User Guide

Amazon Silk

Amazon Silk: User Guide

Table of Contents

What is Amazon Silk?

Amazon Silk is a web browser available for customers to use on Fire Tablets, Fire TVs, and Echo Show devices. Silk is built on the Chromium Project, and is consistently updated to create a faster, secure and more responsive web browsing experience.

Amazon Silk works like any other web browser and allows you to easily add bookmarks, search the web, watch videos, browse your favorite websites and much more. Silk is the preferred web browser for Amazon devices such as the Fire Tablet, Fire TV and Echo Show.

If you're an Amazon Silk user, and you want a more general introduction to Silk, see the Silk User Guide in the browser. Use the guide to learn about customizing the Silk view, using bookmarks, using browser tabs, clearing history, downloading content, and other topics. Off-device user documentation is also available on the Fire, Kindle & Echo Support site.

Silk on Fire Tablet

You can use Amazon Silk to browse the web on your Fire Tablet or app-based Kindle devices. For example, use Silk to

- Search the web for new reading material.
- Keep up with the latest trends and news.
- Shop on the web easily using your device.

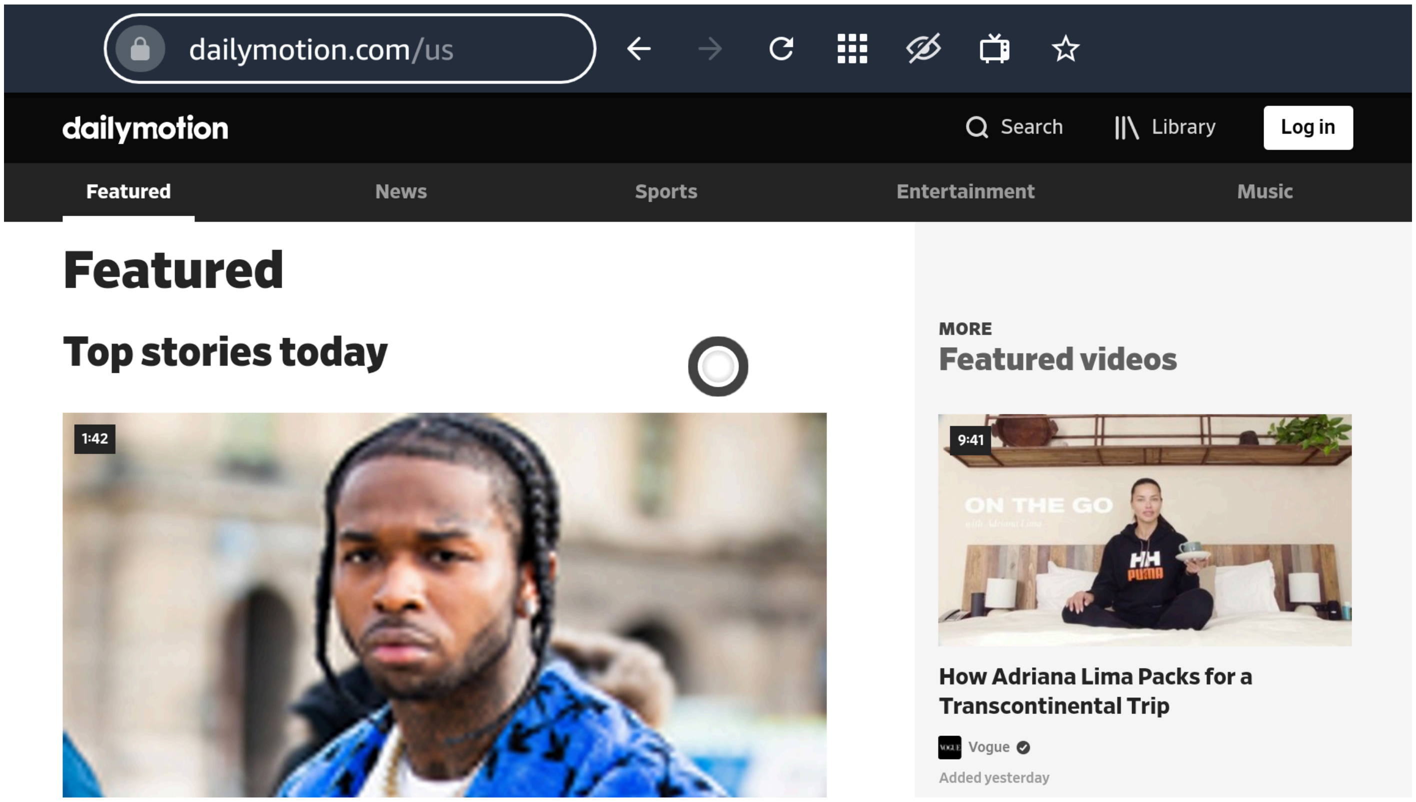
Amazon Silk is pre-installed on your Fire Tablet. Open the App and start browsing. To learn more about Silk on Fire tablet, visit Fire Tablet Help.

Silk on Fire TV

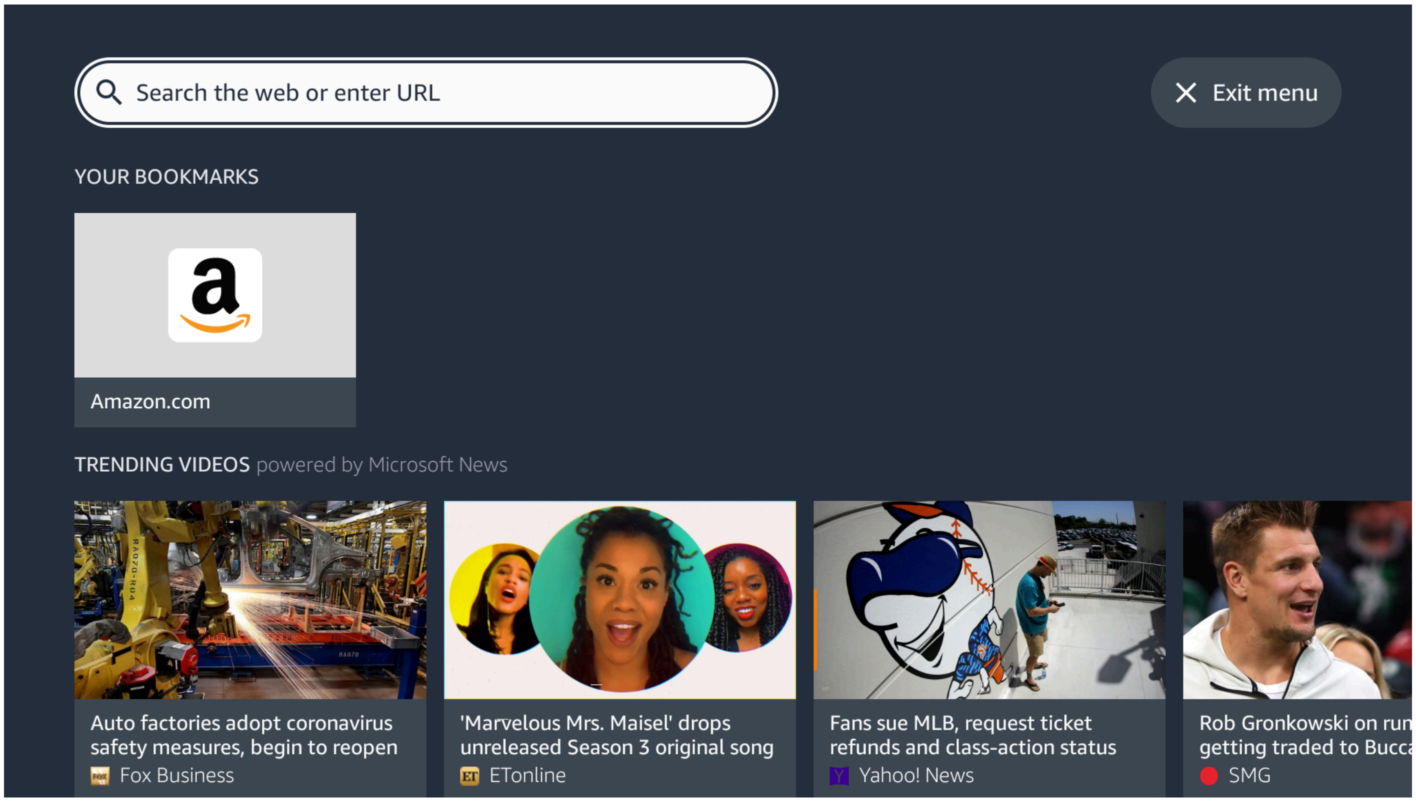
Amazon Silk is the web browser for the big screen. Silk brings limitless content from the internet to your TV, for an immersive experience that's easy to navigate. Go to your favorite websites to watch videos, view live events, browse pictures and shop. Easily control videos and music with Alexa or your Fire TV remote.

With Amazon Silk on your Fire TV, you can

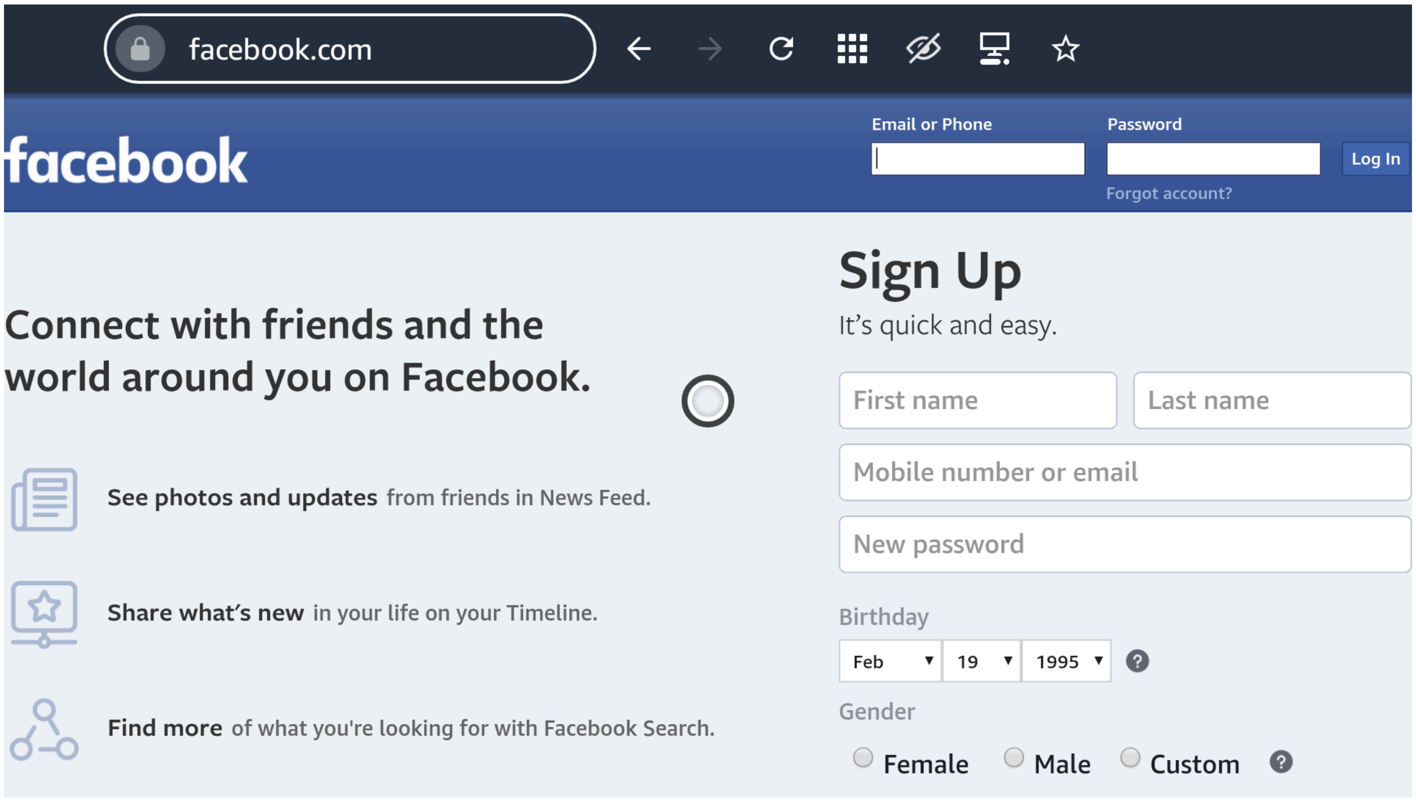
- Access limitless web content from your TV.



- Easily access your favourite websites through your bookmarks and trending videos.


Search the web or enter URL
Exit menu
YOUR BOOKMARKS
Amazon.com
TRENDING VIDEOS powered by Microsoft News
Auto factories adopt coronavirus safety measures, begin to reopen
Fox Business
'Marvelous Mrs. Maisel' drops unreleased Season 3 original song
ETonline
Fans sue MLB, request ticket refunds and class-action status
Yahoo! News
Rob Gronkowski on run getting traded to Bucca
SMG

- Browse websites in private browsing mode.



facebook.com

facebook

Email or Phone

Password

Log In

Forgot account?

Connect with friends and the world around you on Facebook.

See photos and updates from friends in News Feed.

Share what's new in your life on your Timeline.

Find more of what you're looking for with Facebook Search.

Sign Up

It's quick and easy.

First name

Last name

Mobile number or email

New password

Birthday

Feb 19 1995

Gender

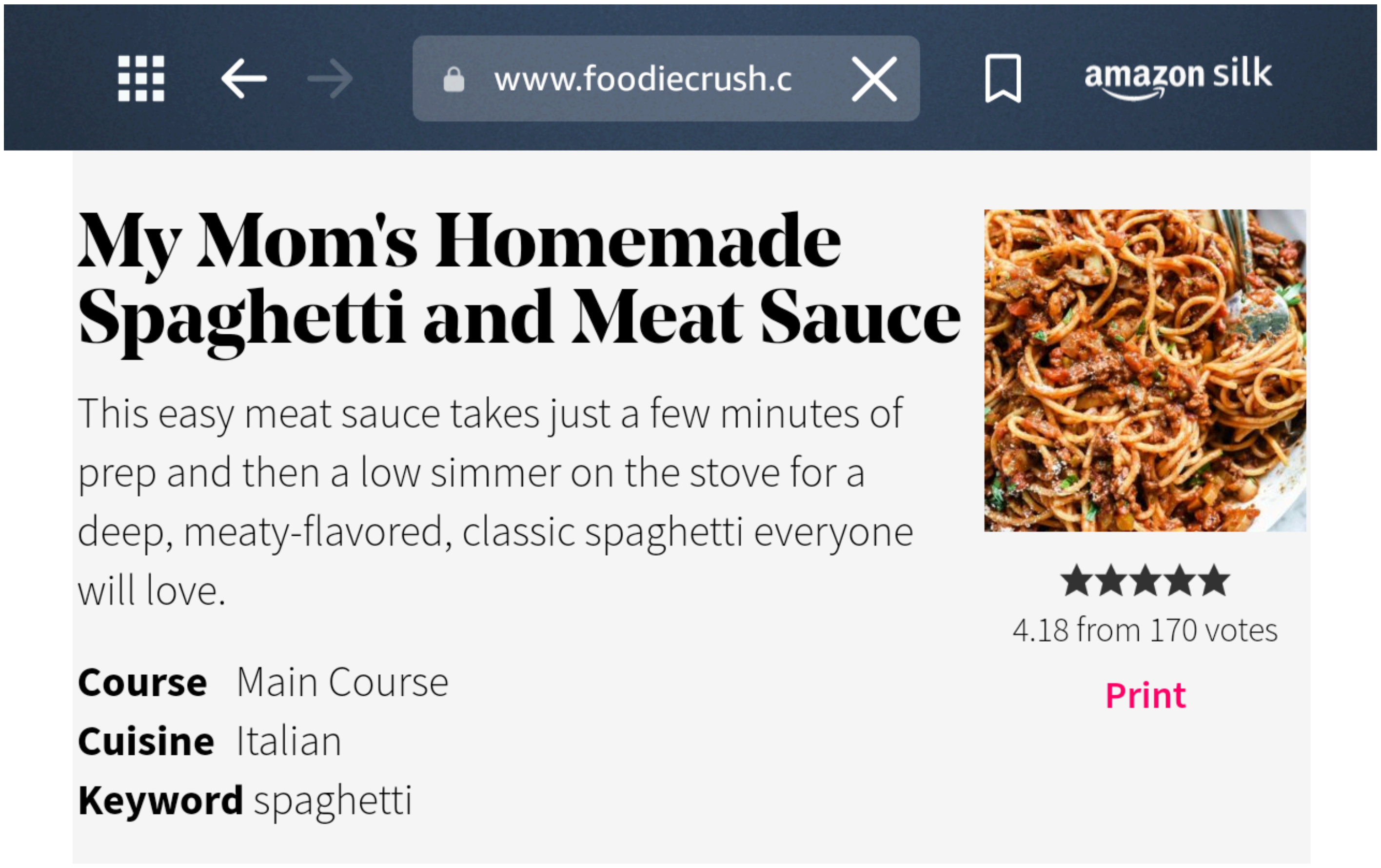
Female Male Custom

Just Say “Alexa, open Amazon Silk” on your Fire TV to start browsing the web.

Silk on Echo Show

Amazon Silk can also be used to browse your favorite websites on the Echo Show. For example,

- Browse the web for your favorite food recipes.



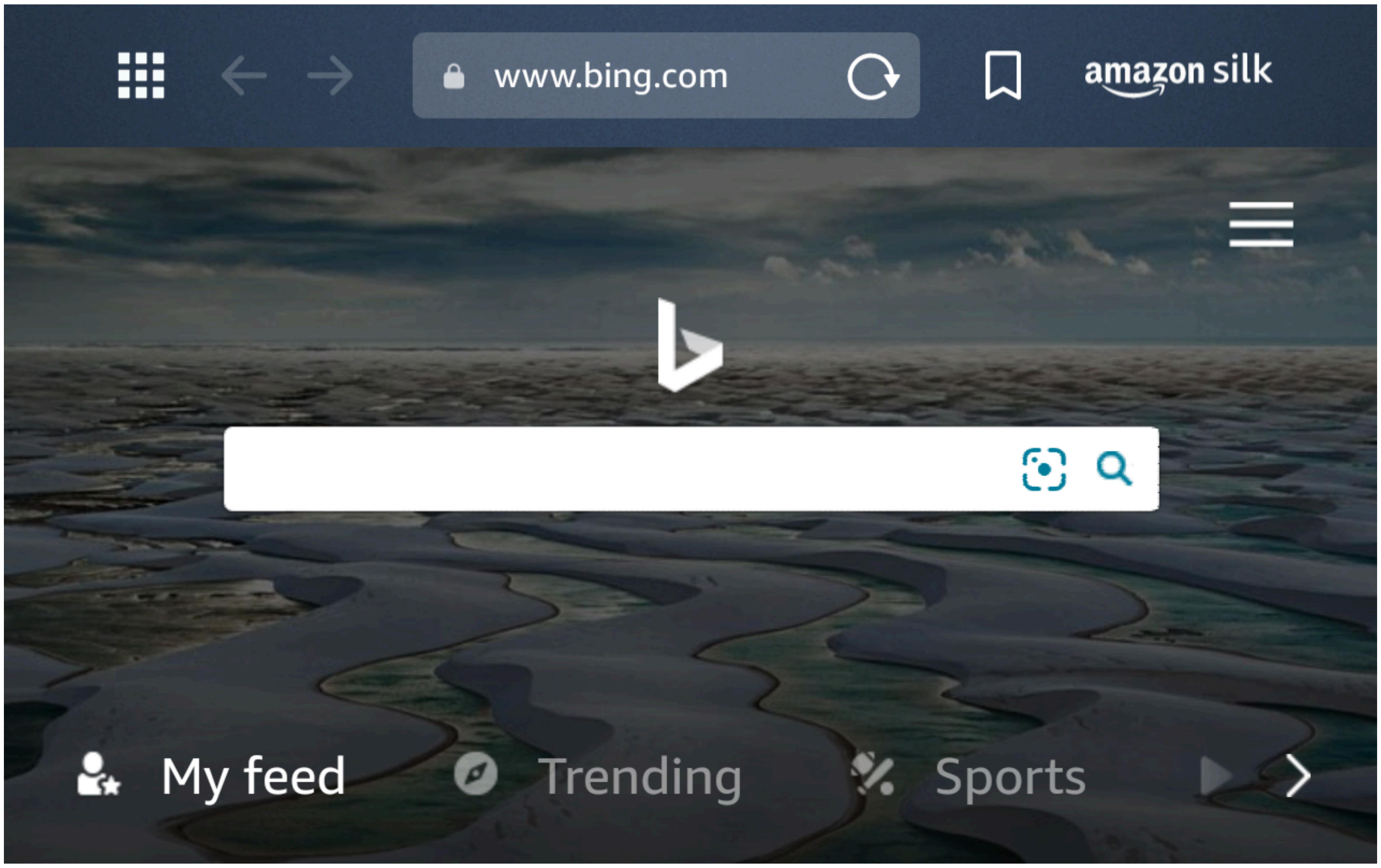
My Mom's Homemade Spaghetti and Meat Sauce

This easy meat sauce takes just a few minutes of prep and then a low simmer on the stove for a deep, meaty-flavored, classic spaghetti everyone will love.

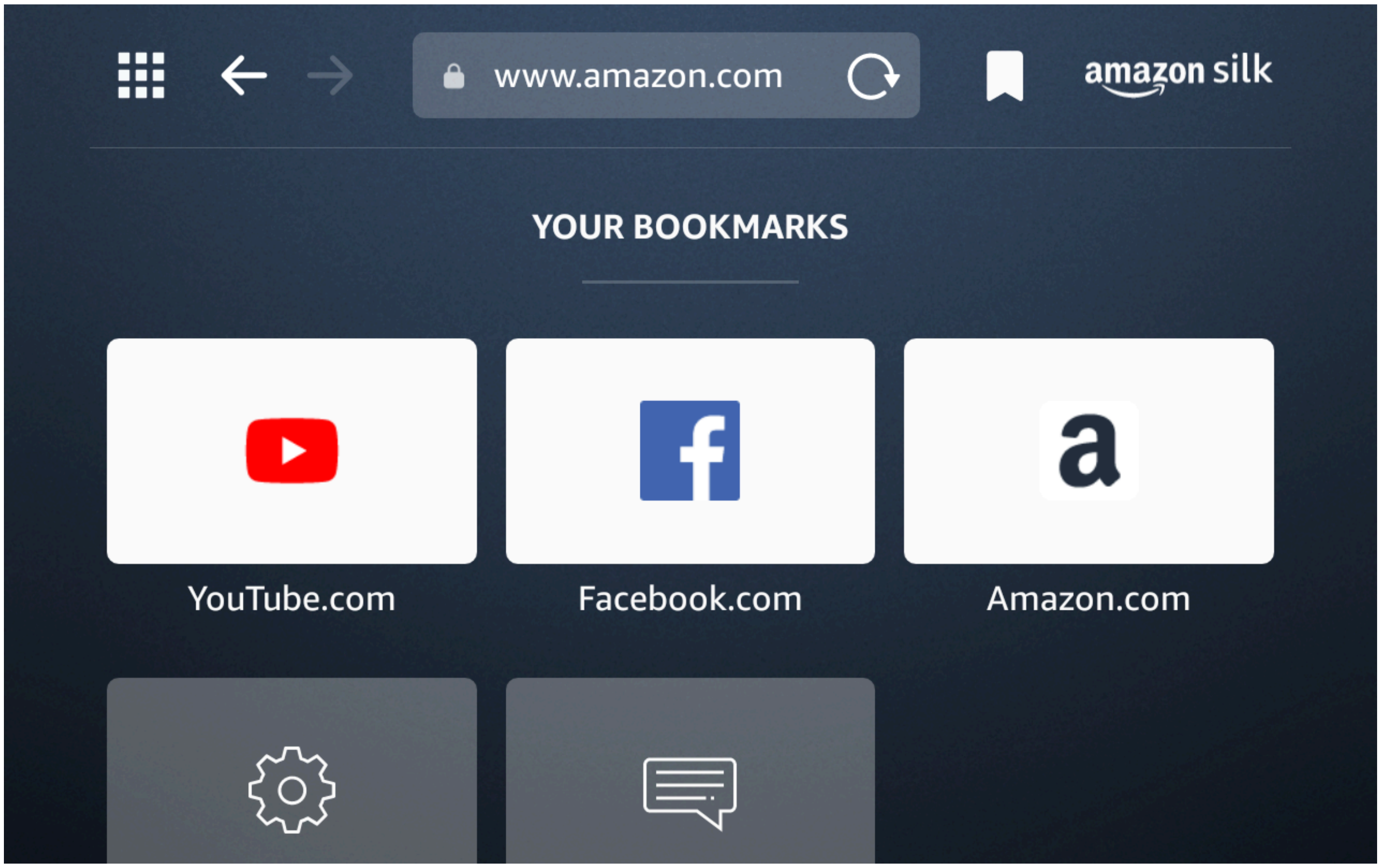
Course Main Course
Cuisine Italian
Keyword spaghetti

★★★★★
4.18 from 170 votes

- Search the web for tips and tricks for your projects and hobbies.



- Bookmark your go-to websites and searches.



Say “Alexa, open Silk” on your Echo Show to get started.

Getting started with Amazon Silk

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.

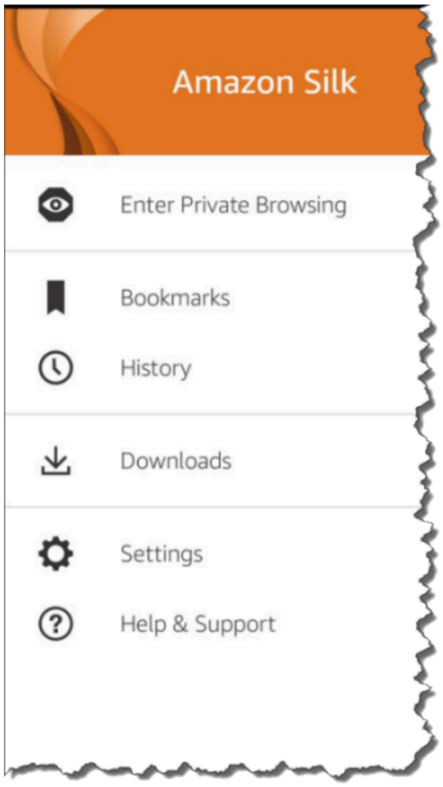
For tips on developing and optimizing web content for Silk, see the sections below:

- At *Developing web content for Amazon Silk*, you'll find information on media handling, screen resolution, the Silk user agent string, and other topics relevant to web development and content optimization.
- 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.

Determining the Amazon Silk build 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 find these values from the Amazon Silk user agent string.

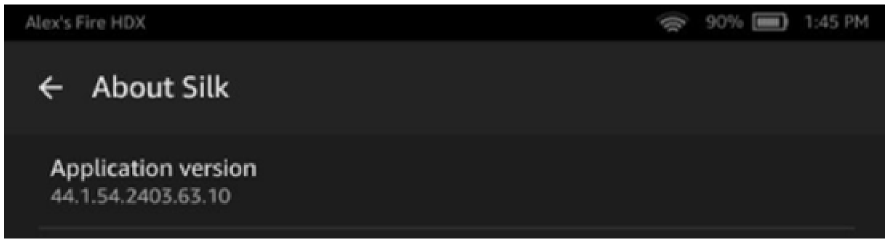
From the Silk browser, tap the menu icon. If you see this menu, you have the latest version of Silk. Use the following procedure to locate the build version and browser version. If your menu looks different, you may have an older version of Silk.



1. From the Silk menu, tap **Settings**, and then tap **About Silk**.
2. Locate the application number, which should look something like this: `44.1.54.2403.63.10`.

   In this example, the first set of numbers, `44.1.54`, is the browser version.

   The second set of numbers, `2403.63.10`, is the build version.



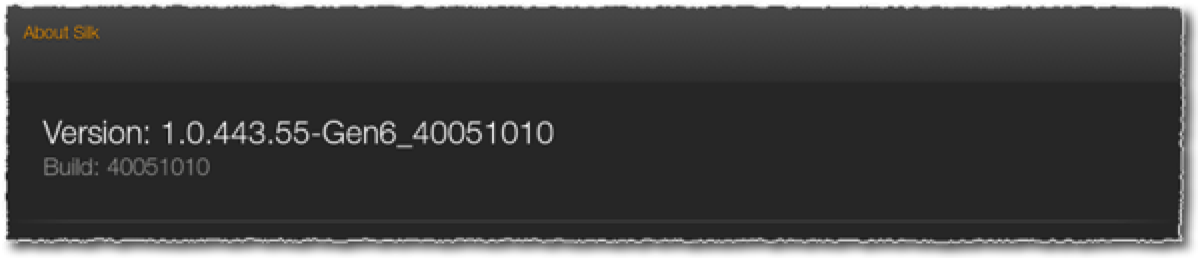
Application version
44.1.54.2403.63.10

How do I find the number for an older version?

In the Silk browser, type **`about:version`** in the address bar. Amazon Silk will display the build version string.

In this example, the first set of numbers, `1.0.443.55`, is the browser version.

The second set of numbers, `40051010`, is the build version.



Version: 1.0.443.55-Gen6_40051010
Build: 40051010

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 API support in Amazon Silk*.

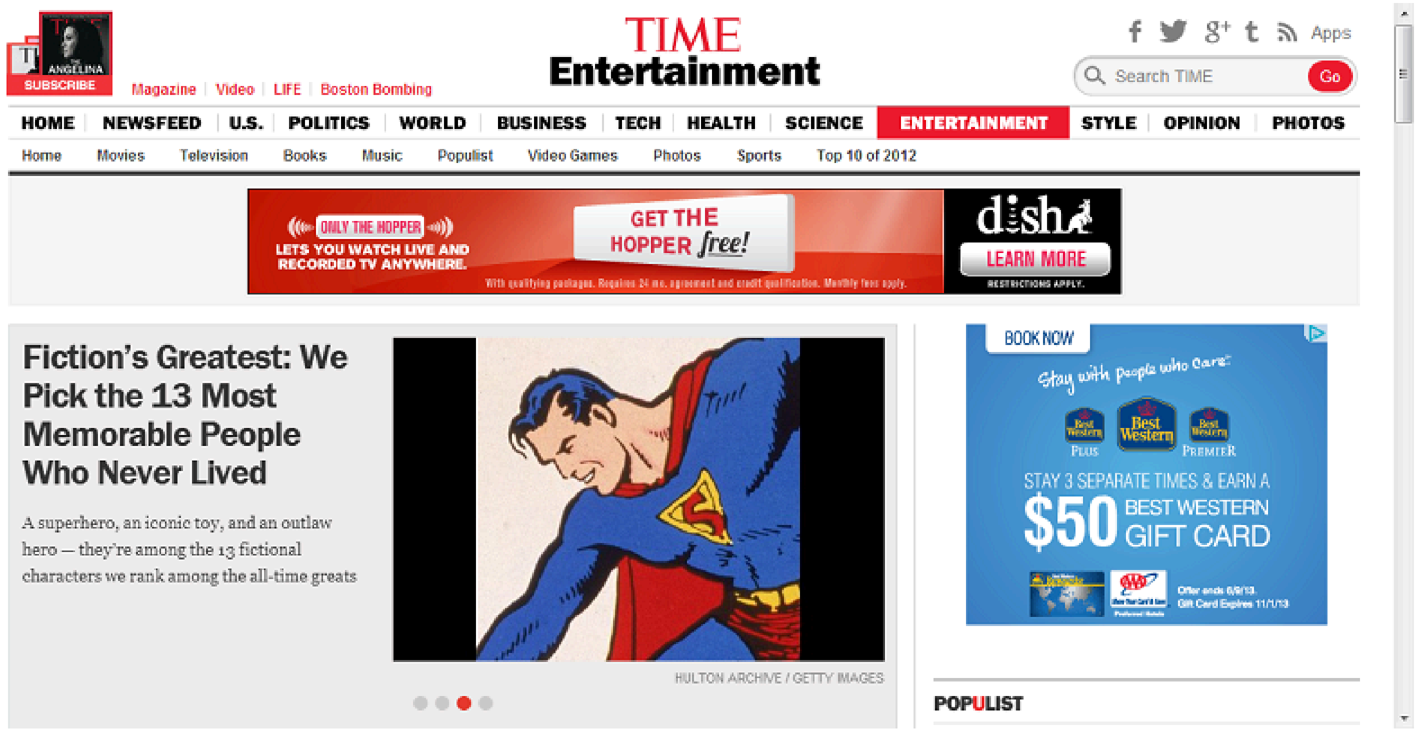
Topics

Learn about responsive web design

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 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.

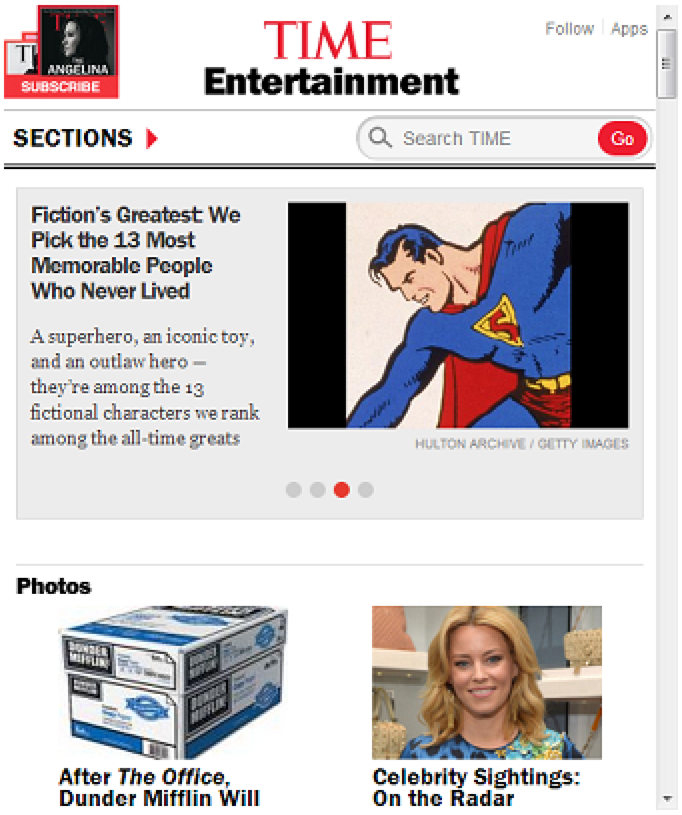
Here's the TIME.com site at desktop scale.


TIME
Entertainment
SUBSCRIBE
Magazine | Video | LIFE | Boston Bombing
Apps
Search TIME
Go
HOME | NEWSFEED | U.S. | POLITICS | WORLD | BUSINESS | TECH | HEALTH | SCIENCE | ENTERTAINMENT | STYLE | OPINION | PHOTOS
Home | Movies | Television | Books | Music | Populist | Video Games | Photos | Sports | Top 10 of 2012
ONLY THE HOPPER LETS YOU WATCH LIVE AND RECORDED TV ANYWHERE.
GET THE HOPPER free!
dish
LEARN MORE
BOOK NOW
Stay with people who care
STAY 3 SEPARATE TIMES & EARN A $50 BEST WESTERN GIFT CARD
HULTON ARCHIVE / GETTY IMAGES
POPULIST

Fiction's Greatest: We Pick the 13 Most Memorable People Who Never Lived

A superhero, an iconic toy, and an outlaw hero — they're among the 13 fictional characters we rank among the all-time greats

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



Fiction's Greatest: We Pick the 13 Most Memorable People Who Never Lived

A superhero, an iconic toy, and an outlaw hero — they're among the 13 fictional characters we rank among the all-time greats

After *The Office*, Dunder Mifflin Will

Celebrity Sightings: On the Radar

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 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 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.

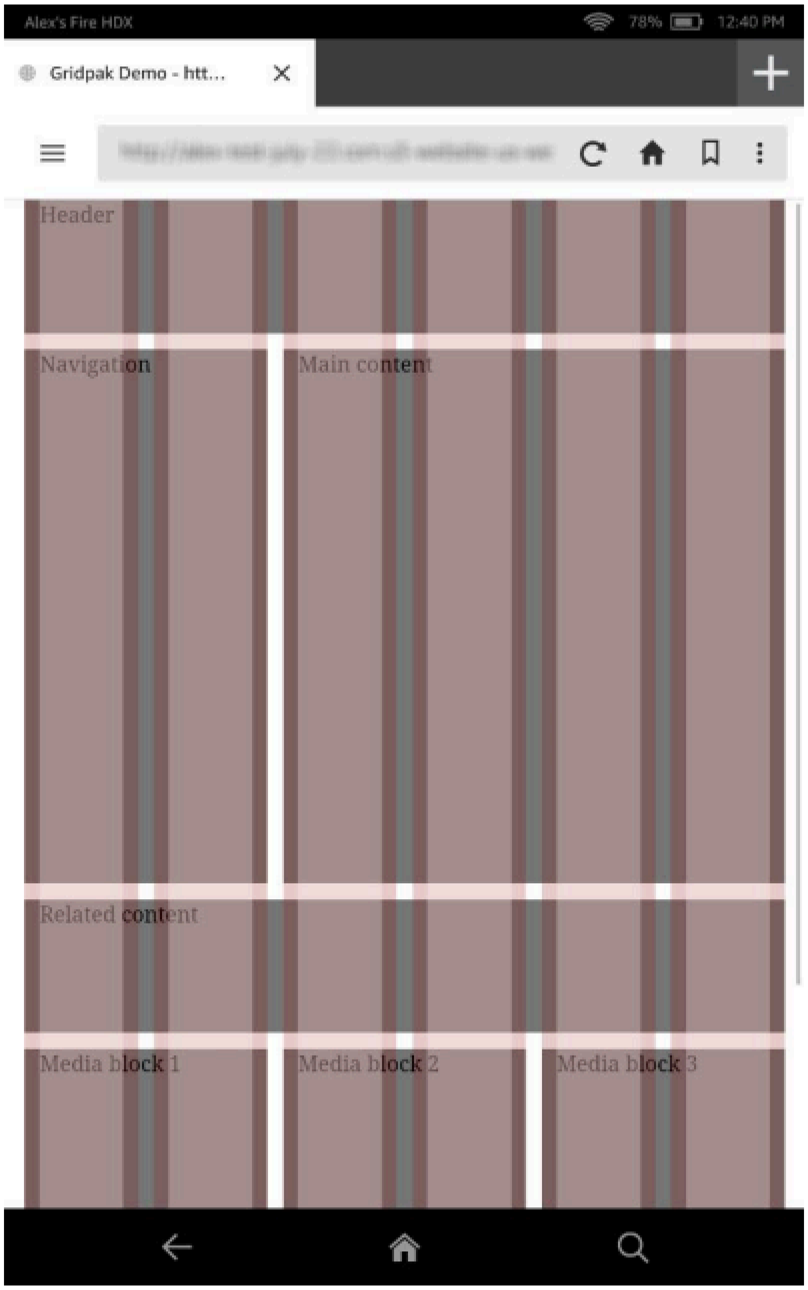
Another strategy (arguably a better one) is to create breakpoints based on your content, and not on device constraints. Designing breakpoints around content aims to make the layout user friendly and aesthetically pleasing across a range of possible viewport widths. With this approach, you can minimize 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.

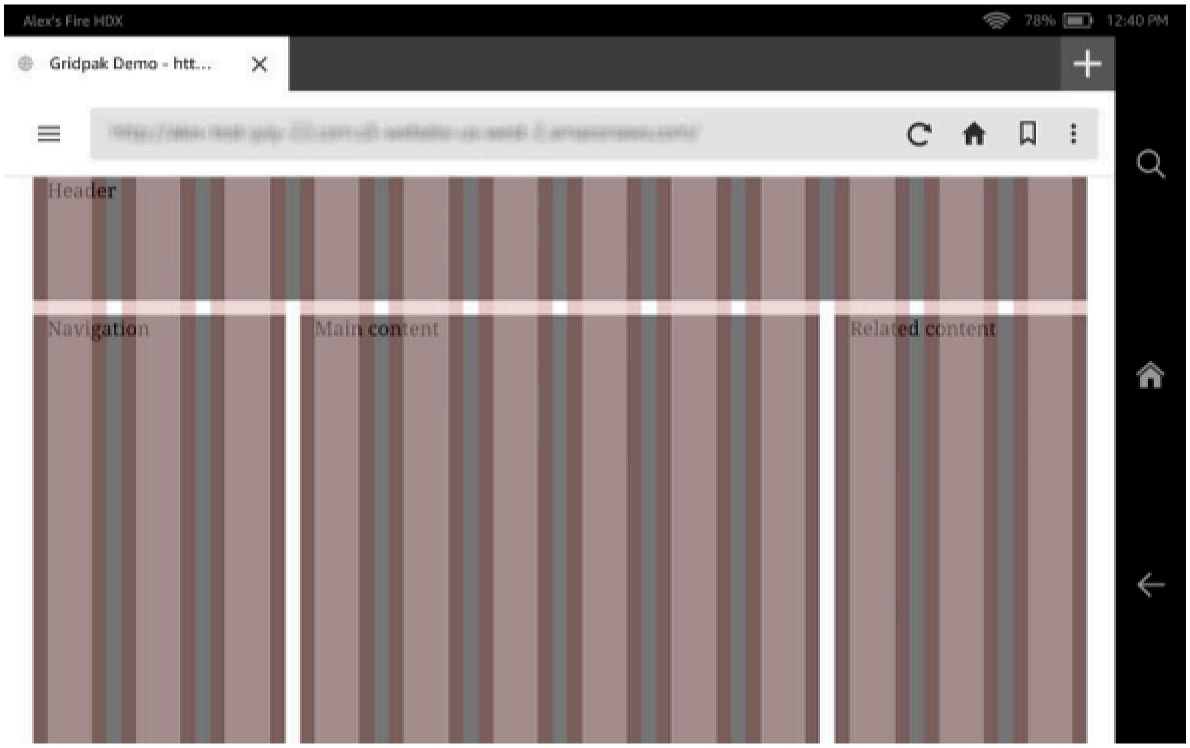
Here's our demo site in portrait view on a Fire HD 8.9" tablet:



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 Lalmalani'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

Learn about 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

You can also use user agent detection to target content, but this approach can be problematic. User agent detection 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. For more information about user agent detection, see Learn about user agent strings.

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.');
    } else {
        alert('Your browser does not support the canvas element.');
    }
    </script>
    </body>
</html>
```

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.

In an application, we 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

Learn about 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; Android android-version; 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) AppleWebKit/webkit-version (KHTML, like Gecko)
    Silk/browser-version like Chrome/chrome-version Safari/webkit-version
```

Mobile

```
Mozilla/5.0 (Linux; Android android-version; 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.

Template fields

- *`android-version`* – The Android platform version (for example, `4.4.3`).
- *`product-model`* – The value of Build.MODEL (for example, `KFTT`).

  For a complete list of the product models, see Learn about screen resolution.
- *`product-build`* – The value of Build.ID (for example, `IML74K`).
- *`webkit-version`* – Indicates the version of WebKit used (for example, `535.19`). The version can change whenever the Fire device receives a software update.
- *`browser-version`* – Indicates the version of the Amazon Silk browser (for example, `44.1.54`). The version can change whenever the Fire device receives a software update.
- *`chrome-version`* – The version of Google Chrome with which the Amazon Silk browser is compatible.

User agent string examples

Examples of the Amazon Silk User Agent String

Tablet

```
Mozilla/5.0 (Linux; Android 4.4.3; KFTHWI Build/KTU84M) AppleWebKit/537.36 (KHTML, like
 Gecko)
    Silk/44.1.54 like Chrome/44.0.2403.63 Safari/537.36
```

Desktop

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
    Silk/44.1.54 like Chrome/44.0.2403.63 Safari/537.36
```

Mobile

```
Mozilla/5.0 (Linux; U; Android 4.4.3; KFTHWI Build/KTU84M) AppleWebKit/537.36 (KHTML,
 like Gecko)
    Silk/44.1.54 like Chrome/44.0.2403.63 Mobile Safari/537.36
```

UA for Kindle Fire 1st Generation

Note

The UA for the Kindle Fire 1st Generation follows a slightly different syntax.

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

You can use user agent detection to target content, but this approach can be problematic. User agent detection 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. For more information, see Learn about feature detection.

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 Safari
\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)"));
}
```

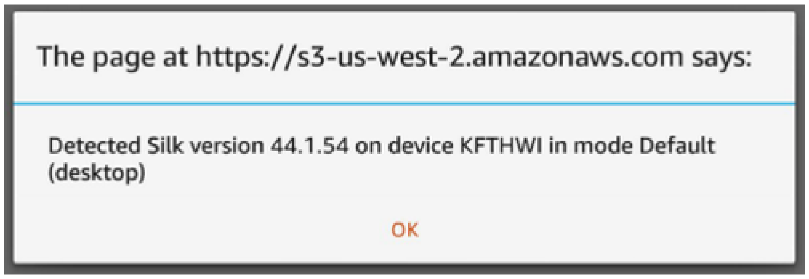
Detecting the Silk User Agent

To determine browser support for a given feature, the Learn about feature detection method is almost always recommended over user agent detection. However, 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 Safari
\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 page at https://s3-us-west-2.amazonaws.com says:

Detected Silk version 44.1.54 on device KFTHWI in mode Default (desktop)

The script detects 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 [Learn about user agent strings](#).

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, `44.1.54`).

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 Learn about user agent strings. To learn more about regular expressions in JavaScript, see Regular Expressions and RegExp at the Mozilla Developer Network.

Learn about media handling

Amazon Silk supports a wide range of media types.

Topics

- Images

- Audio
- Video

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.

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, with playback occurring in the browser. The default controls are improved from earlier 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. 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 Learn about feature detection 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">
</video>
```

The example tries to serve MP4 and then falls back to the WebM file. 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.

Learn about touch interactions

As you might expect, Amazon Silk is designed for touch interactions and fully supports Touch Events. 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 compatability 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 accomodate direct selection.
- The screen size of selectable elements should be sufficient to accomodate 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.

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.

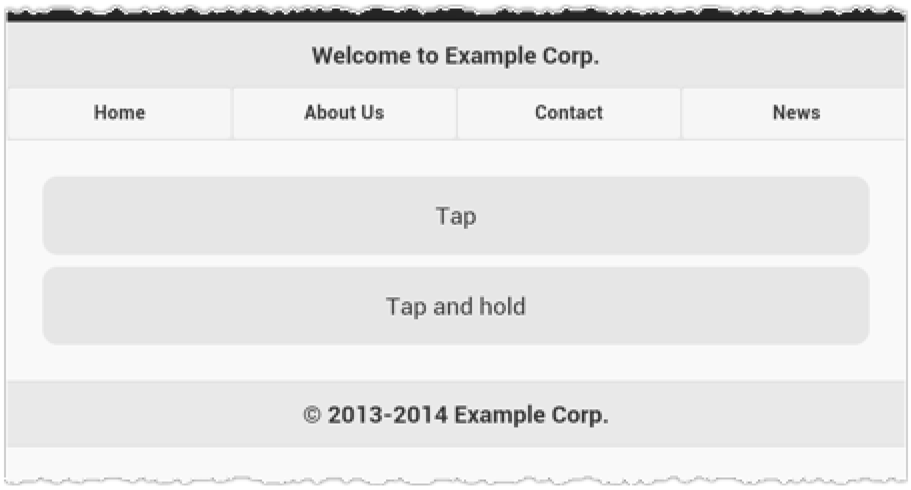
```
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, user-
scalable=no">
    <style>
    div.test {
        text-align: center;
        border-radius: .625em;
        background-color: #E6E6E6;
        padding: 1em;
        margin: .5em;
    }
    </style>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.2/
jquery.mobile-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>
    $(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>
```

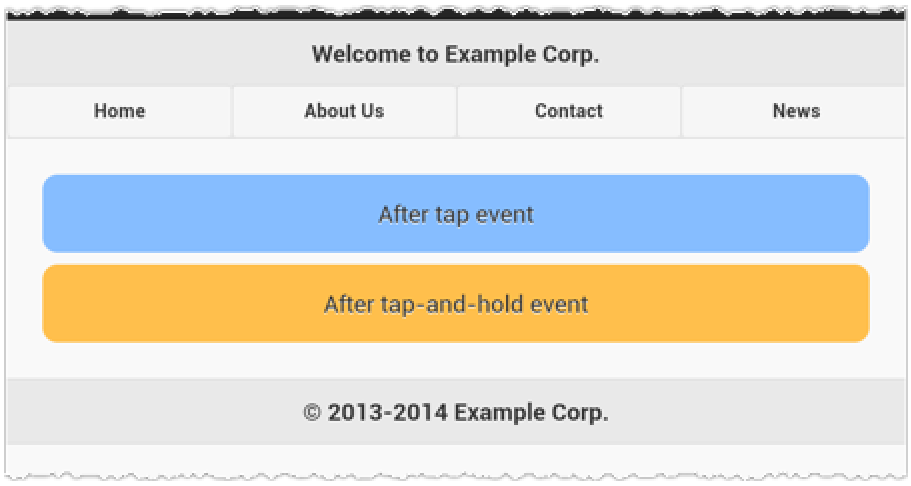
Both the `tap` and `taphold` events are jQuery Mobile extensions of jQuery built-in methods. The `tap` event is fired by a quick touch event on an element. The `taphold` event is fired by a touch-and-hold event (that is, a long press) on an element. (By default, the jQuery Mobile `taphold` event is triggered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

Learn about 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, product model, resolution, and scale factor for Fire tablets and phone. For full device specs, see Tablet Device Specifications.

Device	Screen size	Product model	Screen resolution (px)	Scale factor
Fire 7 (9th Gen)	7-inch screen	KFMUWI — Wi-Fi	1024 x 600	1.0 (mdpi)
Fire HD 8 (8th Gen)	8-inch screen	KFKAWI — Wi-Fi	1280 x 800	(tdvpi)
Fire HD 10 (7th Gen)	10.1-inch screen	KFSUWI — Wi-Fi	1920 x 1200	(hdpi)
Fire 7 (7th Gen)	7-inch screen	KFAUWI — Wi-Fi	1024 x 600	(mdpi)
Fire HD 8 (7th Gen)	8-inch screen	KFDOWI — Wi-Fi	1280 x 800	(tvdpi)
Fire HD 8 (6th Gen)	8-inch screen	KFGIWI — Wi-Fi	1280 x 800	(tvdpi)
Fire HD 10 (5th Gen)	10.1-inch screen	KFTBWI — Wi-Fi	1280 x 800	1.0 (mdpi)

Device	Screen size	Product model	Screen resolution (px)	Scale factor
Fire HD 8 (5th Gen)	8-inch screen	KFMEWI — Wi-Fi	1280 x 800	1.3 (tvdpi)
Fire (5th Gen)	7-inch screen	KFFOWI — Wi-Fi	1024 x 600	1 (mdpi)
Fire HDX 8.9 (4th Gen)	8.9-inch screen	KFSAWI — Wi-Fi KFSAWA — Wi-Fi + 4G LTE	2560 x 1600	2.0 (xhdpi)
Fire HD 7 (4th Gen)	7-inch screen	KFASWI — Wi-Fi	1280 x 800	1.5 (hdpi)
Fire HD 6 (4th Gen)	6-inch screen	KFARWI — Wi-Fi	1280 x 800	1.5 (hdpi)
Fire Phone	4.7-inch screen	SD4930UR	1280 x 720	2.0 (xhdpi)
Kindle Fire HDX 8.9 (3rd Gen)	8.9-inch screen	KFAPWI — Wi-Fi KFAPWA — Wi-Fi + 4G LTE	2560 x 1600	2.0 (xhdpi)
Kindle Fire HDX 7 (3rd Gen)	7-inch screen	KFTHWI — Wi-Fi KFTHWA — Wi-Fi + 4G LTE	1920 x 1200	2.0 (xhdpi)
Kindle Fire HD 7 (3rd Gen)	7-inch screen	KFSOWI — Wi-Fi	1280 x 800	1.5 (hdpi)
Kindle Fire HD 8.9 (2nd Gen)	8.9-inch screen	KFJWI — Wi-Fi KFJWA — Wi-Fi + 4G LTE	1920 x 1200	1.5 (hdpi)

Device	Screen size	Product model	Screen resolution (px)	Scale factor
Kindle Fire HD 7 (2nd Gen)	7-inch screen	KFTT — Wi-Fi	1280 x 800	1.5 (hdpi)
Kindle Fire (2nd Gen)	7-inch screen	KFOT — Wi-Fi	1024 x 600	1.0 (mdpi)
Kindle Fire (1st Gen)	7-inch screen	KFOT — Wi-Fi	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 Learn about user agent strings.

Keep in mind the following 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 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 to make forms more responsive.
- You can use media queries to style your site for a specific screen resolution.

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

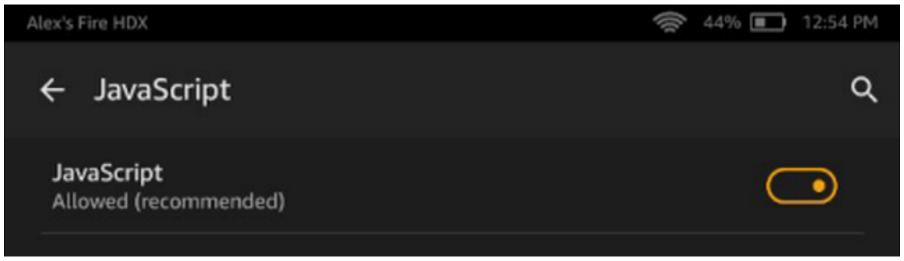
Learn about secure connections for Amazon Silk

Transport Layer Security (TLS) is a cryptographic protocol that provides security for online communications. Amazon Silk supports TLS communication between the device client on Fire devices and origin servers. For enhanced privacy and security, TLS 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 TLS connections. Amazon Silk currently supports up to TLS 1.2 for all Fire devices.

Learn about using JavaScript with Amazon Silk

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.

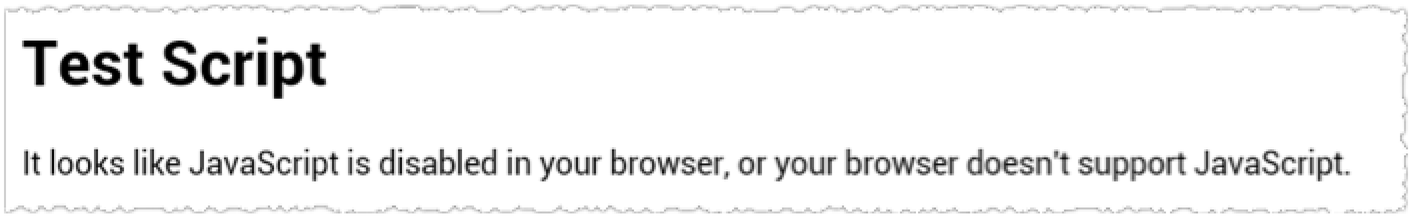


JavaScript
Allowed (recommended)

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:



Test Script

It looks like JavaScript is disabled in your browser, or your browser doesn't support JavaScript.

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.

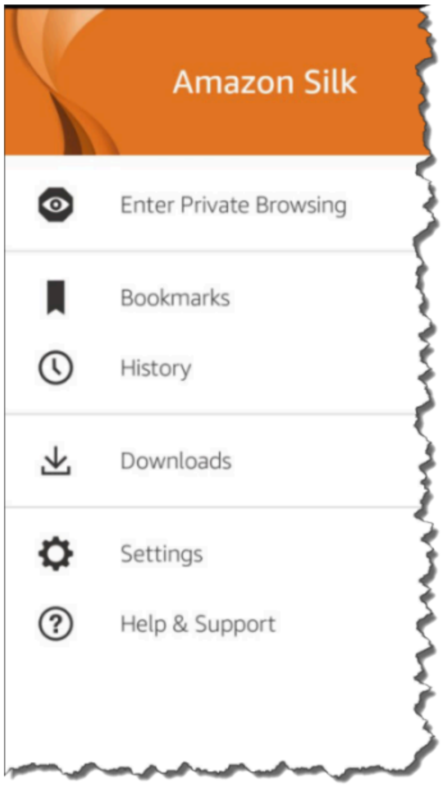
To learn more about improving load times for scripts, see the following resources:

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

Learn about remote debugging

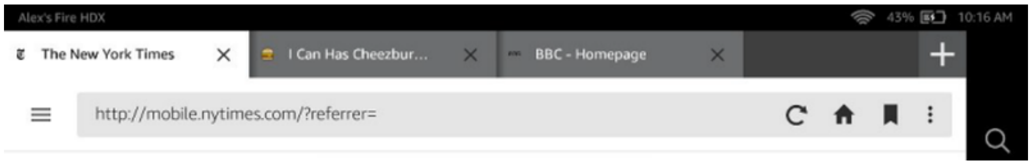
Amazon Silk provides remote debugging through Chrome DevTools. With remote debugging, you can use your development machine to inspect and interact with pages displayed in Silk on a separate device. Remote debugging must be enabled on the device, and Google Chrome must be installed on your development machine.

From the Silk browser, tap the menu icon. If you see this menu, you have the latest version of Silk. Use the following procedure for remote debugging. If your menu looks different, you may have an older version of Silk. Follow the procedure in Remote Debugging for Older Versions of Silk.

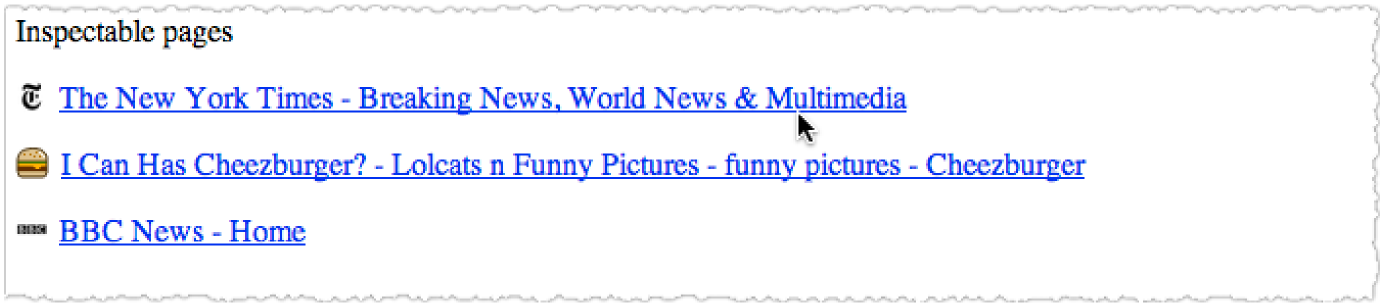


To remote debug your device

1. Install the Chrome browser on your development machine. You can download it from the Google Chrome download page.
2. Swipe down and tap **Setting**, and then tap **Device Options**.
3. Locate the **Serial Number**. Tap it several times until you see a message that says you're a developer.
4. Tap **Developer Options** and then under **Debugging**, tap to **Enable ADB**.
5. Connect your device to your machine via USB.
6. When the USB Compatibility message appears, tap **OK**.
7. On your desktop Chrome browser, navigate to **chrome://inspect**.
8. Your device should appear, displaying your open tabs.
9. 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.



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



Inspectable pages

The New York Times - Breaking News, World News & Multimedia

I Can Has Cheezburger? - Lolcats n Funny Pictures - funny pictures - Cheezburger

BBC News - Home

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

To learn more about remote debugging, see Remote Debugging on Android with Chrome.

Remote Debugging for Older Versions of Silk

If you have an older version of Silk, follow the procedure below.

Install Chrome and ADB

1. Install the Chrome browser on your development machine. You can 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.
3. 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 Fire tablets manufactured in 2012.
   - **Fire OS 3** – Select **Settings** > **Device**, and set **Enable ADB** to **ON**.

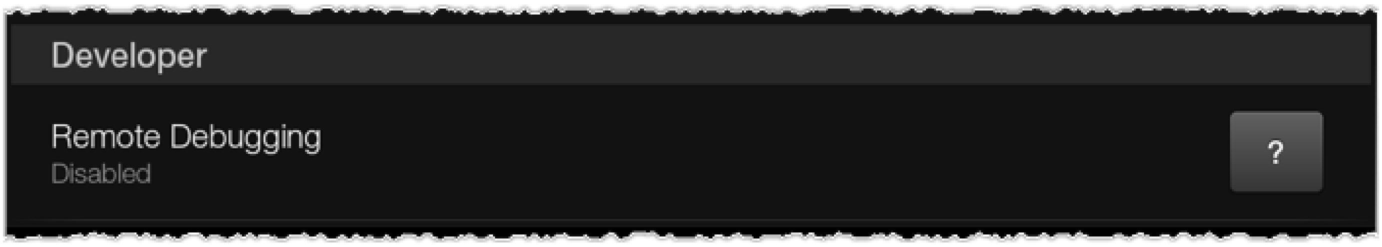
This version of Fire OS is available on Kindle Fire tablets manufactured in 2013.

- **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**.

4. 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:

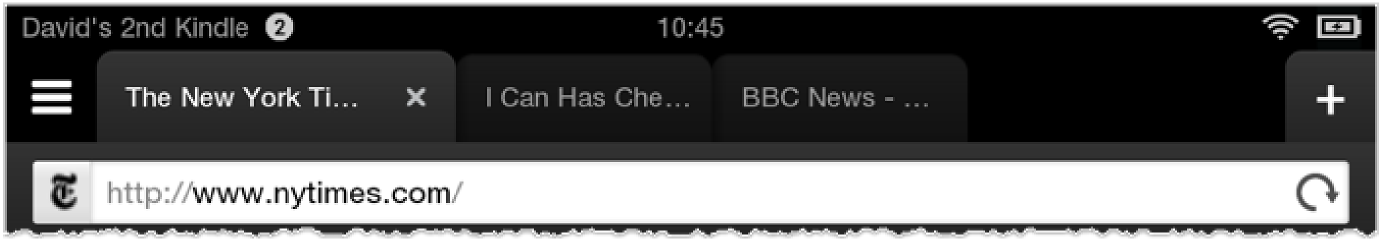
- **Unix Domain Socket** (only available on a Unix-based development machine) – Connect the device to the development machine via USB.
- **TCP** (available on Windows) – Ensure that both the device and the development machine are connected to the same network.

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:

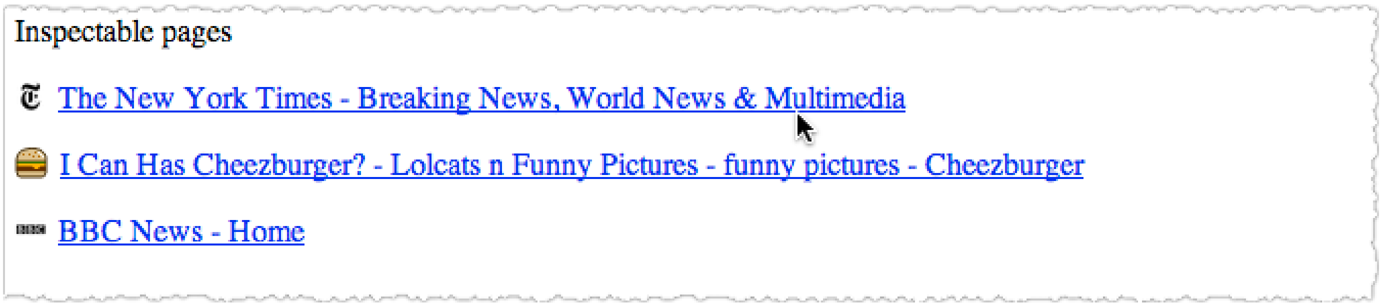
   - **Unix Domain Socket**: To inspect open tabs, run the following:

     ```
     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.



Inspectable pages

The New York Times - Breaking News, World News & Multimedia

I Can Has Cheezburger? - Lolcats n Funny Pictures - funny pictures - Cheezburger

BBC News - Home

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

Learn about the Do Not Track header field

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

To locate the DNT option

1. From the Silk browser, tap the menu icon, and then tap **Settings**.
2. Tap **Privacy**, and then tap **'Do Not Track'**.
3. You can turn DNT on or off.

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

Learn about the silk:is-user-entitled meta tag

The `<meta name="silk:is-user-entitled" content="true|false">` tag is a new meta tag introduced by the Silk web browser application. This tag is designed to inform Silk whether the current user is entitled to the content on the web page. It is particularly useful in scenarios where the webpage implements a paywall or other content restriction mechanism. Silk uses the value of this tag to decide whether to display a web page summarization option to the user.

When Silk encounters a web page with `<meta name="silk:is-user-entitled" content="false">`, it will assume that the current user is not entitled to the contents on the page. In this case, Silk will not provide a web page summarization option to the user on that page. If the tag is Silk present or has `content="true"`, Silk will assume that the user is entitled to the full content on the page, and the summarization option will be displayed.

Usage guidelines

The `<meta name="silk:is-user-entitled" content="true|false">` tag should be included in the `<head>` section of the HTML document. The content attribute can take one of two values:

- `true`: Indicates that the current user is entitled to access the content on the web page.
- `false`: Indicates that the current user is not entitled to access the content on the web page, typically due to a subscription requirement.

Follow these guidelines when implementing this tag:

1. **Consistency with Paywall/Content Restriction Implementation:** The value of the `content` attribute should be consistent with the paywall/content restriction implementation on the website. If a paywall is displayed to a non-subscribed user, the content attribute should be set to `false`.
2. **Omit the Tag for Entitled Users:** If the user is entitled to view the entire content of the page, the `silk:is-user-entitled` tag can be omitted, as Silk will assume entitlement by default.

By following these guidelines, website publishers can ensure that Silk accurately understands the user's entitlement status and provides the appropriate functionality, such as the web page summarization option, based on the specified content restrictions.

Examples

When a page is behind a paywall because the current user does not have a subscription, include the tag with `content="false"`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Paywalled Content Page</title>
    <meta name="silk:is-user-entitled" content="false">
    <!— Other meta tags and stylesheets -->
  </head>
  <body>
    <!— Paywalled content -->
  </body>
</html>
```

When a page is not behind a paywall because the user has a subscription, you can include the tag with `content="true"` or omit the tag altogether:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Free Content Page</title>
      <meta name="silk:is-user-entitled" content="true">
      <!— Other meta tags and stylesheets -->
  </head>
  <body>
    <!— Page content -->
  </body>
</html>
```

HTML5 API support in Amazon Silk

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

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.

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.

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 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.

For more information, see the W3C File API specification.

File System API

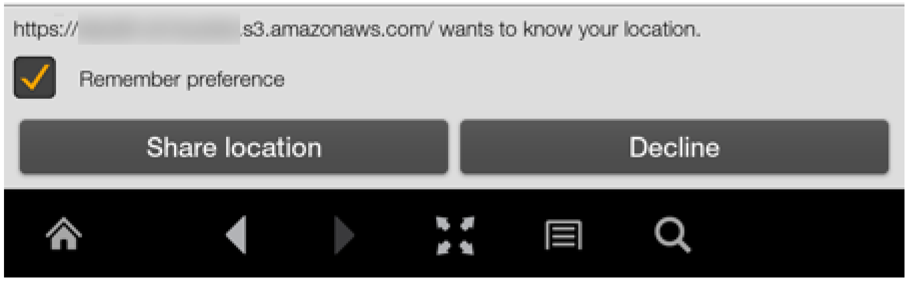
Using the File System API, a web application can create and interact with files in a sandboxed virtual file system on the client. The File System API gives web applications a way to store files, including large binary blobs, locally without using a database.

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.



https://[redacted].s3.amazonaws.com/ wants to know your location.

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.

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.

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.

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`.

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.

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.

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.

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.

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.

For 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.

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 support in Amazon Silk

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.

Topics

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:

HTML5 Audio Element

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 more information, see the HTML Canvas 2D Context specification.

contenteditable Attribute

Setting the `contendeditable` 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.

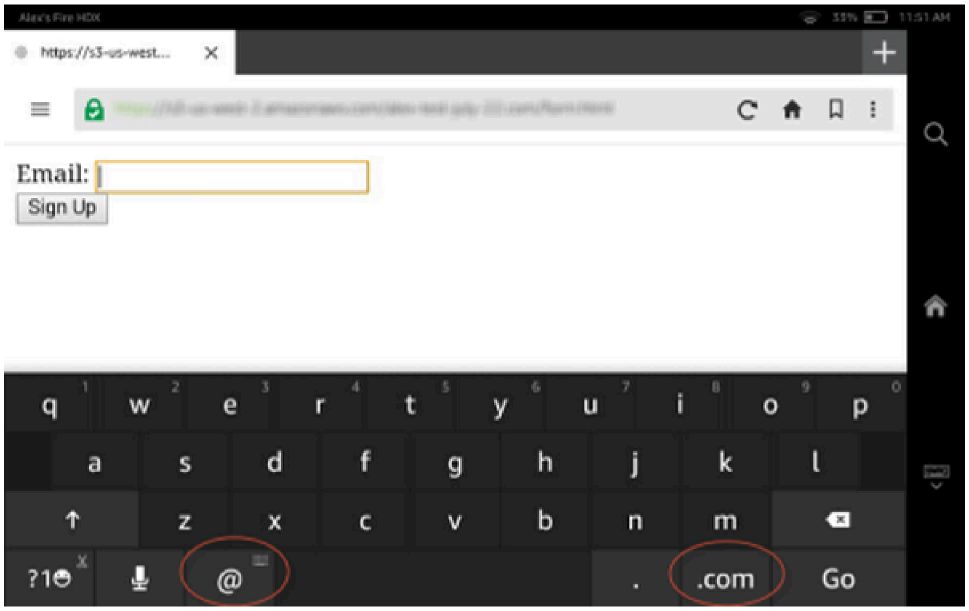
For more information, see Mozilla Developer Network: Content Editable.

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.

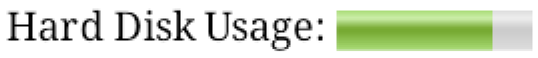
For more information about the <keygen> element, see HTML/Elements/keygen and Mozilla Developer Network: <keygen>.

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:



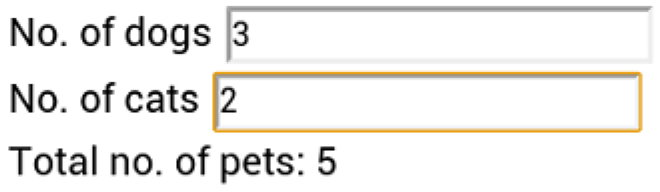
For more information, see HTML/Elements/meter.

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:



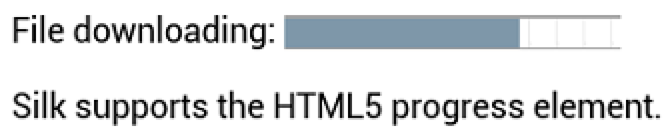
Note that the `oninput` event is not supported by Silk Gen 1. For more on the `<output>` element, see HTML/Elements/output.

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:

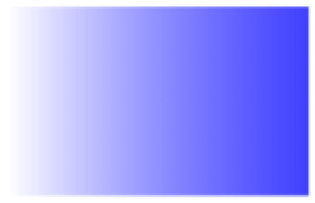
Silk supports the HTML5 progress element.

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>
```



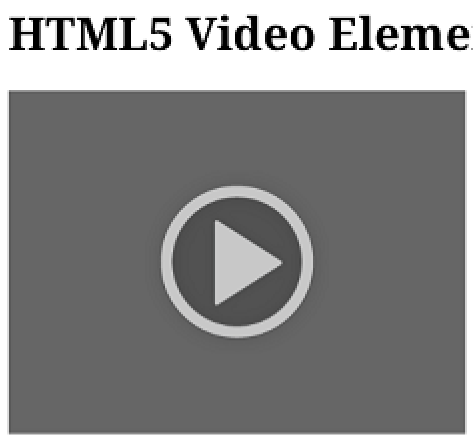
To learn more about SVG, visit the W3C SVG Working Group.

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:



For more information, see the W3C video element wiki.

CSS3 support in Amazon Silk

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 Learn about responsive web design. 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). You can also tailor the user experience to device orientation, so that your layout changes as the user goes from portrait to landscape mode.

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.

For more information, see CSS3 transform Property.

Transitions

Using a CSS3 `transition`, you can add an effect to a property change, without employing JavaScript. The result is an element that changes from one style to another, like an animation.

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.

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

- Learn about responsive web design
- W3C: Viewport META Element

Creating 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.

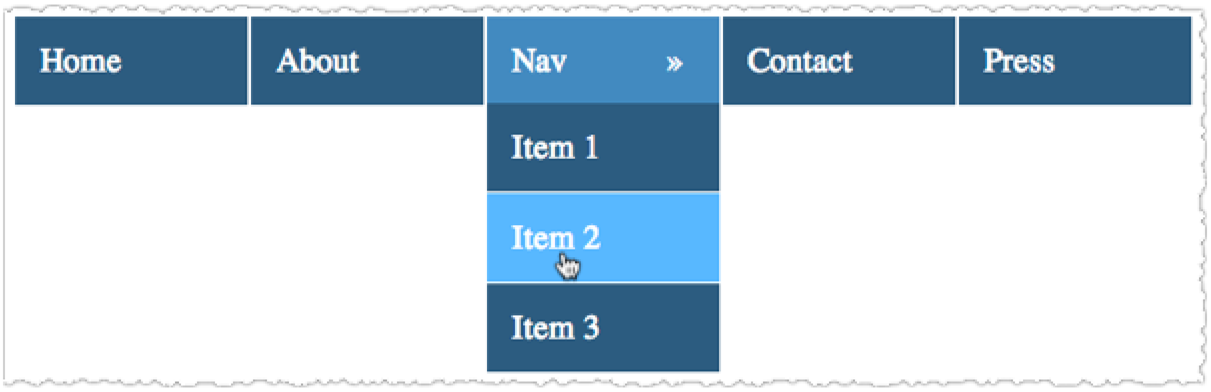
Let's look at an example. The following HTML document contains two unordered lists, one nested within the other. Each <li> element contains a link.

```
<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;
}
```

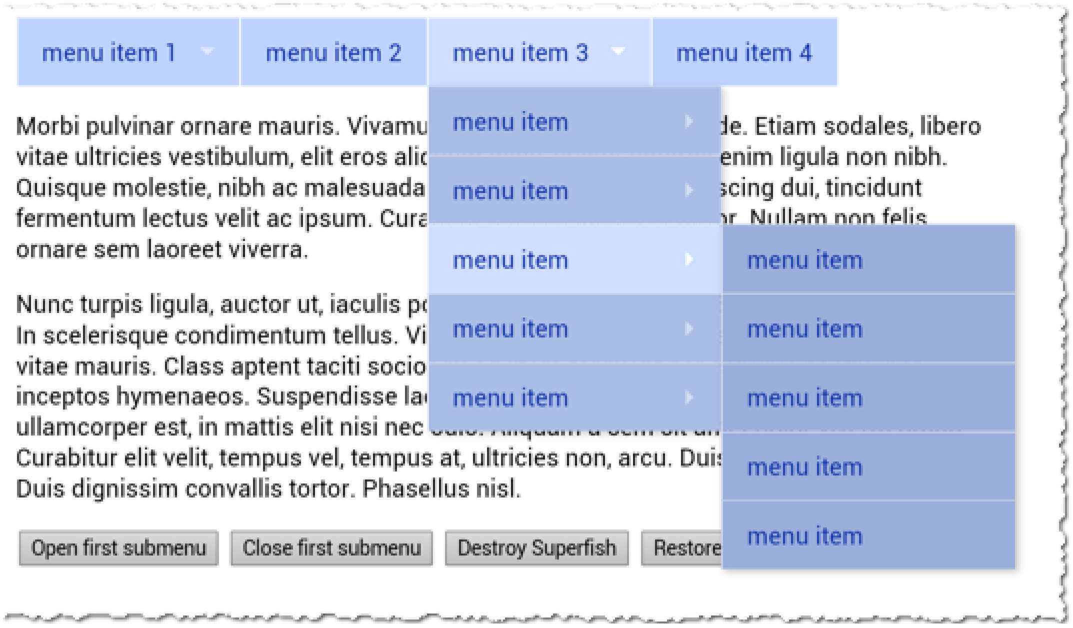
Notice how the `display` property is used. First, we use `display:none` to hide the nested <ul>, and then we use the `:hover` state to trigger `display:list-item`, which overrides the first `display` and shows us the nested <ul>. The result, rendered on a desktop browser, is shown below.



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: Determining 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.

Resources for web development

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

- Web Hosting
- 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*.

- **Latest documentation update:** September 18, 2015

Change	Description	Change Date
Updated documentation to include latest version of Silk.	This version of the *Amazon Silk Developer Guide* includes updated procedures, images, and user agent information for the latest Silk version. • Learn about screen resolution • Learn about remote debugging	September 18, 2015
Added pages on remote debugging and Do Not Track support.	This version of the *Amazon Silk Developer Guide* includes pages on the Silk remote debugger and on the Do Not Track header field: • Learn about the Do Not Track header field • Learn about remote debugging	August 11, 2014
Added a resources page and a user agent tutorial.	This version of the *Amazon Silk Developer Guide* includes a page of additional resources and a user agent detection tutorial: • Resources for web development • Detecting the Silk User Agent	May 9, 2014
HTML5 support table	This version of the *Amazon Silk Developer Guide* includes a table showing HTML5 feature support across Silk generations.	March 28, 2014
New feature detection page and WebGL section	This version of the *Amazon Silk Developer Guide* includes a page on feature detection and a section on WebGL: • Learn about feature detection • WebGL	January 3, 2014

Change	Description	Change Date
HTML5 element examples	This version of the *Amazon Silk Developer Guide* contains example code and images for the following HTML5 elements: • Audio Element • Video Element • Canvas Element • contenteditable Attribute • Keygen Element • Input Types • Output Element • Progress Element • Meter Element	July 12, 2013
New responsive web design page	This version of the *Amazon Silk Developer Guide* includes an overview of best practices in responsive web design: • Learn about responsive web design	June 5, 2013
Initial release	This is the first release of the *Amazon Silk Developer Guide*.	May 9, 2013