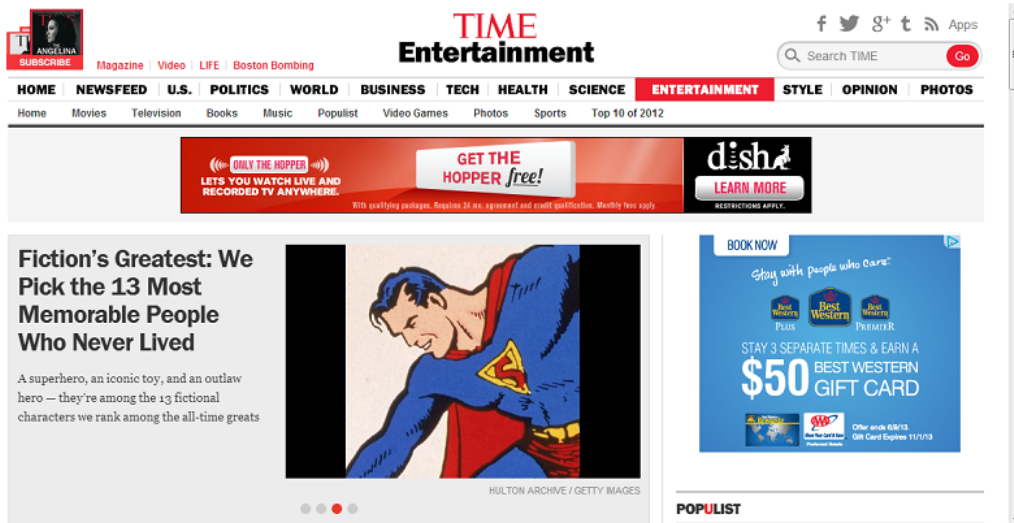
Responsive web design is, among other things, a solution to the rapidly increasing demand for websites that look good across a range of display sizes. A few years ago, web designers only had to think about

## Amazon Silk Developer Guide Responsive Web Design

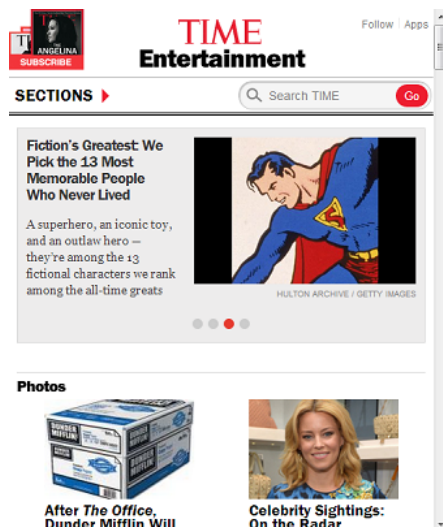
desktop monitors and—perhaps as an afterthought—smart phones. Now designers can count on site visits from mobile, tablet, netbook, laptop, and desktop clients in many different resolutions and aspect ratios. There's device orientation to think about, too, and the size of the browser window (not all users maximize it). With all of these variables, creating targeted content for every popular device is an unwieldy proposition, and one that's unlikely to scale well as device types continue to proliferate.

Fortunately, Ethan Marcotte has pointed to an alternative in his seminal article [Responsive Web Design](#). Instead of creating content for specific devices, we can design sites with flexible layouts that adjust to the size of the viewport (the area that the browser uses to draw the page). For example, the layout of [TIME.com](#) alters with the size of the browser window. At certain breakpoints, as the width of the window decreases, the top navigation disappears and the layout is simplified.

Here's the TIME.com site at desktop scale.



And here's the site with the browser window reduced in size, as it would be on a mobile device.



The site adapts to the changing width of the browser window, moving content and transforming the navigation bar to a pop-out Sections menu.

Responsive web design makes life easier for web designers, who don't have to create a custom layout for every new device on the market. But it's ultimately about creating a better experience for the user, who no longer has to constantly resize, pan across, or otherwise adjust the viewport. In a good responsive design, the layout adapts to the size of the viewport while maintaining the intended proportionality among the various design elements.

Content, too, can be optimized for users. By thinking in terms of a content hierarchy, you can ensure that users easily find what they're looking for on any device. For example, at a certain pixel width, you might decide to collapse some content in order to make other content more discoverable.

Of course, in some cases, it may be better to have a separate mobile site, instead of a responsive site. For example, if you want to divide each of your desktop pages into multiple mobile pages, a single set of responsively designed pages is probably not going to be the best solution. But if you design with the "mobile first" axiom in mind, you may not need to restrict the content available to mobile users.

## How Does Responsive Web Design Work?

Implementing a responsive design involves a combination of web development techniques and technologies. The following can all be part of a responsive design:

### Media Queries

[Media Queries \(p. 46\)](#) provide a way to conditionally apply CSS rules. The attributes `min-width` and `max-width` are especially useful, because they let you apply styles according to the minimum or maximum width of the viewport. For example, the following query sets styles for a viewport of 600 to 1000 pixels wide:

```
@media screen and (min-width: 600px) and (max-width: 1000px) { ... }
```

You can also set `device-width` and `orientation` in media queries. Using the logical operators `and`, `not`, and `only`, you can combine media queries for more finely tuned styling.

The conditional logic of a media query can be applied in several ways. You can embed a media query in a link to a style sheet, so that the style sheet is applied to the markup only if the indicated conditions are met. You can use a media query within an `@import` rule, which imports styles from other style sheets. And you can put media queries directly into a style sheet, as we've done in the Silk examples below.

### Fluid Layout

A fluid layout (also referred to as a fluid grid or liquid layout) keeps individual elements of the web page proportional across various display sizes. Conceptually, the opposite of a fluid layout would be a fixed-width layout in which the sizes of various elements are specified in pixels. The key to creating a fluid layout is to use relative rather than absolute values. A value of 80% can scale proportionally; a value of 600 pixels can't. To create a fluid layout, think percentages and ems (one em is equal to the current font size), not pixels.

To calculate the proportional size of layout elements (text or containers, measured in ems or percentages), use the following formula:  $target / context = result$ . For example, given a main content area of 800 pixels (the *context*), divided into a left column of 500 pixels (the *target*) and a right column of 300 pixels, you set the left column to 62.5% of the main content area ( $500 / 800 = 0.625$ ).

As you consider the technical aspects of creating a fluid layout, you'll probably also want to give some thought to your content hierarchy. Using a fluid layout and breakpoints, you can reorder content containers as the viewport changes size. There's no one right way to design a fluid layout, but it's a good idea to make important content easy to find.

### Context-Aware Images

Context-aware images, or flexible images, are an important part of a fluid layout. The idea is pretty simple. An image with absolute width and height can't scale with a fluid grid. But an image with its `max-width` property set to 100% can scale down for smaller viewports. To save bandwidth, you can

also serve different images according to screen resolution. That way you don't force the client to load a larger image than necessary. The `overflow:hidden` property can also help size images appropriately, by cropping them. Keep in mind that enlarging an image beyond its original dimensions can result in perceptible loss of quality.

### Viewport Meta Element

The viewport is the area within which the browser draws a page, excluding the browser chrome. You can use the [viewport Meta Element \(p. 49\)](#) to ensure that the viewport width is the same as the screen width. This is important for mobile devices, because mobile browsers often try to render an entire desktop-scale page within a mobile viewport. While this zoomed-out view lets the user see the full page without swiping, it also effectively miniaturizes content. Moreover, media queries may not work as expected if the viewport tag isn't implemented properly.

To avoid problems, include the `viewport` meta element in the `head` section of your HTML document, and set `width` to `device-width`:

```
<meta name="viewport" content="width=device-width">
```

This setting prevents browsers from zooming out to fit more pixels in the viewport. The pixel width available to the viewport will be the same as the pixel width of the device screen.

### Breakpoints

A breakpoint represents a particular viewport width at which the layout changes. A breakpoint occurs when the styling specified by one media query gives way to the styling of another media query. In this sense, a breakpoint is not so much a single line of code as it is a design abstraction. One strategy for creating breakpoints is to target common device widths so that your layout is optimized for specific devices.

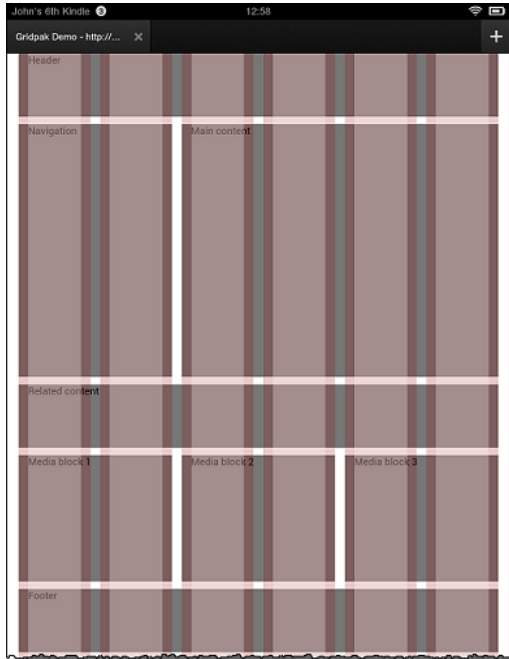
Another strategy (arguably a better one) is to create breakpoints based on your content, and not on device constraints. Designing breakpoints around content aims to make the layout user friendly and aesthetically pleasing across a range of possible viewport widths. With this approach, you can minimize awkward layouts that fall between device widths, and you don't have to plan for as many one-off scenarios (tablet "X" in portrait, tablet "Y" in landscape, and so on). After all, the best outcome is to have content that looks great at any display size.

## Responsive Design and Amazon Silk

Let's see how Silk renders a responsive design template.

We'll use the [Gridpak app](#) to generate a layout that adapts to orientation changes. The Gridpak app lets us configure columns and set breakpoints in a web UI and then download a package of styles and scripts. We'll use a slightly modified version of the Gridpak demo site to try out Silk.

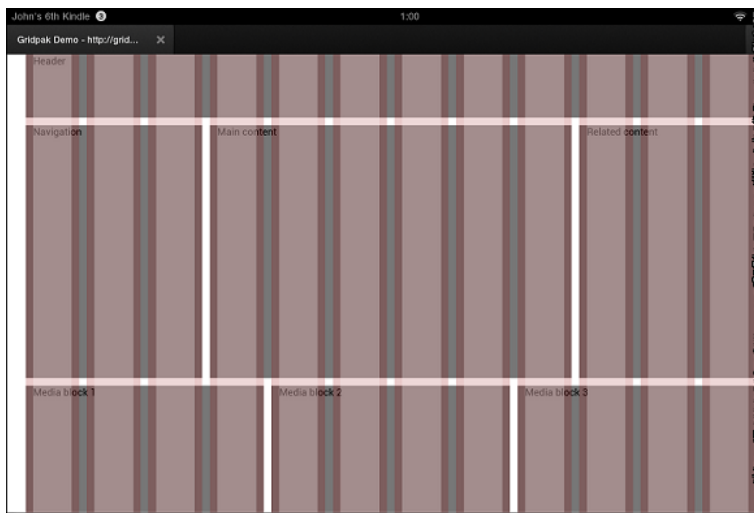
Here's our demo site in portrait view on a Fire HD 8.9" tablet:



The following media query and the associated styles and scripts (which aren't shown) produce a 6-column grid when the viewport is between 320 and 800 pixels wide.:

```
@media screen and (min-width: 320px) and (max-width: 800px) {  
/* styles for 6-column grid */  
}
```

Now here's our demo site in landscape view:



In this case, the following media query, plus the omitted styles and scripts, produce a 12-column grid when the viewport is more than 800 pixels wide.

```
@media screen and (min-width: 801px) {  
/* styles for 12-column grid */  
}
```

We haven't had to reference device orientation in our media queries. We just specify width ranges. To make this layout work correctly on Silk, we also need to add a viewport meta tag to our HTML:

```
<meta name="viewport" content="width=device-width">
```

Here we've made `width` equal to `device-width`. This enables Silk to respond as expected to changes of device orientation. We could also configure the viewport scale using `maximum-scale`, `initial-scale`, and `user-scalable`. For example, the following configuration loads the site at the scale of the viewport and lets the user zoom in to triple the scale of the viewport:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=3.0">
```

A `minimum-scale` setting is also available, but with `width` set to `device-width`, the user won't be able to zoom out past the width of the device.

## The Takeaway

So what's the takeaway for you, the site owner or web designer? Well, it's pretty simple. You don't need to develop separate content for Silk. Of course, we recommend that you test your site on a variety of devices. But with responsive web design, you don't have to implement a new design tactic for every device on the market. Instead, you can focus on developing content that looks great on any device. Silk users will benefit from your efforts, and so will all the rest of your customers.

## Additional Resources

- [Targeting Screens from Web Apps](#) in the *Android Developers Guide*
- Ethan Marcotte, [Responsive Web Design](#) in *A List Apart*
- Grace Smith, [85 Top Responsive Web Design Tools](#) in *Mashable*
- Rahul Lalimalani's series on responsive web design in *MSDN Magazine*:
  - [Why the Web Is Ready for Responsive Web Design](#)
  - [Designing Experiences for Responsive Web Sites](#)
  - [Common Techniques in Responsive Web Design](#)
- Katrien De Graeve, [Responsive Web Design](#) in *MSDN Magazine*
- [Responsive Web design](#) in the *Mozilla Developer Network*
- Rudy Rigot and Sophie Taboni, [Responsive web design: a project-management perspective](#) at *Dev.Opera*
- Chris Mills, [Love your devices: adaptive web design with media queries, viewport and more](#) at *Dev.Opera*
- Andreas Bovens, [An introduction to meta viewport and @viewport](#) at *Dev.Opera*
- Examples of responsively designed sites in [Media Queries](#)

## Amazon Silk User Agent Strings

In the User-Agent request header field, Amazon Silk sends one of three user agent strings, depending on the view requested on the device by the customer. If a specific view is not requested, the view defaults to Tablet for Fire tablets, and Mobile for Fire phones. If a site has compatibility issues when rendered in the default view, a different view may be selected by the device. The templates for the Amazon Silk user agents are shown below.

### Tablet

```
Mozilla/5.0 (Linux; U; Android android-version; locale; product-model  
Build/product-build) AppleWebKit/webkit-version (KHTML, like Gecko)  
Silk/browser-version like Chrome/chrome-version Safari/webkit-version
```

### Desktop

```
Mozilla/5.0 (X11; Linux x86_64; U; locale) AppleWebKit/webkit-version (KHTML,  
like Gecko)  
Silk/browser-version like Chrome/chrome-version Safari/webkit-version
```

### Mobile

```
Mozilla/5.0 (Linux; U; Android android-version; locale; product-model  
Build/product-build) AppleWebKit/webkit-version (KHTML, like Gecko)  
Silk/browser-version like Chrome/chrome-version Mobile Safari/webkit-version
```

### Note

Red and italics indicates variable fields, which are described below.

The [Amazon Silk UA for Kindle Fire 1st Generation](#) (p. 13) follows a slightly different syntax.

### Template fields

- *locale* – Indicates the chosen language and country or region for the tablet. Locale is in hyphenated lowercase format, as in en-us (US English). For Kindle Fire (1st Generation), locale is always en-us.
- *product-model* – The value of `Build.MODEL` (for example, KFTT).
- *android-version* – The [Android platform version](#) (for example, 4.0.3).
- *product-build* – The value of `Build.ID` (for example, IML74K).
- *webkit-version* – Indicates the version of WebKit used (for example, 535.19). This value can change whenever the Kindle Fire receives a software update.
- *browser-version* – Indicates the version of the Amazon Silk browser, in the format major.minor (for example, 2.1). The browser version can change whenever the Kindle Fire receives a software update.
- *chrome-version* – The version of Google Chrome with which the Amazon Silk browser is compatible.

### Related product models

- **Kindle Fire** – KFOT
- **Kindle Fire HD** – KFTT
- **Kindle Fire HD 8.9"** – KFJWI
- **Kindle Fire HD 8.9" 4G** – KFJWA
- **Kindle Fire HD 7" (3rd Generation)** – KFSOWI



- **Kindle Fire HDX 7" (3rd Generation)** – KFTHWI
- **Kindle Fire HDX 7" (3rd Generation) 4G** – KFTHWA
- **Kindle Fire HDX 8.9" (3rd Generation)** – KFAPWI
- **Kindle Fire HDX 8.9" (3rd Generation) 4G** – KFAPWA
- **Fire HD 6 (4th Generation)** – KFARWI
- **Fire HD 7 (4th Generation)** – KFASWI
- **Fire HDX 8.9 (4th Generation)** – KFSAWI
- **Fire HDX 8.9 (4th Generation) 4G** – KFSAWA
- **Fire Phone** – SD4930UR

## User Agent String Examples

### Examples of the Amazon Silk User Agent String

#### Tablet

```
Mozilla/5.0 (Linux; U; Android 4.2.2; en-us; KFTHWI Build/JDQ39) AppleWebKit/537.36 (KHTML, like Gecko) Silk/3.22 like Chrome/34.0.1847.137 Safari/537.36
```

#### Desktop

```
Mozilla/5.0 (X11; Linux x86_64; U; en-us) AppleWebKit/537.36 (KHTML, like Gecko) Silk/3.22 like Chrome/34.0.1847.137 Safari/537.36
```

#### Mobile

```
Mozilla/5.0 (Linux; U; Android 4.2.2; en-us; KFTHWI Build/JDQ39) AppleWebKit/537.36 (KHTML, like Gecko) Silk/3.22 like Chrome/34.0.1847.137 Mobile Safari/537.36
```

### Amazon Silk UA for Kindle Fire 1st Generation

In the examples below, red and italic text indicates variable fields. The Amazon Silk-Accelerated value can be either `true` or `false`, depending upon customer selection.

#### Desktop/Tablet

```
Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_3; en-us; Silk/1.0.13.81_10003810) AppleWebKit/533.16 (KHTML, like Gecko) Version/5.0 Safari/533.16 Silk-Accelerated=true
```

#### Mobile

```
Mozilla/5.0 (Linux; U; Android 2.3.4; en-us; Silk/1.0.13.81_10003810) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1 Silk-Accelerated=true
```

## User Agent Detection

You can use JavaScript to detect the Amazon Silk user agent across various Kindle Fire device families. Unless you want to provide a unique experience for different Kindle Fire devices, we recommend a general match that will work over time as product models and version numbers change. The following examples illustrate best practices for matching the Amazon Silk user agent across Kindle Fire device types.

### Note

The tests below also match the user agent string for the PlayStation Vita browser. If this is a concern, you can exclude that browser using the following condition:

```
!/Playstation/.test(navigator.userAgent)
```

### To detect Amazon Silk

```
if (/bSilk\b/.test(navigator.userAgent)) {  
    alert("Silk detected!");  
}
```

### To detect the Amazon Silk version

```
var match = /bSilk\/([0-9._-]+)\b/.exec(navigator.userAgent);  
if (match) {  
    alert("Detected Silk version "+match[1]);  
}
```

### To detect mobile/desktop preference

```
var match = /bSilk\/(.*bMobile Safari\b)?/.exec(navigator.userAgent);  
if (match) {  
    alert("Detected Silk in mode "+(match[1] ? "Mobile" : "Default (desktop)"));  
}
```

### To detect multiple variables at once

```
var match = /(?:; ([^;]+) Build\/.*)?bSilk\/([0-9._-]+)\b(.*bMobile Sa  
fari\b)?/.exec(navigator.userAgent);  
if (match) {  
    alert("Detected Silk version "+match[2]+" on device "+(match[1] || "Kindle  
Fire")+ " in mode "+(match[3] ? "Mobile" : "Default (desktop)"));  
}
```

## Feature Detection

When you build a website, you want to deliver the best possible experience to all of your customers, no matter what browser and platform they're using. You can do this with feature detection. Feature detection is a content-delivery strategy predicated on feature availability, not browser functionality. Instead of

checking to see if a customer is using "browser X version 1.1" and then assuming that this version of browser X supports some feature, you test for the feature directly and serve content accordingly.

**Note**

It's also possible to target content via user agent detection. But user agent detection can be problematic. It requires you to keep track of the browsers your customers use and the features that those browsers support. Those variables will change over time, so user agent detection isn't future proof. User agent detection can be useful if feature detection is expensive or if a particular feature is only partially implemented by a browser. But in most cases, feature detection is the right choice. To learn more about the Silk user agent, see [Amazon Silk User Agent Strings \(p. 12\)](#).

## How To Detect a Feature

There are several ways to detect features. You can write your own feature detection scripts, or you can use a helper library. Both methods are demonstrated below.

### Test for Feature Support

The following code detects support for the HTML5 canvas element by testing one of the element's attributes (`getContext`). The attribute test is important because browsers can create elements that they don't actually support. If we can interact with an attribute, that gives us more information.

```
<!DOCTYPE html>
<html>
  <body>
    <div>Test support for the HTML5 canvas element.</div>
    <script>
      function supportsCanvas() {
        return !!document.createElement('canvas').getContext;
      }
      if (supportsCanvas()) {
        alert('Your browser supports the canvas element.');
```

If our test for the `getContext` method returns true, we assume that canvas is supported. The test uses double negation `!!` to force a boolean value (true or false). For more on this canvas test and on the Modernizr test below, see [Canvas](#) in Mark Pilgrim's [Detecting HTML5 Features](#).

### Test for a Feature with Modernizr

You can also detect features by using [Modernizr](#), a JavaScript library that identifies support for HTML5 and CSS3 features. It minimizes the amount of code you have to write. You can build a custom Modernizr library to test for specific features, but the example below just links to the [Modernizr Development Library](#), which works for testing purposes.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://modernizr.com/downloads/modernizr-latest.js"></script>

  </head>
  <body>
```

```
<div>Use Modernizr to test support for the HTML5 canvas element.</div>
<script>
if (Modernizr.canvas) {
    alert('Your browser supports the canvas element.');
```

Here we use the Modernizr object to test for canvas support. If `Modernizr.canvas` is true, the browser supports the canvas element.

Of course, in a real application, we actually need to serve content for the supported feature. And if the feature isn't supported, we need to have a fallback. Polyfills can help with this. Polyfills are JavaScript shims that approximate native browser functionality. Modernizr maintains a listing of [HTML5 Cross Browser Polyfills](#).

## Additional Resources

- [HTML5 Rocks: Feature, Browser, and Form Factor Detection: It's Good for the Environment](#)
- [A List Apart: Taking Advantage of HTML5 and CSS3 with Modernizr](#)
- [Dive Into HTML5: Detecting HTML5 Features](#)

## Media Handling

Amazon Silk supports a wide range of media types.

### Topics

- [Images \(p. 16\)](#)
- [Audio \(p. 17\)](#)
- [Video \(p. 17\)](#)

## Images

In addition to all the expected image formats, Amazon Silk supports scalable vector graphics (SVG) both inline via the `<svg>` tag and externally using the `<img>`, `<object>`, and `<embed>` tags. For more information, see [SVG Element \(p. 45\)](#).

### Image Optimization

Though not specific to Amazon Silk, the following tips can help you optimize images for your web site.

- **Set the width and height attributes:** It's usually a good idea to set the width and height attributes explicitly on images. Browsers use the width and height attributes to determine the amount of space that an image needs in the page layout. If these attributes are not set, or if they don't match the actual dimensions of the image, the browser may have to reflow the page, which can increase page load time.
- **Remove excess white space:** If you want to add padding around an image, you're better off doing it with CSS than with extra white space around the image itself. Extra space in an image increases file size.

- **Use the right file format:** Using the correct file format for a given color palette and use case can help you minimize file size. As a general rule, use GIF for animations, GIF or PNG-8 for flat graphics or where transparency is needed, and JPG or PNG for photos and colorful, detailed images.
- **Consider using an image optimizer:** Image optimizers can reduce the size of image files. Available image optimization tools include [TinyPNG](#) and the [Grunt imagemin plugin](#).

For more information on image optimization, see [Optimizing web graphics](#).

## Audio

Amazon Silk fully supports the HTML5 [Audio Element \(p. 41\)](#), with playback occurring in the browser. The default controls are improved from earlier Kindle Fire devices, and Silk also supports custom controls; playback of multiple streams; and MP3, OGG, and WAV audio formats. JavaScript support is good, enabling, for example, algorithmic PCM Audio using data URIs.

Amazon Silk doesn't support background audio, meaning that audio playback is paused when a tab loses focus.

## Video

Amazon Silk supports the HTML5 `video` tag, and MP4 and WEBM video formats. If a tag contains multiple formats, MP4 is given priority. Note that the audio encoding in video streams must be at most stereo. The Fire family of devices doesn't support 5.1 audio.

Amazon Silk also supports the RTSP and HLS network protocols. For more information about media formats that the Android platform supports, see [Android Supported Media Formats](#).

## Graceful Degradation with the Video Element

It's usually a good idea to use [Feature Detection \(p. 14\)](#) for features and media formats that aren't supported by all browsers. To detect video support, you can also try a simpler solution: graceful degradation with the HTML5 video element, which doesn't require JavaScript. In the example below, the video element provides a built-in fallback structure that lets you easily deliver multiple video container formats.

```
<video controls>
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <object data="video.mp4" width="320" height="240">
    <embed src="player.swf" width="320" height="240">
  </object>
</video>
```

The example tries to serve MP4 and WebM files and then falls back to a Flash player. The browser will play the first compatible format. For more on graceful video degradation, see [Video for Everybody](#); for more on HTML5 video, see [Video Element \(p. 45\)](#).

## Touch

As you might expect, Amazon Silk is designed for touch interactions and fully supports [Touch Events \(p. 36\)](#). In general, Amazon Silk and other touch browsers know how to apply touch interactions to click-based DOM events, even with no mouse involved. For example, Silk will fire the following `onclick` event in response to a tap.

```
<html>
  <body>
    <button onclick="testFunction()">Tap me</button>
    <div id="test"></div>
    <script>
      function testFunction() {
        document.getElementById("test").innerHTML="The tap fired an onclick
event.";
      }
    </script>
  </body>
</html>
```

In many web development scenarios, you can rely on this sort of compatibility between mouse events and a touch interface. But touch gestures are more than just a substitute for mouse interactions. Touch is an integral part of the web browsing experience on a tablet, and if you want to go beyond parity with desktop functionality—that is, if you want to provide an experience tailored to a tablet or mobile form factor—providing added support for touch gestures is a good place to start.

**Note**

Designing for touch events can also provide a better experience for trackpad users.

The W3C makes several [recommendations for touch-based interactions](#):

- The spacing of selectable elements should be wide enough to accommodate direct selection.
- The screen size of selectable elements should be sufficient to accommodate direct selection.
- Because elements have to be selected to be in focus, you should be careful not to hide information behind a focus state.

The first two recommendations address the "fat finger" problem. If selectable page elements are small and densely placed, it's easy to select the wrong one. The solution is to design navigation elements that are large enough to be easily selected by touch.

The third item pertains to information that's intended to be hidden until an element comes into focus. The issue is that, with touch interactions, it can be problematic to distinguish between focusing on an element and selecting an element. This issue is particularly important when it comes to drop-down menus. On desktop, a user can trigger the `:focus` or `:hover` pseudoclasses by mousing over an element. Thus, for desktop you can create a drop-down navigation that displays child elements on mouseover. With a touch device, such hover menus are more complicated. To learn more about creating hover menus for Amazon Silk, see [Create Drop-down Menus for a Touch Screen \(p. 53\)](#)

## Touch Example Using jQuery Mobile

Consider a few examples of DOM events specifically designed for touch. These examples use [jQuery Mobile](#), as it provides good touch support. jQuery Mobile is a web development framework designed to make HTML5-compatible, responsive, touch-optimized websites. The example below specifies targets and create handlers for two touch events: `tap` and `taphold`.

**Note**

This example includes the jQuery libraries from a content delivery network, so you should be able to paste the code below directly into a page and run it without downloading any additional resources.

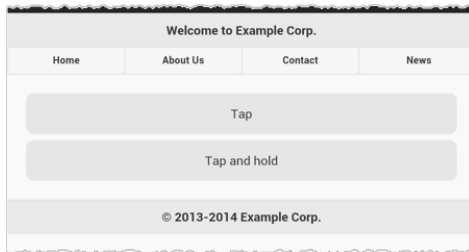
```
<html>
  <head>
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-
scalable=no">
<style>
div.test {
  text-align: center;
  border-radius: .625em;
  background-color: #E6E6E6;
  padding: 1em;
  margin: .5em;
}
</style>
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.2/jquery.mo
bile-1.4.2.min.css" />
</head>
<body>
<div data-role="page" id="home">
  <div data-role="header">
    <h1>Welcome to Example Corp.</h1>
    <div data-role="navbar">
      <ul>
        <li><a href="#home" class="ui-btn ui-btn-inline ui-corner-
all">Home</a></li>
        <li><a href="#about" class="ui-btn ui-btn-inline ui-corner-all">About
Us</a></li>
        <li><a href="#contact" class="ui-btn ui-btn-inline ui-corner-
all">Contact</a></li>
        <li><a href="#news" class="ui-btn ui-btn-inline ui-corner-
all">News</a></li>
      </ul>
    </div>
  </div>
  <div data-role="main" class="ui-content">
    <div class="test tap">Tap</div>
    <div class="test taphold">Tap and hold</div>
  </div>
  <div data-role="footer">
    <h1>&copy; 2013-2014 Example Corp.</h1>
  </div>
</div>
<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
<script src="http://code.jquery.com/mobile/1.4.2/jquery.mobile-
1.4.2.min.js"></script>
<script>
$(function(){
  $("div.test.tap").on("tap", function() {
    $(this).text("After tap event").css("background-color", "#86BDFF");
  });
  $("div.test.taphold").on("taphold", function() {
    $(this).text("After tap-and-hold event").css("background-color",
"#FFBF4C");
  });
});
</script>
</body>
</html>
```

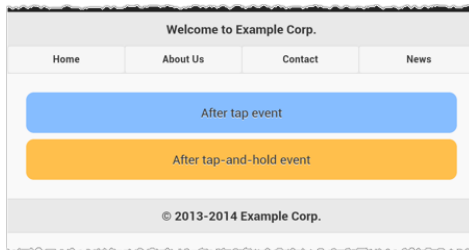
Both the `tap` and `taphold` events are jQuery Mobile extensions of jQuery built-in methods. The `tap` event is fired by a quick touch event on an element. The `taphold` event is fired by a touch-and-hold event (that is, a long press) on an element. (By default, the jQuery Mobile `taphold` event is triggered by a sustained tap of 750 milliseconds, but you can change this.)

We could also have implemented other jQuery Mobile touch events, like [swipe](#), [swipeleft](#), and [swiperight](#). The code above sets two anonymous callback functions, one on the `div.test.tap` selector, and one on the `div.test.taphold` selector. In each case, we're binding a target (`div.test.tap` or `div.test.taphold`) to an event (`tap` or `taphold`), and in each case the event handler modifies the CSS applied to the target element.

Here's what the page looks like immediately after page load:



And here's the page after the tap and tap-and-hold events.



The example above illustrates the beginning of a touch-friendly web design. Even in portrait mode, the top navigation links are wide enough to accommodate touch interactions comfortably. jQuery Mobile makes it especially easy to create such a touch-friendly design, and there are other options, too. [Bootstrap](#) is a design framework that provides good touch support, and there are various plugins available for improving touch support.

So here are the takeaways:

- Design and develop with touch interactions in mind. Even if your site or application is not specifically targeted for touch devices, you should provide a good experience for all users.
- Frameworks like jQuery Mobile and Bootstrap can help you provide a great experience to touch-device users. These frameworks won't be useful for every project, but creating touch-friendly interactions is important for every site.

## Additional Resources

- [W3C Touch Events Recommendation](#)
- [Multi-touch Web Development](#)
- [Touch events](#)
- [Touch And Mouse: Together Again For The First Time](#)



## Screen Resolution

When you're developing web content for Amazon Silk (or any other mobile browser), it's a good idea to be aware of devices' screen resolution and related specifications. The following table shows screen size, resolution, and scale factor for Fire tablets and phone.

Device	Screen size	Screen resolution (px)	Scale factor
Fire HDX 8.9 (4th Gen)	8.9-inch screen	2560 x 1600 (supports 1080p HD resolution)	2.0 (xhdpi)
Fire HD 7 (4th Gen)	7-inch screen	1280 x 800 (supports 720p HD resolution)	1.5 (hdpi)
Fire HD 6 (4th Gen)	6-inch screen	1280 x 800 (supports 720p HD resolution)	1.5 (hdpi)
Fire Phone	4.7-inch screen	1280 x 720 (supports 720p HD resolution)	2.0 (xhdpi)
Kindle Fire HDX 8.9" (3rd Gen)	8.9-inch screen	2560 x 1600 (supports 1080p HD resolution)	2.0 (xhdpi)
Kindle Fire HDX 7" (3rd Gen)	7-inch screen	1920 x 1200 (supports 1080p HD resolution)	2.0 (xhdpi)
Kindle Fire HD 7" (3rd Gen)	7-inch screen	1280 x 800 (supports 720p HD resolution)	1.5 (hdpi)
Kindle Fire HD 8.9" (2nd Gen)	8.9-inch screen	1920 x 1200 (supports 1080p HD resolution)	1.5 (hdpi)
Kindle Fire HD 7" (2nd Gen)	7-inch screen	1280 x 800 (supports 720p HD resolution)	1.5 (hdpi)
Kindle Fire (2nd Gen)	7-inch screen	1024 x 600	1.0 (mdpi)
Kindle Fire (1st Gen)	7-inch screen	1024 x 600	1.0 (mdpi)

You can use the Silk user agent strings to detect a particular Fire device and target the user experience accordingly. For more information about the Silk user agent string, see [Amazon Silk User Agent Strings \(p. 12\)](#).

Here are a few additional things to keep in mind when developing web content for Silk and Fire devices:

- The viewport is the portion of the browser dedicated to displaying the webpage. Viewport size and screen size for mobile devices are not necessarily identical. They differ because the browser uses up some screen real estate to show its chrome. You can use the [viewport meta element \(p. 49\)](#) to specify attributes of the viewport, including width and height.
- With a mobile form factor, HTML forms can be challenging for users to complete. You can use [HTML5 input types \(p. 43\)](#) to make forms more responsive.
- You can use [media queries \(p. 46\)](#) to style your site for a specific screen resolution.

For more information about Fire device specifications, see the [Device and Feature Specifications](#) (tablets) and [Fire Phone Specifications](#) pages on the Amazon Apps & Games Developer Portal. To learn more about scale factor, see [Screen Layout and Resolution](#).

## Secure Connections

Secure Sockets Layer (SSL) is a cryptographic protocol that provides security for online communications. Amazon Silk supports SSL communication between the device client on the Kindle Fire and origin servers. For enhanced privacy and security, SSL traffic is not routed through Silk remote proxies in the Amazon Cloud, and we don't collect any metrics regarding web page resources downloaded using SSL connections.

## JavaScript

Amazon Silk uses the [V8 JavaScript engine](#) to compile and execute JavaScript.

Using the Amazon Silk **Settings** menu, users can enable or disable JavaScript. The setting is enabled by default.

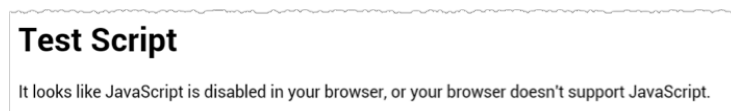


To learn more about the Settings menu, see [Amazon Silk Settings Menu \(p. 4\)](#).

When JavaScript is disabled, Silk ignores the content of `<script>` tags. You can use the `<noscript>` tag to let users know that JavaScript content is disabled.

```
<html>
  <body>
    <h1>Test Script</h1>
    <script>document.write("JavaScript is enabled in your browser.")</script>
    <noscript>It looks like JavaScript is disabled in your browser, or your
browser doesn't support JavaScript.</noscript>
  </body>
</html>
```

Here's the output of the above markup rendered by Amazon Silk with JavaScript disabled:



## JavaScript Loading

Though not specific to Amazon Silk, the following recommendations for loading JavaScript can improve the performance of your website.

- Wherever practical, combine multiple external script files into a single file to improve page load time.
- Avoid including duplicate scripts on a page.
- Include external CSS files before external scripts. Otherwise script loading may block style sheet loading. And include script files as far down the page as possible. HTML parsing is blocked by the downloading and execution of scripts.
- Don't put inline scripts between included CSS files and other resources. Like an external script, an inline script can prevent a CSS file from downloading in parallel to other resources. In general, it's best to avoid inline scripts altogether and instead include external JavaScript files.

- Use minified JavaScript. Minified JavaScript files are smaller and therefore download faster. Available tools for compressing JavaScript include [UglifyJS](#), [Closure Compiler](#), and [YUI Compressor](#).
- Consider using the `async` and `defer` attributes in your script elements. With `async` set, a script is downloaded asynchronously and then executed, blocking HTML parsing. With `defer` set, a script is downloaded asynchronously and then executed after HTML parsing has completed.

To learn more about improving load times for scripts, see the following resources:

- [Properly including stylesheets and scripts](#)
- [14 Rules for Faster-Loading Web Sites](#)
- [Deep dive into the murky waters of script loading](#)

## Caching

Caching on the Amazon Silk back end is one of the primary mechanisms Silk uses to accelerate page loading. The back end (i.e. the remote proxy service) only caches page elements that you explicitly mark as cacheable. You can ensure that your site is optimized to take advantage of this caching by explicitly setting at least one of the following HTTP cache-control headers:

- `max-age` indicates that the client will only accept a response whose age is no greater than the specified time in seconds.
- `Expires` indicates the date after which the response is considered stale. Normally, a stale cache entry won't be returned unless it's first validated with the origin server or with an intermediate cache.
- `public` indicates that the response can be cached by any cache, even if it would normally be noncacheable.

The longer the cacheable lifespan of a resource, the more benefit Silk users receive.

Silk currently obeys all standard caching semantics, both on the device client and on the remote proxy server.

Silk's distributed cache will cache anything with proper headers, so you can derive substantial benefit from adding proxy caching headers.

### Note

Silk won't cache resources on the proxy for an HTTPS connection or for a request that sends cookies, but these resources can be cached on the device.

Caching headers can have multiple restriction levels:

- `no-cache` indicates neither the device nor the proxy can cache the resource.
- `private` indicates the device can cache the resource, but the proxy cannot.
- `public` indicates both the device and the proxy can cache the resource.

We recommend setting the caching level to `public`, because resources set at this level are included in the Silk cache. This means that devices will receive their content faster from our cache, and the traffic to origin servers will be substantially reduced, since there's only a single resource request per Availability Zone per day for each cacheable resource for the entire on-cloud Fire device population.

# Remote Debugging

Amazon Silk provides remote debugging through Chrome DevTools. With remote debugging, you can use your development machine to inspect and interact with pages displayed in Silk on a separate device. Remote debugging has to be enabled on the device, and the Android Debug Bridge (ADB) and Google Chrome have to be installed on your development machine.

To set up remote debugging, follow the steps below.

## Install Chrome and ADB

1. If you don't have the Chrome browser installed on your development machine, download it from the [Google Chrome download page](#).
2. ADB is automatically installed with the Android SDK. To install the SDK, go to the [Get the Android SDK](#) site, download the bundle, and follow the setup instructions.

## Enable remote debugging

1. Swipe down from the top of the screen to open the main menu and locate the ADB setting on the device. The location of this setting varies depending on which version of Fire OS the device is running.
  - **Fire OS (based on Android 4.0.3)** – Select **More > Security**, and set **Enable ADB** to **ON**.  
This version of Fire OS is available on Kindle Fire tablets manufactured in 2012.
  - **Fire OS 3** – Select **Settings > Device**, and set **Enable ADB** to **ON**.  
This version of Fire OS is available on Kindle Fire tablets manufactured in 2013.
  - **Fire OS 4** – Select **Settings > Device Options** or **About > Developer Options**, and then set **Enable ADB** to **ON**.  
This version of Fire OS is available on Fire tablets manufactured in 2014. If you do not see the *Developer Options* option in the *Device Options* menu, tap the **Serial Number** field several times to turn on developer mode.

Read the message regarding security and confirm your selection by tapping **OK** or **Enable**.

2. Open the Silk browser and enter `about:developer` in the URL bar. A dialog opens, giving you the option to enable developer settings. (If entering `about:developer` doesn't open the dialog, Silk may need to be updated before you can use remote debugging.) If you want to proceed, tap **Enable**. Developer options appear at the bottom of the Settings menu.



Tap **Remote Debugging**. In the dialog, select the connection type that fits your development environment. Then connect as follows:

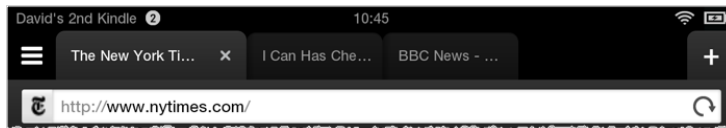
- **Unix Domain Socket** (only available on a Unix-based development machine) – Connect the device to the development machine via USB.
- **TCP** (available on Windows) – Ensure that both the device and the development machine are connected to the same network.

While the URL for the Unix Domain Socket is always `http://127.0.0.1:9222/`, the URL for TCP varies by device. Thus, if you're enabling a TCP connection, note the URL listed in the **Settings** menu.

## Debug your site

With Chrome and ADB installed, and remote debugging enabled on the device, you're ready to debug.

1. Using Silk on the device, navigate to the page you want to inspect. You can inspect multiple pages by opening each page in its own tab.



2. Follow the directions for your remote debugging configuration:

- **Unix Domain Socket:** To inspect open tabs, run the following:

```
adb forward tcp:9222 localabstract:com.amazon.cloud9.devtools
```

Then navigate to `http://127.0.0.1:9222/` in Chrome on your development machine.

- **TCP:** To inspect open tabs, ensure that both your device and development machine are connected to the same network. Then navigate to the remote debugging URL in Chrome on your development machine. This URL is listed in the **Settings** menu.

3. Pages that are open in Silk tabs are listed at the remote debugging address.



Open inspectable pages and interact with them using the developer tools. To learn more about inspecting pages with the developer tools, see [Chrome DevTools](#).

## Do Not Track

Amazon Silk supports the Do Not Track (DNT) header request field, an HTTP header field that specifies a user preference about data collection. Many web sites collect information about user browsing behavior, and with the DNT field users can opt out of such data collection.

Amazon Silk users configure their DNT preferences on the device, and these preferences are then communicated in the HTTP request header in the DNT field. A DNT header with a value of `1` indicates that the user prefers **not** to allow the target site to track the user's browsing behavior. A DNT header with a value of `0` indicates that the user has the default setting, which allows the target site to track the user's browsing behavior.

Here's a [W3C example](#) of an HTTP header with the DNT field set to `1`:

```
GET /something/here HTTP/1.1  
Host: example.com  
DNT: 1
```

The Do Not Track standard also allows for a window property to detect Do Not Track status inside JavaScript or other dynamic site content. Amazon Silk supports this property and will set it if the user has expressed a Do Not Track preference. The property is `navigator.doNotTrack`, and it will be set to “yes” if the user has selected Do Not Track, and “no” (the default) if the user has decided to permit tracking.

Note that the W3C specification for DNT behavior defines request and response headers for communicating tracking preferences, but it does not require sites to respect an expressed user preference.

## Additional Resources

To learn more about the Do Not Track standard, see the following resources:

- [Tracking Preference Expression](#)
- [Mozilla Developer Network: The Do Not Track Field Guide](#)
- [Mozilla Developer Network: Navigator.doNotTrack](#)
- [Do Not Track: Universal Web Tracking Opt Out](#)

# HTML5 Support

---



Amazon Silk provides broad support for the emerging HTML5 standard and for related media features. In this section of the *Silk Developer Guide*, you'll find information on feature support across Silk versions, and examples of supported features and APIs. Though the documentation aims to be comprehensive, the feature set described here is not all-inclusive. Look for HTML5 support to expand and improve as Silk evolves.

## Topics

- [Support for HTML5 and Related Features \(p. 27\)](#)
- [HTML5 APIs \(p. 29\)](#)
- [HTML5 Elements and Attributes \(p. 40\)](#)
- [CSS3 Support \(p. 46\)](#)

## Support for HTML5 and Related Features

The following table shows HTML5 feature support across Amazon Silk versions. The columns are divided by Amazon Silk generation (Gen1, Gen2, Gen3) and indicate the build version number at the time this page was last updated. If the WebKit browser prefix is required for a given feature, it's indicated by `-webkit-` in parentheses.

### Note

As the table below indicates, feature support for Silk Gen 1 differs significantly from Gens 2 and 3. In general, feature support for Silk Gen 1 is the same as that for [Android WebView 2.3](#).

Feature	Gen 1, v. 1.0.143.1	Gen 2, v. 3.21	Gen 3, v. 3.21
<a href="#">Animation Timing API (p. 29)</a>	Not supported	Supported	Supported

**Amazon Silk Developer Guide**  
**Support for HTML5 and Related Features**

<b>Feature</b>	<b>Gen 1, v. 1.0.143.1</b>	<b>Gen 2, v. 3.21</b>	<b>Gen 3, v. 3.21</b>
<a href="#">Application Cache API (p. 30)</a>	Supported	Supported	Supported
<a href="#">Audio Element (p. 41)</a>	Supported	Supported	Supported
<a href="#">Canvas Element (p. 41)</a>	Partially supported	Supported	Supported
<a href="#">contenteditable Attribute (p. 42)</a>	Not supported	Supported	Supported
<a href="#">Cross-Origin Resource Sharing (p. 31)</a>	Supported	Supported	Supported
<a href="#">File API (p. 32)</a>	Not supported	Supported	Supported
<a href="#">File System API (p. 32)</a>	Not supported	Supported	Supported
<a href="#">Geolocation API (p. 34)</a>	Supported	Supported	Supported
<a href="#">Indexed Database API (p. 35)</a>	Not supported	Supported	Supported
<a href="#">Input Types (p. 43)</a>	Not supported	Supported	Supported
<a href="#">Keygen Element (p. 43)</a>	Supported	Supported	Supported
<a href="#">Media Queries (p. 46)</a>	Supported	Supported	Supported
<a href="#">Meter Element (p. 44)</a>	Not supported	Supported	Supported
<a href="#">Output Element (p. 44)</a>	Supported	Supported	Supported
<a href="#">Progress Element (p. 44)</a>	Not supported	Supported	Supported
<a href="#">Server-Sent Events (p. 36)</a>	Not supported	Supported	Supported
<a href="#">SVG Element (p. 45)</a>	Not supported	Supported	Supported
<a href="#">Touch Events (p. 36)</a>	Supported	Supported	Supported
<a href="#">Transform Property (2D) (p. 48)</a>	Supported (-webkit-)	Supported (-webkit-)	Supported (-webkit-)
<a href="#">Transitions (p. 48)</a>	Supported (-webkit-)	Supported	Supported
<a href="#">Video Element (p. 45)</a>	Supported	Supported	Supported
<a href="#">viewport Meta Element (p. 49)</a>	Not supported	Supported	Supported
<a href="#">XMLHttpRequest Level 2 (p. 37)</a>	Not supported	Supported	Supported
<a href="#">Web SQL Database (p. 37)</a>	Supported	Supported	Supported
<a href="#">Web Storage (p. 38)</a>	Supported	Supported	Supported



Feature	Gen 1, v. 1.0.143.1	Gen 2, v. 3.21	Gen 3, v. 3.21
<a href="#">Web Workers API (p. 39)</a>	Not supported	Supported	Supported
<a href="#">WebGL (p. 39)</a>	Not supported	Most functionality supported	Most functionality supported
<a href="#">WebSocket API (p. 40)</a>	Not supported	Supported	Supported

## HTML5 APIs

Amazon Silk supports many of the HTML5 APIs. Though not intended to be comprehensive, the list below describes supported HTML5 APIs and notes any Amazon Silk-specific implementation details.

### Topics

- [Animation Timing API \(p. 29\)](#)
- [Application Cache API \(p. 30\)](#)
- [Cross-Origin Resource Sharing \(p. 31\)](#)
- [File API \(p. 32\)](#)
- [File System API \(p. 32\)](#)
- [Geolocation API \(p. 34\)](#)
- [Indexed Database API \(p. 35\)](#)
- [Server-Sent Events \(p. 36\)](#)
- [Touch Events \(p. 36\)](#)
- [XMLHttpRequest Level 2 \(p. 37\)](#)
- [Web SQL Database \(p. 37\)](#)
- [Web Storage \(p. 38\)](#)
- [Web Workers API \(p. 39\)](#)
- [WebGL \(p. 39\)](#)
- [WebSocket API \(p. 40\)](#)

## Animation Timing API

The Animation Timing API can be used to create script-based animations where the user agent is called upon to determine the appropriate frame update rate at runtime. This allows animations to run more smoothly and efficiently than they would with the `setInterval` or `setTimeout` methods, which schedule callbacks at specified intervals. In the example below, we draw a circle using the `requestAnimationFrame` method of the Animation Timing API.

```
<html>
  <body>
    <canvas width="200" height="200" id="canvas"></canvas>
    <script>
      var endAngle = 0;
      var canvas = document.getElementById('canvas');
      var context = canvas.getContext('2d');

      function animate() {
        context.beginPath();
```

```
context.arc(100, 100, 75, 0, endAngle);
context.stroke();
context.lineWidth = 5;
context.strokeStyle = "green";
if (endAngle <= 2 * Math.PI) {
    endAngle += 0.01;
}
window.requestAnimationFrame(animate);
}
animate()
</script>
</body>
</html>
```

The image below shows the partially drawn circle.



To learn more about the Animation Timing API, see the W3C specification [Timing Control for Script-based Animations](#)

## Application Cache API

The Application Cache API, or AppCache, enables web applications to run offline. AppCache can also improve application performance, as cached resources load faster and reduce server load.

You can use AppCache to make your site or web application available to customers without a network connection. To make a page cacheable, add the `manifest` attribute to the `html` tag.

```
<html manifest="manifest.appcache">
```

The `manifest` attribute points to a manifest file on which you'll list resources to be cached for offline access.

```
CACHE MANIFEST
#manifest.appcache
#Cache the following resources for offline access:

http://example.com/index.html
http://example.com/stylesheet.css
```

The manifest file must have a MIME type of `text/cache-manifest`. Silk caches the resources listed in the manifest file and also any requested page on which the `manifest` attribute is set. These resources

can then be loaded directly from the cache on the device. Of course, a network connection has to be available the first time a customer visits your site.

To learn more about the HTML5 Application Cache, see the following resources:

- [HTML5 Application Cache](#)
- [A Beginner's Guide to Using the Application Cache](#)
- [HTML5 Offline Web Applications](#)

## Cross-Origin Resource Sharing

The Cross-Origin Resource Sharing (CORS) specification defines a method for making HTTP requests that are not limited by the same-origin policy. The same-origin policy restricts scripts from one domain from interacting with resources from a different domain. But when CORS is implemented, a web client can fetch resources from an origin other than its own. In practice, CORS requests are usually made through the XMLHttpRequest API.

Browsers handle the client-side implementation of CORS. This means that you can use XMLHttpRequest to make cross-origin requests, and Amazon Silk will take care of the HTTP request header and any necessary preflight requests (requests for authorization from cross-origin servers).

To implement CORS on the server side, you have to include an `Access-Control-Allow-Origin` header with the response. (For more information on server-side CORS logic, see [Enable CORS](#).)

The following code sample checks for CORS support by testing for the `withCredentials` attribute of the XMLHttpRequest object. The `withCredentials` attribute is a boolean type that enables cookies and credentials in CORS requests. If this attribute is present, then the XMLHttpRequest object supports CORS. (Note, however, that we're not testing for the `XDomainRequest` object, which provided partial CORS support in older versions of Internet Explorer.)

```
<html>
  <body>
    <script>
      var request = new XMLHttpRequest();
      if ("withCredentials" in request) {
        alert("Your browser supports CORS.")
      } else {
        alert("Your browser doesn't seem to support CORS.")
      }
    </script>
  </body>
</html>
```

To learn more about CORS, see the following resources:

- [W3C Recommendation: Cross-Origin Resource Sharing](#)
- [HTML5 Rocks: Using CORS](#)
- [MDN: HTTP access control \(CORS\)](#)
- [Cross-domain Ajax with Cross-Origin Resource Sharing](#)

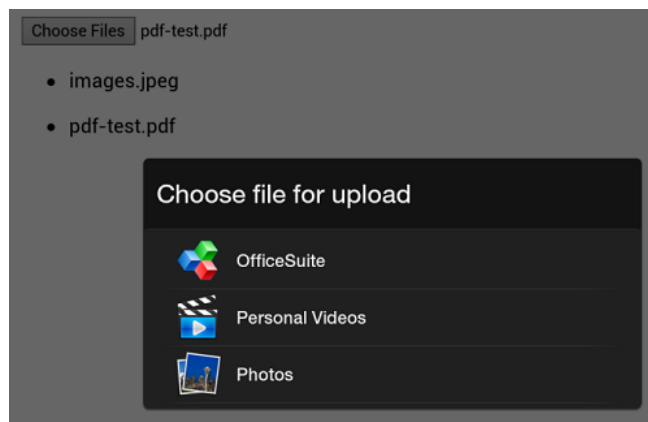
## File API

The File API provides a secure, standardized way for web applications to interact with local files. Using the File API, a web application can represent file objects, programmatically select them, and parse file data.

The example below uses the File API to display the `name` attribute of each file selected by the user.

```
<html>
  <head>
    <script>
      function selectFiles() {
        var filesPicked = document.getElementById('filePicker').files;
        var ul = document.createElement("ul");
        for (var i = 0; i < filesPicked.length; i++) {
          var li = document.createElement("li");
          var text = document.createTextNode(filesPicked[i].name);
          li.appendChild(text);
          ul.appendChild(li);
        }
        document.getElementById("list").appendChild(ul);
      }
    </script>
  </head>
  <body>
    <form>
      <input type="file" id="filePicker" onchange="selectFiles()" multiple>
      <output id="list"></output>
    </form>
  </body>
</html>
```

When the user taps **Choose Files**, the **Choose file for upload** dialog opens.



For more information, see the W3C [File API](#) specification.

## File System API

Using the File System API, a web application can create and interact with files in a sandboxed virtual file system on the client. The File System API gives web applications a way to store files, including large binary blobs, locally without using a database.

The example below creates two file system entries (a file and a directory) and, in response to an onclick event, outputs a list of the entries to the HTML document.

```
<html>
  <head>
    <script>
      var fileSystem;
      navigator.webkitPersistentStorage.requestQuota(1024*1024*5, function(bytesAllowed) {
        window.webkitRequestFileSystem(PERSISTENT, bytesAllowed, function(fs)
        {
          fileSystem = fs;
          createEntries();
        });
      })
      function createEntries() {
        fileSystem.root.getFile("exampleFile.txt", {create: true});
        fileSystem.root.getDirectory("exampleDirectory", {create: true});
      }
      function displayEntries() {
        var reader = fileSystem.root.createReader();
        reader.readEntries(function(entries) {
          var ul = document.createElement("ul");
          for (var i = 0; i < entries.length; i++) {
            var li = document.createElement("li");
            var text = document.createTextNode(entries[i].name);
            li.appendChild(text);
            ul.appendChild(li);
          }
          document.getElementById("result").appendChild(ul);
        });
      }
    </script>
  </head>
  <body>
    <input type="button" value="Show File System" onclick="displayEntries()">
    <output id="result"></output>
  </body>
</html>
```

When Silk loads the page, `webkitPersistentStorage.requestQuota()` requests 5MB of persistent storage and then passes this storage to `webkitRequestFileSystem()` in an anonymous callback. In turn, `webkitRequestFileSystem()` requests a file system of the allowed size and uses an anonymous callback to invoke `createEntries()`, which creates a file and a directory in the requested file system. Tapping **Show File System** calls `displayEntries()`, which outputs a list of the entries to the HTML document.

Show File System

- exampleFile.txt
- exampleDirectory

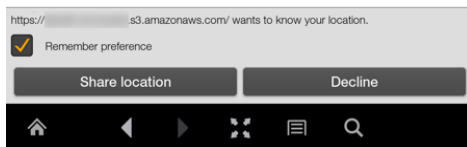
Note the use of the `webkit` prefix in the example. Currently, Silk requires the prefix for `webkitRequestFileSystem()`. Also, note that—unlike Google Chrome—Silk does not open a dialog requesting user approval in response to `webkitPersistentStorage.requestQuota()`.

For more information, see the following resources:

- [File API: Directories and System](#)
- [Exploring the File System APIs](#)

## Geolocation API

The Geolocation API provides an interface to a device's location information, returned as coordinates of latitude and longitude. The first time an app or website tries to access device location with the Geolocation API, the browser has to obtain user permission. All browsers that support the Geolocation API must respect this requirement, although the implementation varies. Amazon Silk prompts the user with a dialog requesting permission.



In the Settings menu, Silk users can disable location access for an individual website or for all websites.

As a developer, you can use the Geolocation API to get an initial position for a device and to watch for changes of position.

For example, the following page includes a script that retrieves the current device position and displays that position using one of the Google Maps APIs.

```
<html>
  <body>
    <div id="map">Tap the button to see your location on a map.</div>
    <div><button onclick="getPosition()">Map location</button></div>
    <script>
      function getPosition() {
        if (navigator.geolocation) {
          navigator.geolocation.getCurrentPosition(mapPosition);
        }
        document.getElementById("map").innerHTML = "Mapping position ..."
      }
      function mapPosition(position) {
        var lat = position.coords.latitude;
        var lng = position.coords.longitude;
        var map_url = "http://maps.googleapis.com/maps/api/staticmap?center="
+ lat + "," + lng + "&zoom=14&size=400x500&sensor=false";
        document.getElementById("map").innerHTML="<img src='"+map_url+"'>";
      }
    </script>
  </body>
</html>
```

Using the `getPosition()` function, we test support for the `navigator.geolocation` property, and then we invoke the asynchronous `getCurrentPosition()` method, to which we pass `mapPosition()` as a callback. The `mapPosition()` function, which is invoked when `getCurrentPosition()` returns successfully, displays a Google Maps map centered at the returned latitude and longitude coordinates.

### Note

The code sample above doesn't alert the user if Geolocation is not supported, and it doesn't handle errors. You would probably want to address those scenarios in production code.

To learn more about the Geolocation API, see the following resources:

- [W3C Geolocation API Specification](#)
- [MDN: Using geolocation](#)
- [Dive Into HTML5: The Geolocation API](#)
- [A Simple Trip Meter using the Geolocation API](#)

## Indexed Database API

The Indexed Database API, or IndexedDB API, is an interface to a high-performance, object-oriented database that can store large amounts of structured data on the browser. Data objects are stored as key-value pairs and can be accessed on- or offline. The following example populates a new client-side database and provides a function, `addBook()`, that puts a new object into the object store.

```
<html>
  <body>
    <input type="button" value="Add Alice in Wonderland" onclick="addBook();"
  />
  <script>
    var db;
    function openDB() {
      var request = indexedDB.open('victorian_novels_DB', 1);
      request.onerror = function(event) {
        console.log('Database error: ' + event.target.errorCode);
      };
      request.onsuccess = function(event) {
        db = request.result;
      };
      request.onupgradeneeded = function(event) {
        var store = event.currentTarget.result.createObjectStore('victori
an_novels', {keyPath: 'isbn'});
        store.createIndex('title', 'title', {unique: true});
        store.createIndex('author', 'author', {unique: false});
        store.put({title: "Bleak House", author: "Charles Dickens", isbn:
1-23456-789-0});
        store.put({title: "Great Expectations", author: "Charles Dickens",
isbn: 2-34567-890-1});
        store.put({title: "Middlemarch", author: "George Eliot", isbn: 3-
45678-901-2});
        store.put({title: "Jude the Obscure", author: "Thomas Hardy", isbn:
4-56789-012-3});
      };
    };
    function addBook() {
      var transaction = db.transaction('victorian_novels', 'readwrite');
      var store = transaction.objectStore('victorian_novels');
      store.put({title: 'Alice in Wonderland', author: 'Lewis Carroll', isbn:
5-67890-123-4});
      transaction.oncomplete = function() {
        alert('Added Alice!');
      };
    };
    openDB();
  </script>
```

```
</body>  
</html>
```

For more information, see the [W3C Indexed DB specification](#).

## Server-Sent Events

The Server-Sent Events interface enables a client to receive updates from the server automatically without having to request them. You can use Server-Sent Events to display news and other updates on a website. The following example shows client-side code for handling a Server-Sent Events stream. The `EventSource` object listens for event data from the `stream` URL, and an event handler writes the data to the page. The server-side code is not shown.

```
<html>  
  <body>  
    <div id="result"></div>  
    <script>  
      var source = new EventSource("/stream");  
      source.onmessage = function(event) {  
        document.getElementById("result").innerHTML += event.data + "<br>";  
      };  
    </script>  
  </body>  
</html>
```

For more information, see the [W3C Server-Sent Events specification](#).

## Touch Events

Touch Events interpret finger motions on a touch-sensitive screen, so that web applications can handle touch input directly. Touch events include `touchstart`, `touchend`, `touchcancel`, and `touchmove`.

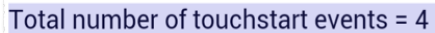
The example below counts the total number of `touchstart` events on the document. A `touchstart` event occurs when a user places a touch point—for example, a finger tap—on the surface of the device.

```
<html>  
  <head>  
    <style>  
      #blueDiv {  
        color: #0F0F3D;  
        background-color: #D6D6F5;  
        font-size: x-large;  
      }  
    </style>  
  </head>  
  <body>  
    <div id="blueDiv"></div>  
    <script>  
      var totalTouches = 0;  
      document.addEventListener('touchstart', function() {  
        totalTouches = totalTouches + 1;  
        var blueDiv = document.getElementById('blueDiv');  
        blueDiv.innerHTML = "Total number of touchstart events = " +  
totalTouches;  
      });  
    </script>  
  </body>  
</html>
```



```
});  
</script>  
</body>  
</html>
```

The following image shows the page after four taps on the device screen:



Total number of touchstart events = 4

To learn more about Touch Events, see the following resources:

- [Touch Events W3C Specification](#)
- [Multi-touch Web Development](#)

## XMLHttpRequest Level 2

The XMLHttpRequest API enables a web application to make asynchronous HTTP requests to the server. XMLHttpRequest Level 2, which is sometimes associated with HTML5, introduces new functionality. For example, with XMLHttpRequest Level 2, you can use the Cross-Origin Resource Sharing (CORS) API to make secure cross-origin requests, and you can transfer binary data in a straightforward way. In the example below, we use an `onclick` event to invoke a function that creates a new XMLHttpRequest for a binary object—in this case, an image.

```
<html>  
<body>  
  <img id="logo" src=""><br>  
  <input type="button" value="Request Image" onclick="display_image()">  
  <script>  
    function display_image() {  
      var request = new XMLHttpRequest();  
      request.open('GET', 'Media/silk_logo.png', true);  
      request.responseType = 'blob';  
      request.onload = function() {  
        if (this.status == 200) {  
          var silkLogo = this.response;  
          var img = document.createElement('img');  
          img.src = window.URL.createObjectURL(silkLogo);  
          document.getElementById("logo").src = img.src;  
        }  
      };  
      request.send();  
    }  
  </script>  
</body>  
</html>
```

For more information, see the W3C specification [XMLHttpRequest Level 2](#).

## Web SQL Database

The Web SQL Database API is an interface for storing data on the client in a database that can be queried with SQLite. The W3C no longer actively maintains the Web SQL Database specification. The example below opens a database, creates a table, and adds data to the table.

```
<html>
  <body>
    <script>
      var webDB = {};
      webDB.db = null;
      webDB.open = function() {
        webDB.db = openDatabase('Books', '1.0', 'Book Inventory', 5 * 1024 *
1024);
      };
      webDB.createTable = function() {
        var db = webDB.db;
        db.transaction(function(trx) {
          trx.executeSql('CREATE TABLE IF NOT EXISTS books(ID INTEGER PRIMARY
KEY ASC, title TEXT, author TEXT)', []);
        });
      };
      webDB.addBook = function(bookTitle, bookAuthor) {
        var db = webDB.db;
        db.transaction(function(trx) {
          trx.executeSql('INSERT INTO books(title, author) VALUES (?,?)',
[bookTitle, bookAuthor]);
          alert('Added ' + bookTitle + ' by ' + bookAuthor);
        });
      };
      webDB.open();
      webDB.createTable();
      webDB.addBook('Wings of the Dove', 'Henry James');
    </script>
  </body>
</html>
```

For more information, see the W3C [Web SQL Database specification](#).

## Web Storage

Web Storage is an interface for storing data in key-value pairs on the client. It's designed to be a faster, more secure alternative to cookies. The Web Storage API provides two storage types: local storage and session storage. Local storage has no expiration date, while session storage persists for one session only. The following example uses the `localStorage.setItem` method to set "companyName" to "Example Corp."

```
<html>
  <body>
    <div id="company"></div>
    <script>
      function testStorage() {
        localStorage.setItem("companyName", "Example Corp.");
        document.getElementById("company").innerHTML = localStorage.companyName;
      }
      testStorage();
    </script>
  </body>
</html>
```

To learn more about the Web Storage API, see the following resources:

- [W3C Web Storage Recommendation](#)
- [HTML5 Web Storage](#)
- [An Overview of the Web Storage API](#)

## Web Workers API

The Web Workers API can improve application performance by enabling JavaScript to run as a background process. When a script runs as a Worker object, it's executed on a background thread, in parallel to the main page. This prevents the script from affecting UI performance.

In the following example, we use a worker object to run `webworker.js` in the background. The `webworker.js` script, which is not shown below, creates a simple JavaScript clock.

```
<html>
<head>
  <script>
    var webworker;
    function startClock() {
      webworker = new Worker("webworker.js");
      webworker.onmessage = function (event) {
        document.getElementById("clock").innerHTML = event.data;
      };
    }
    function stopClock() {
      webworker.terminate();
    }
  </script>
</head>
<body>
  <h1><output id="clock"></output></h1>
  <button onclick="startClock()">Start Clock</button>
  <button onclick="stopClock()">Stop Clock</button>
</body>
</html>
```

For more information about the Web Workers API, see the [W3C Web Workers specification](#).

## WebGL

WebGL is a web standard that facilitates the rendering of interactive 3-D graphics in the browser without a plugin. Based on OpenGL ES 2.0, WebGL specifies both a JavaScript API and interaction with the graphics processing unit (GPU). The HTML5 `canvas` element functions as the rendering context. Amazon Silk has enabled WebGL and supports most WebGL functionality.

The following example tests for WebGL support.

```
<html>
<body>
  <script>
    if (window.WebGLRenderingContext) {
      alert('Your browser supports WebGL.');
```

```
</body>  
</html>
```

For more WebGL initialization tests, see [Khronos WebGL FAQ](#).

To learn more about WebGL, see the following resources:

- [Khronos WebGL Overview](#)
- [Khronos WebGL Specification](#)

## WebSocket API

The WebSocket API facilitates event-driven client-server communication over an open connection. Using the WebSocket API, the server can send updates to the client without the client having to request resources. The example below shows a client-side script that opens a WebSocket connection and sends a message to a test server at [WebSocket.org](#). The server echoes back the message, and that response data is written to the HTML document.

```
<html>  
  <body>  
    <h1>WebSocket API Test</h1>  
    <div id="output"></div>  
    <script>  
      var socket = new WebSocket("wss://echo.websocket.org/");  
      socket.onopen = function() {  
        socket.send("Amazon Silk supports the WebSocket API.");  
      };  
      socket.onmessage = function(e) {  
        var output = document.getElementById("output");  
        output.innerHTML = e.data;  
      };  
    </script>  
  </body>  
</html>
```

To learn more about the WebSocket API, see the following resources:

- [The WebSocket API](#)
- [Introducing WebSockets: Bringing Sockets to the Web](#)
- [WebSocket.org](#)

## HTML5 Elements and Attributes

### Topics

- [Audio Element \(p. 41\)](#)
- [Canvas Element \(p. 41\)](#)
- [contenteditable Attribute \(p. 42\)](#)
- [Input Types \(p. 43\)](#)
- [Keygen Element \(p. 43\)](#)
- [Meter Element \(p. 44\)](#)
- [Output Element \(p. 44\)](#)

- [Progress Element \(p. 44\)](#)
- [SVG Element \(p. 45\)](#)
- [Video Element \(p. 45\)](#)

Amazon Silk supports many of the HTML5 elements and attributes. Though not intended to be comprehensive, the list below describes supported elements and attributes and notes Amazon Silk-specific implementation details, if applicable.

## Audio Element

The `<audio>` element makes it possible to embed audio files directly in a web page without using plug-ins. By nesting `<source>` elements within an `<audio>` element, you can reference multiple file types.

```
<h1>HTML5 Audio Element</h1>
<audio controls>
  <source src="Media/audio_sample.ogg" type="audio/ogg">
  <source src="Media/audio_sample.mp3" type="audio/mpeg">
  Sorry. Your browser doesn't support the HTML5 audio element.
</audio>
```

The sample code above produces the player control shown below:



For more information, see the [W3C audio element wiki](#).

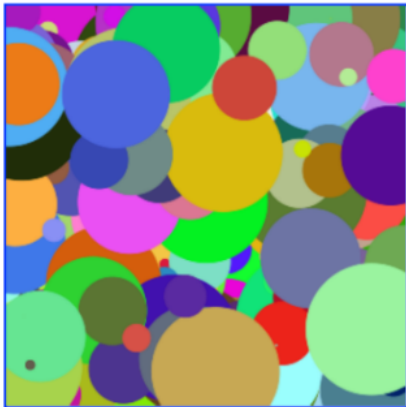
## Canvas Element

You can use the `<canvas>` element to draw two-dimensional graphics on a web page. Use the `<canvas>` element to create a container for a graphic, and then render the graphic itself on the fly using JavaScript. For example, the following code (adapted from the [W3C canvas wiki](#)) draws circles of random size, color, and position.

```
<html>
  <head>
    <style>
      canvas {
        border: 2px solid #0033FF;
        margin-left:10px;
      }
    </style>
    <script>
      function drawCircles() {
        var sampleCanvas = document.getElementById("bubbleCanvas");
        var context = sampleCanvas.getContext("2d");
        var width = sampleCanvas.width, height = sampleCanvas.height;
        var i = 0;
        do {
          context.fillStyle = "rgb(" + Math.round(255 * Math.random()) + ","
```

```
        + Math.round(255 * Math.random()) + ","  
        + Math.round(255 * Math.random()) + "));  
  
        context.beginPath();  
        context.arc(width * Math.random(), height * Math.random(), 50 *  
Math.random(), 0, Math.PI * 2, true);  
        context.closePath();  
        context.fill();  
    } while (++i != 1000);  
}  
</script>  
</head>  
<body onLoad="drawCircles();">  
    <canvas id="bubbleCanvas" width="300" height="300"></canvas>  
</body>  
</html>
```

The resulting canvas element is shown below:



For more information, see the [HTML Canvas 2D Context specification](#).

## contenteditable Attribute

Setting the `contenteditable` attribute to "true" makes a text element on your web page available for the user to edit. You can also use the `localStorage` object to save the user's changes, effectively turning your web page into a text editor. The following example applies the `contenteditable` attribute to a paragraph.

```
<p contenteditable="true">Silk supports the HTML5 contenteditable attribute.</p>
```

As a result, we can tap the paragraph element and enter text ("HOORAY!") using the virtual keyboard.

Silk supports the HTML5 contenteditable attribute. HOORAY!

For more information, see [Mozilla Developer Network: Content Editable](#).

## Input Types

In HTML5, you can use input types to make forms more responsive to mobile clients. By specifying in your markup the type of input required, you can trigger the mobile device to display an appropriate virtual keyboard. Amazon Silk supports this functionality. For example, if you set the `type` attribute to "email", Amazon Silk displays a virtual keyboard featuring the `@` and `.com` keys, which make the keyboard more user-friendly for typing an email address.

```
<form>
  Email: <input type="email" name="email"><br>
  <input type="submit" value="Sign Up">
</form>
```

The Silk virtual keyboard is shown below:



For more information, see [Making Forms Fabulous with HTML5](#).

## Keygen Element

The `<keygen>` element specifies a form field for securely authenticating users. When the form containing the `<keygen>` element is submitted, a public/private key pair is generated. The key pair can be used as part of a certificate authentication system. For example, the following form contains a `<keygen>` element with a `keytype` attribute that specifies RSA encryption.

```
<form action="signup.php" method="post">
  Email: <input type="email" name="user_email"><br>
  Password: <input type="password" name="user_password"><br>
  <keygen name="key" keytype="rsa">
  <input type="submit" value="Submit">
</form>
```

The sample code above produces the form shown below:

Email:

Password:

2048 (High Grade)


For more information about the `<keygen>` element, see [HTML/Elements/keygen](#) and [Mozilla Developer Network: <keygen>](#).

## Meter Element

You can use the `<meter>` element to measure data within a given range. It specifies a fractional value or gauge. For example, you could use the `<meter>` element to gauge hard disk usage.

```
<p>Hard Disk Usage: <meter min="0" value="239" max="296">239 GB used out of 296 GB total</meter></p>
```

The sample code above produces the gauge below:

Hard Disk Usage: 

For more information, see [HTML/Elements/meter](#).

## Output Element

The `<output>` element represents the result of a calculation. Although the `<output>` element is associated with a form, it doesn't have to be a child of the form. You can collect input for a calculation in a form, and then use the `<output>` element to display the results of the calculation elsewhere in the document. In the following example, the `<output>` element displays the sum of the values entered into the two input fields:

```
<form oninput="pets.value=parseInt(dogs.value)+parseInt(cats.value)">  
  No. of dogs  
  <input type="number" id="dogs" value="0"><br>  
  No. of cats  
  <input type="number" id="cats" value="0"><br>  
  Total no. of pets:  
  <output name="pets" for="dogs cats"></output>  
</form>
```

Here's how the markup looks in Silk, after a user enters values in the input fields:

No. of dogs   
No. of cats   
Total no. of pets: 5

Note that the `oninput` event is not supported by Silk Gen 1. For more on the `<output>` element, see [HTML/Elements/output](#).

## Progress Element

The `<progress>` element represents progress toward completion of some task, like a file download. You can use the `<progress>` tag together with JavaScript to create a progress display.



```
<div>File downloading: <progress value="70" max="100">70%</progress></p></div>
<div>Silk supports the HTML5 progress element.</div>
```

The sample code above produces the progress display shown below:

File downloading: 

Silk supports the HTML5 progress element.

For more information, see [HTML/Elements/progress](#).

## SVG Element

Scalable Vector Graphics (SVG) is an XML-based format for describing two-dimensional web graphics. Amazon Silk supports SVG both inline via the `<svg>` tag and externally using the `<img>`, `<object>`, and `<embed>` tags. The example below creates an SVG rendering of a rectangle with a horizontal linear gradient. The image shows the result rendered by Silk.

```
<html>
  <body>
    <svg width="180" height="120">
      <defs>
        <linearGradient id="gradient" x1="0%" y1="0%" x2="100%" y2="0%">
          <stop offset="0%" style="stop-color:rgb(255,255,255);stop-opacity:1.0"
        />
          <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:0.75" />
        </linearGradient>
      </defs>
      <rect x="10" y="10" width="160" height="100" style="fill:url(#gradient)">
    </svg>
  </body>
</html>
```



To learn more about SVG, visit the [W3C SVG Working Group](#).

## Video Element

The `<video>` element makes it possible to embed video files directly in a web page without using plug-ins. By nesting `<source>` tags within a `<video>` element, you can reference multiple file types.

```
<h1>HTML5 Video Element</h1>
<video width="320" height="240" controls>
  <source src="Media/video_sample.mp4" type="video/mp4">
  <source src="Media/video_sample.ogv" type="video/ogg">
```

```
<source src="Media/video_sample.webm" type="video/webm">
Sorry. Your browser doesn't support the HTML5 video element.
</video>
```

The sample code above produces the player control shown below:

### HTML5 Video Element



For more information, see the [W3C video element wiki](#).

## CSS3 Support

### Topics

- [Media Queries \(p. 46\)](#)
- [Transform Property \(2D\) \(p. 48\)](#)
- [Transitions \(p. 48\)](#)
- [viewport Meta Element \(p. 49\)](#)

Amazon Silk supports basic CSS3 features like backgrounds, opacity, rounded corners, and text effects, as well as other features. Though not intended to be comprehensive, this page describes supported features and notes Amazon Silk-specific implementation details, where applicable.

## Media Queries

Media queries provide a way to apply style rules based on particular media features like width, height, and resolution. Using media queries, you can deliver layouts that, in effect, respond to the form factor of the user agent.

Media queries are a key component of [Responsive Web Design \(p. 6\)](#). You can use media queries to create a fluid grid that responds to changes in viewport size. (For more on the viewport, see [viewport Meta Element \(p. 49\)](#)). You can also tailor the user experience to device orientation, so that your layout changes as the user goes from portrait to landscape mode. For example, the following media query tests for a device in portrait mode.

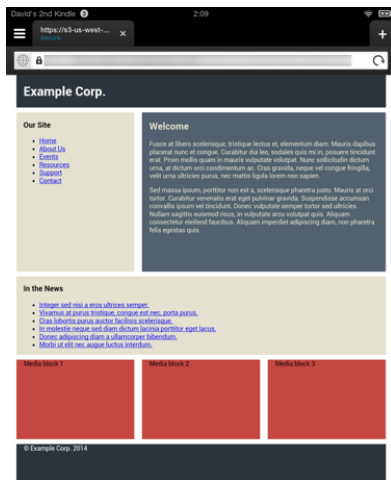
```
@media screen and (orientation: portrait) { ... }
```

And this next media query tests for a device in landscape mode.

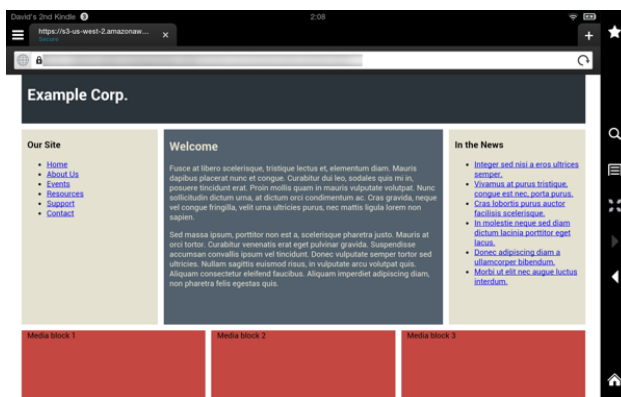
```
@media screen and (orientation: landscape) { ... }
```

In each case, you'd put style rules between the braces, making it possible to deliver a different layout for each mode.

Many developers use media queries to test for viewport width. By testing for viewport width, you can target the actual screen area available to the browser. The following image shows a responsive layout (adapted from [Gridpak](#)) viewed in portrait mode on a Kindle Fire HD 8.9".



And the next image shows the site viewed in landscape mode on the same device.



Notice the difference in the containers. In the first layout (portrait mode), there are two containers directly under the header ("Example Corp."). In the second (landscape mode), there are three. Media queries enable this dynamic change in layout. In portrait mode, the following media query is in effect:

```
@media screen and (min-width: 320px) and (max-width: 999px) { ... }
```

The device has a viewport width less than 999 pixels in portrait mode, so the query tests true. In the landscape example, the following media query governs the style rules applied.

```
@media screen and (min-width: 1000px) { ... }
```

This query tests true because, in landscape mode, the device has a viewport width greater than 1000 pixels. Instead of using `width` to detect the viewport size, you could also use `device-width` to detect

the actual screen size of the device. Unlike `width`, `device-width` is not dependent on device orientation. You can test for other media features, too. For a list, see [Media features](#).

To learn more about media queries, see the following resources:

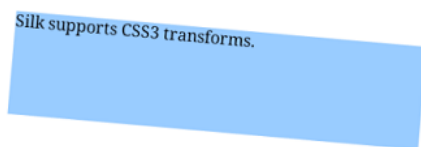
- [W3C Media Queries specification](#)
- [Media Queries](#)

## Transform Property (2D)

The CSS3 `transform` property can be used to rotate, scale, skew, and otherwise alter an element. You can use the `transform` property to make a web page more interactive.

```
<html>
  <head>
    <style>
      div {
        width:400px;
        height:100px;
        background-color:#99CCFF;
        -webkit-transform:rotate(5deg);
      }
    </style>
  </head>
  <body>
    <br>
    <div>Silk supports CSS3 transforms.</div>
  </body>
</html>
```

The sample code above produces the rotated element shown below:



For more information, see [CSS3 transform Property](#).

## Transitions

Using a CSS3 `transition`, you can add an effect to a property change, without employing JavaScript or Flash. The result is an element that changes from one style to another, like an animation. In the example below, the `<input>` element widens by 100 pixels and changes color when it comes into focus.

```
<html>
  <head>
    <style>
      input {
        width: 100px;
        margin: 5px 0;
        padding: 5px;
        background-color: #CCCCCC;
      }
    </style>
  </head>
  <body>
    <input type="text" value="Silk supports CSS3 transitions." />
  </body>
</html>
```

```
        border-radius: 5px;
        transition: 2s;
    }
    input:focus {
        width:200px;
        background-color: #FFFFFF;
    }
</style>
</head>
<body>
    <form action="">
        <input type="text" name="FirstName" value="First Name">
    </form>
</body>
</html>
```

Here's how it looks in Amazon Silk:



For more information, see the W3C specification [CSS Transitions](#).

## viewport Meta Element

The viewport represents the display area available to the browser. The viewport can be bigger or smaller than the physical screen of the device, and it doesn't include browser chrome (the browser borders and controls). For some calculations, it may be useful to differentiate between a visual viewport (the dimensions of the area visible on the screen) and a layout viewport (the dimensions of the entire display area). For more on the visual and layout viewports, see Peter-Paul Koch's [A tale of two viewports — part two](#).

Using the `viewport` meta element, which implements the same functionality as the `@viewport` CSS rule, you can alter the viewport for your site and improve the mobile experience for your users. The `viewport` meta tag lets you specify the width, height, and scale of the viewport. In effect, you can use the tag to give Silk and other browsers instructions on how to render your site.

As a general rule, if you're creating a responsive web design, you'll want to set the viewport element. There's no single correct configuration for the viewport, so choose settings that work well with your site and with the user agents you want to support. There are a number of [viewport properties](#) that you can set and combine to achieve your desired display.

Let's look at an example of the `viewport` meta element. The following markup sets the viewport width equal to the width of the device, so that the site content will have an initial scale appropriate to the device's screen size.

```
<head>
  <meta name="viewport" content="width=device-width">
</head>
```

If we wanted to achieve the same effect using the `@viewport` rule in CSS, we could do the following:

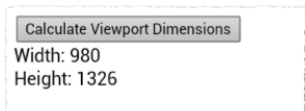
```
@viewport {  
  width: device-width;  
}
```

### Detecting the Silk Viewport

You can obtain the dimensions of the Silk viewport by accessing the `innerWidth` and `innerHeight` properties of the `window` object. The following example outputs the values of `window.innerWidth` and `window.innerHeight` to the HTML document.

```
<html>  
  <head>  
    <script>  
      function getViewPortDimensions() {  
        var viewportWidth = window.innerWidth;  
        var viewportHeight = window.innerHeight;  
        document.getElementById("output").innerHTML = "Width: " + viewportWidth  
+ "<br>Height: " + viewportHeight;  
      }  
    </script>  
  </head>  
  <body>  
    <div><button onclick="getViewPortDimensions()">Calculate Viewport Dimen  
sions</button></div>  
    <div id="output"></div>  
  </body>  
</html>
```

Here's the result after tapping the button on a Kindle Fire HD 8.9" in portrait mode.



If we ran the script in landscape mode or with a zoomed-in view, we'd get a different output, as the visual viewport changes with device orientation and zoom factor.

For more information on the `viewport` meta element, see the following resources:

- [Responsive Web Design \(p. 6\)](#)
- [W3C: Viewport META Element](#)

# Tutorials

---

The tutorials below will help you optimize your site or application for Amazon Silk.

## Topics

- [Detect the Silk User Agent \(p. 51\)](#)
- [Create Drop-down Menus for a Touch Screen \(p. 53\)](#)

## Detect the Silk User Agent

As a method to determine browser support for a given feature, [Feature Detection \(p. 14\)](#) is almost always preferable to user agent detection. Nevertheless, in a few scenarios (for example, performing site analytics) you may need to detect a particular user agent.

If you want to detect version and configuration info in the Amazon Silk user agent, you can use a script like the one embedded in the following HTML document:

```
<html>
  <body>
    <script>
      var match = /(?:; ([^;])+) Build\/(.*)?\bSilk\/([0-9._-]+)\b(.*\bMobile Sa
fari\b)?/.exec(navigator.userAgent);
      if (match) {
        alert("Detected Silk version "+match[2]+" on device "+(match[1] ||
"Kindle Fire")+ " in mode "+(match[3] ? "Mobile" : "Default (desktop)");
      }
    </script>
  </body>
</html>
```

Here's the page displayed in Silk on a Kindle Fire HDX 7".



The script has detected the browser version, product model, and requested mode. We could search for other fields, too. To learn more about fields in the Silk user agent, see [Amazon Silk User Agent Strings](#) (p. 12).

### How the Regex Works

You can make use of the above script without delving into the details of the regular expression. But if you're interested in the details, read on.

Our regular expression, which is an object in JavaScript, is assigned to a variable, `match`:

```
var match = /(?:; ([^;])+) Build\/.*?\bSilk\/([0-9._-]+)\b(.*\bMobile Safari\b)?/.exec(navigator.userAgent);
```

The first and last `/` of the pattern indicate the beginning and ending of the regular expression literal.

```
/(?:; ([^;])+) Build\/.*?\bSilk\/([0-9._-]+)\b(.*\bMobile Safari\b)?/
```

We can think of our regex as comprising three subpatterns. In the first, we use noncapturing parentheses and a trailing `?` to specify an optional match:

```
(?:; ([^;])+) Build\/.*?
```

This pattern will match the build ID (for example, `KFTHWI Build`), if it's available. The inner parentheses capture one or more characters that are not `;` and `)`. Thus, these parentheses will capture `KFTHWI` or another build ID. The entire pattern group needs to be optional because 1st Generation Kindle Fire devices don't have a build ID.

Following this first subexpression is a pattern that matches the Silk browser version:

```
\bSilk\/([0-9._-]+)
```

The character set to be captured includes digits, dot, underscore, and dash, with which we can capture either the browser version (for example, `3.7`) or the 1st Generation Kindle version (for example, `1.0.13.81_10003810`).

The final pattern is comparatively straightforward:

```
\b(.*\bMobile Safari\b)?
```

We match the string `Mobile Safari`, preceded by any character 0 or more times. If Silk requests a mobile view, the user agent will include the string `Mobile Safari`. So this final pattern does exactly what you'd expect: It identifies the browser as a mobile client.

To access the captured substrings, we can use the array object returned by the `exec()` method. In our example, we call `.exec(navigator.userAgent)` on the entire regular expression literal. The `exec()`



method tries to match the regex object against a string passed in as a parameter. In this case, the string to be matched is the user agent object (`navigator.userAgent`). The `exec()` method returns the captured matches as elements of an array. In our example script, we use the returned array elements to build a string for an alert message.

```
if (match) {
    alert("Detected Silk version "+match[2]+" on device "+(match[1] || "Kindle
    Fire")+ " in mode "+(match[3] ? "Mobile" : "Default (desktop)"));
}
```

We use logical OR `||` and the conditional operator `?` to provide appropriate default values in case of nonmatches. Of course, if you're doing user agent detection in a production website, you probably want to do something other than call the `alert()` method.

To learn more about the Silk user agent, see [Amazon Silk User Agent Strings \(p. 12\)](#). To learn more about regular expressions in JavaScript, see [Regular Expressions](#) and [RegExp](#) at the [Mozilla Developer Network](#).

## Create Drop-down Menus for a Touch Screen

Because Amazon Silk runs on a touch-screen device, it doesn't handle the CSS pseudoclass `:hover` the same way that a desktop browser does. On a desktop browser, `:hover` becomes a match when you move the pointer over an element on which `:hover` is set. This behavior is useful for drop-down menus, as you can create a menu that's hidden until the user hovers over the parent element. But on a touch screen, this sort of hover-based menu design can lead to problems.

Let's look at an example. The following HTML document contains two unordered lists, one nested within the other. Each `<li>` element contains a link.

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="dropdown.css">
  </head>
  <body>
    <div class="nav">
      <ul>
        <li><a href="http://example.com/">Home</a></li>
        <li><a href="http://example.com/">About</a></li>
        <li class="more"><a href="http://example.com/">Nav</a>
          <ul>
            <li><a href="http://example.com/">Item 1</a></li>
            <li><a href="http://example.com/">Item 2</a></li>
            <li><a href="http://example.com/">Item 3</a></li>
          </ul>
        </li>
        <li><a href="http://example.com/">Contact</a></li>
        <li><a href="http://example.com/">Press</a></li>
      </ul>
    </div>
  </body>
</html>
```

By applying CSS to this markup, we can create a simple drop-down navigation. Here's our style sheet:

```
div.nav ul {
  padding: 0;
  margin: 0;
  list-style-type: none;
}

div.nav ul li {
  color: #FFF;
  padding: 15px;
  font-size: 20px;
  border-right: 2px #FFF solid;
  border-bottom: 1px #FFF solid;
  float:left;
  background-color:#335A7F;
  width: 110px;
}

div.nav ul li a {
  color: #FFF;
  text-decoration: none;
}

div.nav li:hover {
  background-color: #4C88BF;
}

div.nav ul li ul {
  display:none;
}

div.nav ul li:hover ul {
  display: list-item;
  position: absolute;
  margin-top: 14px;
  margin-left: -15px;
}

div.nav ul li:hover ul li {
  float:none;
}

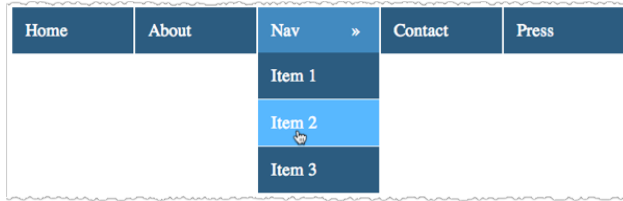
div.nav ul li ul li:hover {
  float:none;
  background-color: #66B5FF;
}

li.more:after {
  content: "\00BB";
  float: right;
  margin-right: 7px;
}
```

Notice how the `display` property is used. First, we use `display:none` to hide the nested `<ul>`, and then we use the `:hover` state to trigger `display:list-item`, which overrides the first `display` and shows us the nested `<ul>`. The result, rendered on a desktop browser, is shown below.

## Amazon Silk Developer Guide

### Create Drop-down Menus for a Touch Screen



A single `<li>` contains both a hidden list and a link. To display the drop-down menu, you hover over the appropriate `<li>`. To follow a link, you click the `<a>` element within the appropriate `<li>`. In other words, you need to register two different events under the same parent element. This works fine as long as you're using a mouse, which supports both hovering and clicking. But Silk relies on a single gesture—a tap—to represent both hovering and clicking. As a result, a user might tap an element with the intention of showing menu items, and the effect would be to follow the link. That's a potentially frustrating user experience.

There are several ways to avoid this problem. One possibility, given the prevalence of touch-screen devices, is simply not to use menus that are dependent on a hover state. Another option is to detect touch-screen devices and then deliver a different, touch-optimized menu. Similarly, you can use scripting to alter the way that the menu responds to touch events. [Superfish](#), a [jQuery](#) plugin, provides such a solution.

In the following example page, which is distributed by Superfish and is shown here rendered by Silk on a Kindle Fire HDX, the drop-down menus open on tap.



To follow a top-level menu item (for example, menu item 3), you'd tap it a second time. On a desktop browser, the drop-down menus unfold on hover, and you follow links by clicking. Thus, the menus are navigable on both touch-screen and desktop browsers.

Superfish is just one option among many, and it may not be the best solution for your site. The point is that it's important to create drop-down navigations that provide a good experience for both desktop and touch-screen visitors. To do so, you'll need to ensure that the `:hover` pseudoclass is not hiding content from touch-screen users.

For more on drop-down menus and touch screens, see the following resources.

#### Additional Resources

- [Touch and Mouse: Together Again for the First Time](#)
- [Mozilla Developer Network :hover](#)

# Troubleshooting

---

The following documentation can help you troubleshoot problems that you might have with Amazon Silk.

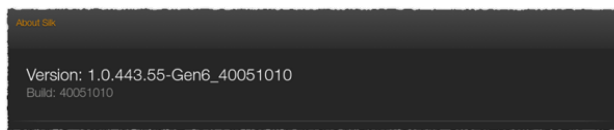
## Topics

- [How do I determine the Amazon Silkbuild version? \(p. 56\)](#)
- [Does Amazon Silk support inline playback of HTML5 video? \(p. 57\)](#)
- [Why is the mobile \(or desktop\) version of my site being delivered? \(p. 57\)](#)
- [How do I determine the client's IP address? \(p. 57\)](#)
- [Why won't my Flash content play? \(p. 58\)](#)
- [Cookie Issues with ASP.NET 4.0 \(p. 58\)](#)

## How do I determine the Amazon Silkbuild version?

Each version of Amazon Silk includes a build version and a browser version. In some troubleshooting scenarios, you may need to know both the browser and build versions for a given Amazon Silk client. You can obtain the browser version from the Amazon Silk user agent string.

You can obtain the build version from the Kindle Fire device. To do so, enter `about:version` in the address bar. Amazon Silk will display the build version string.



The following table maps the current Amazon Silk browser version to build versions. For each browser version there can be several build versions, each of which corresponds to a generation of Kindle Fire devices. To learn more about Kindle Fire device types, see the [Kindle Support device list](#).

**Amazon Silk Developer Guide**  
**Does Amazon Silk support inline playback of HTML5 video?**

---

**Note**

The version of Amazon Silk running on 1st Gen devices is identified only by build version, and not by a browser version.

Amazon Silk Browser Version	Amazon Silk Build Version		
	1st Gen—Kindle Fire	2nd Gen—Kindle Fire HD 8.9"; Kindle Fire HD 7"; Kindle Fire (2nd Gen)	3rd Gen—Kindle Fire HDX 8.9"; Kindle Fire HDX 7"; Kindle Fire HD 7" (3rd Gen)
n/a	1.0.143.1	n/a	n/a
3.21	n/a	1.0.263.20	1.0.614.20

**Note**

Last updated: August 7, 2014

## Does Amazon Silk support inline playback of HTML5 video?

Older versions of Amazon Silk (version 3.4 and earlier) always played HTML5 video in full-screen mode. Newer versions of the browser support playing inline video. In a few cases, sites with video players have detected Amazon Silk and assumed that it doesn't support inline video. In such cases, site logic should be updated to consider the version of Amazon Silk being used. For more on user agent detection, see [Amazon Silk User Agent Strings \(p. 12\)](#). For more on feature detection, see [Feature Detection \(p. 14\)](#).

## Why is the mobile (or desktop) version of my site being delivered?

Users can choose to request a mobile view, a desktop view, or a view that Amazon Silk determines automatically. If the user chooses the automatic option, Amazon Silk requests the desktop view except in cases where a mobile view provides a better user experience. For more information about the requested view, see [Amazon Silk User Agent Strings \(p. 12\)](#).

## How do I determine the client's IP address?

By default, and when it's efficient, Amazon Silk routes requests through a remote proxy server in the Amazon cloud. Thus, the source IP for a request may be that of the remote proxy, and not of the originating client.

### Obtaining the Client IP Address

When Amazon Silk does not route requests through a remote proxy server, the source IP address of the request can be obtained as it normally would be on any HTTP request.

When browsing is routed through a remote proxy, the source IP address of the end client is supplied in the X-Forwarded-For request header. Note that different requests from a single end user may be routed through different cloud servers. In other words, a website may receive a series of requests from different source IP addresses but with the same X-Forwarded-For header.

Additionally, a single Amazon Silk cloud server can support multiple end users. This means that a website may see requests with the same source IP address but different X-Forwarded-For headers.

## Why won't my Flash content play?

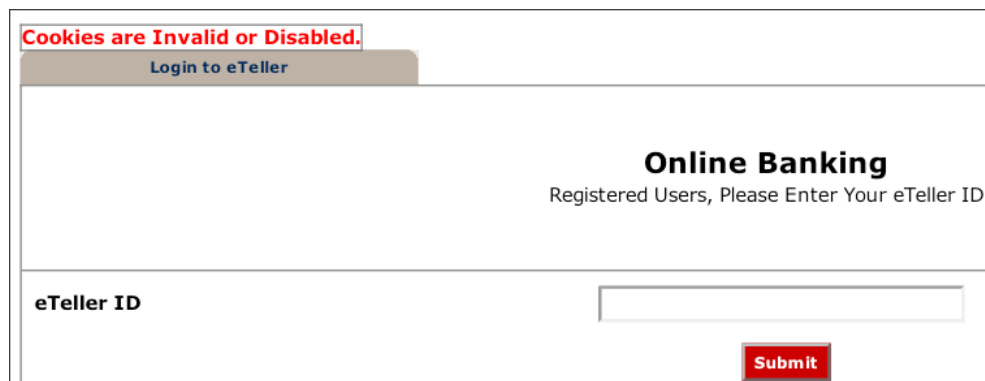
Flash is not supported by Amazon Silk. Embedding video content using the HTML5 video element will yield the best results.

## Cookie Issues with ASP.NET 4.0

### Important

This section describes the solution to a serious, customer-impacting issue. If you own a site that runs on ASP.NET 4.0, we recommend that you read this page, contact Microsoft support to obtain the fix described below, and install it immediately.

Some Amazon Silk users have reported problems with websites running on ASP.NET 4.0. While browsing in Amazon Silk with cookies enabled, users may see spurious messages indicating that cookies aren't supported.



The problem stems from the way that ASP.NET 4 interprets the Amazon Silk user agent string. In order to serve browser-appropriate content, ASP.NET maps browsers to their capabilities. A browser's capabilities might include support for cookies, JavaScript, and certain HTML elements. To identify a browser, ASP.NET uses regular expressions to match patterns in the user agent string.

Because the Amazon Silk 2.0 user agent string doesn't match the default regular expressions used by ASP.NET 4, Amazon Silk isn't mapped to support cookies. Amazon Silk users may be affected when they visit sites running on ASP.NET 4 with default settings. Users may also encounter this problem on ASP.NET 2.0 or 3.5 if a [browser definition file hotfix](#) has been installed.

## I own an ASP.NET 4–based site. How can I tell if my customers are encountering the cookie problem?

If you have access to a Kindle Fire HD running Amazon Silk 2.0, you can test cookie support manually by navigating to pages on your site and checking for warning messages. Alternatively, you can test your site by spoofing the Amazon Silk user agent string.

## I own an ASP.NET 4–based site. How do I fix the cookie problem?

Microsoft has created a hotfix that corrects the cookie issue. To obtain the hotfix, visit the Microsoft support page for [Hotfix rollup 2828843](#).

## I’m a Amazon Silk user encountering the cookie problem. What should I do?

First, go to the **Amazon Silk Settings** menu and make sure that cookies are enabled: Tap the menu button and select **Settings**. You should see a checkmark by **Accept cookies**.



If cookies are enabled for Amazon Silk, and you still encounter a cookie alert when you visit a site, then the site probably needs to add a custom browser definition file. We recommend that you report the issue to the site owner and include a link to this page.

## Why is Amazon Silk 2.0 affected, but not Amazon Silk 1.0?

The ASP.NET 4 cookie issue affects Amazon Silk 2.0 but not Amazon Silk 1.0 due to a difference in user agent strings. Here’s the user agent string for Amazon Silk 1.0:

```
Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_3; en-us;
    Amazon Silk/1.0.13.81_10003810) AppleWebKit/533.16 (KHTML,
    like Gecko) Version/5.0
    Safari/533.16 Amazon Silk-Accelerated=true
```

And here’s the user agent string for Amazon Silk 2.0:

```
Mozilla/5.0 (Linux; U; en-us; KFTT Build/IML74K) AppleWebKit/535.19 (KHTML,
    like Gecko) Amazon Silk/2.1 Safari/535.19 Amazon Silk-Accel
    erated=true
```

While the Amazon Silk 1.0 user agent contains the string `Version`, the Amazon Silk 2.0 user agent doesn’t. When ASP.NET 4 identifies Amazon Silk’s capabilities, it uses the `Version` string to map Amazon

**Amazon Silk Developer Guide**  
**Why is Amazon Silk 2.0 affected, but not Amazon Silk**  
**1.0?**

---

Silk to cookie support. Because Amazon Silk 2.0 doesn't contain this string, ASP.NET 4 doesn't recognize the browser's cookie support.



# Resources

---

Web development is a fast-moving field. The following resources provide perspectives on what's new, what's standard, and what's changing. Here you'll find suggestions and tools for developing great web sites and applications for Amazon Silk or any other browser.

## Sites

### Web Performance Optimization

- [Google Developers: Web Performance Best Practices](#)
- [HTTP Archive](#)
- [Steve Souders: High Performance Web Sites](#)
- [Yahoo! Developer Network: Exceptional Performance](#)

### Development and Design

- [A List Apart](#)
- [Growing with the Web](#)
- [HTML5 Rocks](#)
- [LukeW: Writings on Digital Product Strategy and Design](#)
- [Mozilla Developer Network](#)
- [Mozilla Hacks](#)
- [NCZOnline](#)

### Web Standards and Tests

- [Can I use...](#)
- [Mobile HTML5](#)
- [WHATWG](#)
- [World Wide Web Consortium \(W3C\)](#)

### Hosting and Deploying with Amazon Web Services

- [Deployment and Management on AWS](#)
- [Getting Started with AWS](#)
- [Getting Started with AWS: Deploying a Web Application](#)
- [Host a Static Website on Amazon Web Services](#)

## Video Talks and Tutorials

- [Amazon Silk: Amazon's Revolutionary Cloud-Accelerated Web Browser](#)
- [Paul Irish: Delivering the Goods](#)
- [Steve Souders: High Performance Mobile](#)

## Books

- [Crockford, Douglas. JavaScript: The Good Parts](#)
- [Grigorik, Ilya. High Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance](#)
- [Souders, Steve. High Performance Web Sites: Essential Knowledge for Front-End Engineers](#)

# Document History

The following table describes important changes to the *Amazon Silk Developer Guide*. We also make regular revisions in response to customer feedback.

- **Latest documentation update:** August 11, 2014

Change	Description	Change Date
Added pages on remote debugging and Do Not Track support.	This version of the <i>Amazon Silk Developer Guide</i> includes pages on the Silk remote debugger and on the Do Not Track header field: <ul style="list-style-type: none"> <li>• <a href="#">Do Not Track</a> (p. 25)</li> <li>• <a href="#">Remote Debugging</a> (p. 24)</li> </ul>	August 11, 2014
Added a resources page and a user agent tutorial.	This version of the <i>Amazon Silk Developer Guide</i> includes a page of additional resources and a user agent detection tutorial: <ul style="list-style-type: none"> <li>• <a href="#">Resources</a> (p. 61)</li> <li>• <a href="#">Detect the Silk User Agent</a> (p. 51)</li> </ul>	May 9, 2014
HTML5 support table	This version of the <i>Amazon Silk Developer Guide</i> includes a table showing HTML5 feature support across Silk generations: <a href="#">Support for HTML5 and Related Features</a> (p. 27).	March 28, 2014
New feature detection page and WebGL section	This version of the <i>Amazon Silk Developer Guide</i> includes a page on feature detection and a section on WebGL: <ul style="list-style-type: none"> <li>• <a href="#">Feature Detection</a> (p. 14)</li> <li>• <a href="#">WebGL</a> (p. 39)</li> </ul>	January 3, 2014

Change	Description	Change Date
HTML5 element examples	<p>This version of the <i>Amazon Silk Developer Guide</i> contains example code and images for the following HTML5 elements:</p> <ul style="list-style-type: none"> <li>• <a href="#">Audio Element (p. 41)</a></li> <li>• <a href="#">Video Element (p. 45)</a></li> <li>• <a href="#">Canvas Element (p. 41)</a></li> <li>• <a href="#">contenteditable Attribute (p. 42)</a></li> <li>• <a href="#">Keygen Element (p. 43)</a></li> <li>• <a href="#">Input Types (p. 43)</a></li> <li>• <a href="#">Output Element (p. 44)</a></li> <li>• <a href="#">Progress Element (p. 44)</a></li> <li>• <a href="#">Meter Element (p. 44)</a></li> </ul>	July 12, 2013
New responsive web design page	<p>This version of the <i>Amazon Silk Developer Guide</i> includes an overview of best practices in responsive web design:</p> <ul style="list-style-type: none"> <li>• <a href="#">Responsive Web Design (p. 6)</a></li> </ul>	June 5, 2013
Initial release	This is the first release of the <i>Amazon Silk Developer Guide</i> .	May 9, 2013