



Virtual University of Pakistan
FEDERAL GOVERNMENT UNIVERSITY

COURSE HANDOUTS

Web Development for Portable Devices

CS420

Order Your VU Books Now

**Get all Virtual University books in high-quality hard copy,
delivered right to your doorstep. Visit vubookshoppk.com**



Order Now

Table of Contents

No.	Main Topic
01	Introduction to Course
02	Introduction to Mobile Web Development
03	Why Responsive Web Design?
04	Importance of Performance
05	Analyzing Performance
06	Touch Interaction
07	Simple rules for touch UI
08	Click Delay
09	Supporting Touch
10	Mouse Emulation
11	Getting Started With Touch Events
12	Adding Mouse Support
13	Pointer Events
14	Pointer Code Example
15	Input Introduction
16	Regular Expression Validation
17	Two Important Features of Input
18	Input in older browsers
19	Wrapping in Label
20	telecolon URL
21	Sensors on mobile
22	Camera Access
23	Capture Extension

No.	Main Topic
24	Drawbacks of Capture Extension
25	Live Input
26	Navigator.getUserMedia()

Introduction to Course

Welcome to CS420 - Web Development for Portable Devices. This course develops the ability to apply familiar HTML, CSS, and JavaScript skills to the mobile web, an environment that has become a major part of everyday web use. The emphasis is on understanding how portable screens, changing network conditions, processing limits, touch interaction, mobile input controls, media, and device capabilities affect the way a web experience should be designed and implemented.

The discussion therefore begins with the mobile web itself and then moves through the tools, layout techniques, interaction models, form controls, performance concerns, and device capabilities needed to build for portable devices. The objective is not to replace HTML, CSS, or JavaScript, but to apply them with a clear understanding of the environment in which a mobile page is viewed and used.

Topics To Be Covered:

- 1) Introduction to Mobile Web Development
- 2) Differences of Mobile Web and Ordinary Web
- 3) Development Tools
- 4) Development tools for Desktop
- 5) Mobile Development Tools
- 6) Setup for Mobile
- 7) Using Development Tools on Mobile
- 8) Reverse Port Forwarding
- 9) Mobile Tools for iOS
- 10) Mobile First
- 11) Constraints of the Canvas

Introduction to Mobile Web Development

Mobile web development uses the same foundation as ordinary web development, but the context is different. A page must remain readable, usable, and efficient across narrow screens, changing orientations, different pixel densities, touch input, and varying network conditions. This topic builds that foundation step by step.

A mobile page must present the most useful information without forcing the user to navigate a desktop-sized canvas. Development tools make it possible to inspect the real page on a device, while viewport settings and fluid dimensions allow the layout to follow the available screen. These ideas prepare the page for the responsive rules introduced later.

Introduction to Mobile Web Development

The mobile web uses the same core web technologies, but the design must be considered from the perspective of portable devices and mobile users.

The term mobile web does not describe a separate set of web technologies. It describes web content and applications viewed in the context of portable devices, where screen size, orientation, input method, connectivity, and device capabilities can differ substantially. A useful definition therefore focuses on the conditions in which the same web platform is being used.

Differences of Mobile Web and Ordinary Web

Mobile users generally need essential information and functions quickly, so clarity, accessibility, and careful content selection become central design concerns.

Mobile web development is ordinary web development with additional constraints and opportunities. It continues to use HTML for structure, CSS for presentation, and JavaScript for behavior, while also considering mobile-oriented browser features and APIs.

Smaller screens, variable bandwidth, and limited processing resources require careful prioritization. These constraints encourage a mobile-first approach in which the most important parts of the user experience are made clear before enhancements for larger screens are added.

The course concentrates on the platform fundamentals required by a mobile web developer rather than teaching a particular framework or a native-application wrapper. This keeps the principles applicable even when development tools and libraries change.

Mobile users commonly visit a page to reach a specific piece of information or complete a focused task. Content and navigation should therefore be clear, accessible, carefully selected, and quick to reach.

On a traditional website, depth and completeness are king: a user must be able to find all the information on a brand, a product. In many ways this is the opposite to a mobile web page as a mobile user only accesses essential information and functions.

- These differences entail that these websites' ergonomics must be adapted to phone and tablet format in order to allow the user to browse through the mobile webpages with ease. Development Tools

Difference of Mobile and Web Development

The key difference is prioritization: a mobile page brings essential information and actions forward instead of reproducing every desktop detail at a smaller size.

For example, the Google News experience on the desktop shows a lot of information, but the mobile experience focuses on just the news stories. Pushing all of the secondary information out of the way.

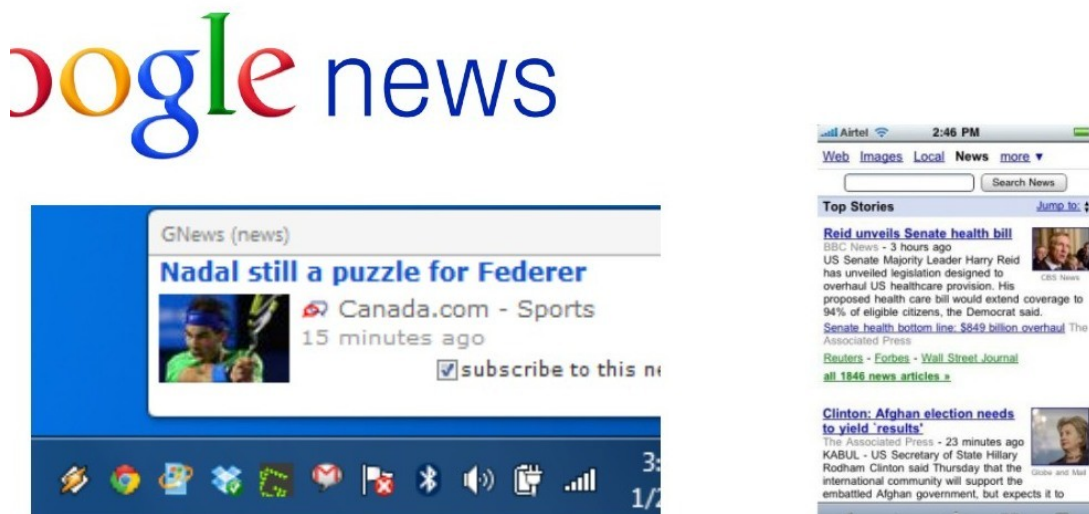


Figure 1. Desktop and mobile Google News experiences.

Development Tools

The development workflow combines browser inspection, live editing, debugging, and performance analysis for pages running on both desktop and mobile browsers.

Inspection is useful because it exposes the document structure, the styles applied to an element, the resources downloaded by the page, and the time spent performing browser work. Live editing provides immediate feedback, while the original source can be updated after the correct change has been identified.

- In this topic, we're going to be discussing the mobile development tools, showing you how to hook up the DevTools to your mobile browser. The Chrome developer tools, which enable you to analyze and debug, your web applications.

Now you may already, be familiar with the Dev Tools. But, what's really exciting, about them, is that you can now, easily ,use these tools, for your mobile debugging. Let's get started. Development Tools

Development Tools for Desktop

Desktop developer tools allow live inspection of HTML and CSS, direct editing, and analysis through network, timeline, and profile views before the same page is tested remotely on a device.

- On any web page, like www.vu.edu.pk, you can open the developer tools in the google chrome by clicking on the shortcut listed in tools, developer tools.

Now technically speaking you don't need a mobile device in order to take this course or even to test out most of what you'll learn. But you're missing a lot of the experience of mobile web development, without a device. Preferably at least one Android device.

You can use Chrome's great mobile debugging tools. Which we'll be using in this course. But, the more devices you have, and the more variety, the better. Development Tools for Desktop

- A traditional website is accessed via computer, with help from a mouse and a large screen. A mobile website, on the other hand, is accessed via smartphone or tablet, with a smaller screen and touchscreen navigation.

Chrome Dev tools provides a series of tabs that allow you to debug and inspect your web apps. For example you can select any element on the page by right clicking the element and selecting inspect element or as I'll do here you can click on the magnifying glass and hover over the area that you would like to inspect Development Tools for Desktop

- Let's change the text in one of the floor selectors. We can do that by simply expanding the element that contains it and then editing it live on the page.

You can also make live edits to the CSS. Let's take a look at changing the selected floor class. Here, I can set the background color to whatever I want. For example, blanched almond because who doesn't like blanched almonds. Because the background color is now pretty light, we may also want to change the text color. Chrome Depth Tools provides a handy color picker for that. We can pick any color we want.

Shift + clicking the color picker, allows you to cycle through the different color schemes. RGB HSL, and so on. If we scroll a little bit further, so our floor selector class, we can play around with the different settings there. You can use the scroll wheel to cycle through the different sizes. And we can set the border width in a similar fashion.

As you can see, Chrome Dev Tools provides with rich functionality to edit and inspect your pages. And than was just one tab. We'll be diving deeper into the Network, Timeline, and Profiles tabs.

Setup For Mobile

Remote debugging depends on enabling developer mode and USB debugging before connecting the mobile browser to the development machine.

After the device is prepared, the desktop tools can discover the mobile browser and inspect an open page. This arrangement preserves the real mobile rendering and input environment while providing the larger screen and debugging controls of the development computer.

- You also want to make sure you're set up correctly to debug and profile your web apps on mobile. We're going to show you how to do that with chrome dev tools and chrome for android.

The set up is simple. All you need is an Android device, a USB cable, and your development machine. Let's take a look. Before you get started you need to turn on the Developer Mode in your Android device.



Figure 2. Mobile debugging setup.

This may be different on any given device and you can check your device's manual on how to do this. In many cases though, you need to go to your device's settings, click on About Device and then click on Build Number seven times. Seriously.

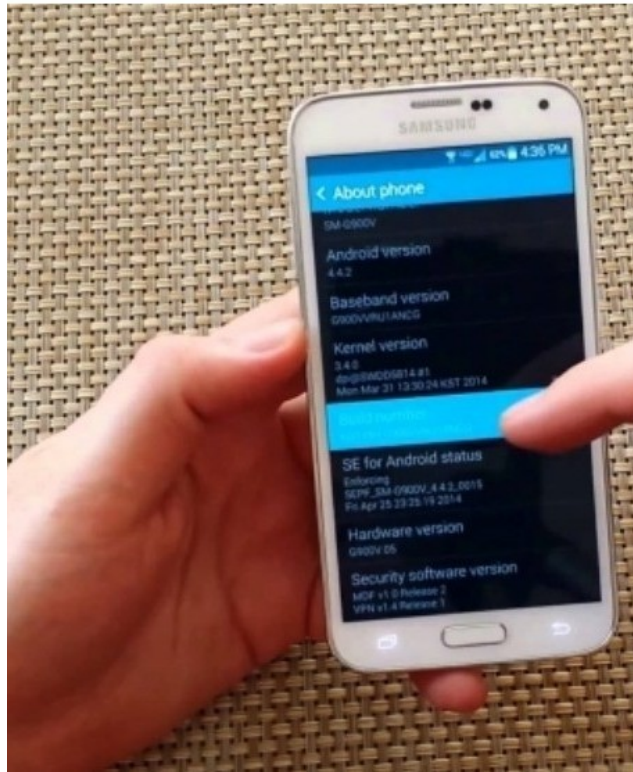


Figure 3. Enabling developer mode.

- Next you'll want to turn on USB debugging. Again, this varies slightly on your given device, but this is usually located in the developer options.
- We also need to make sure we have the right tools. On my laptop, I have Chrome Canary and on my mobile device, I have Chrome Beta installed.
- Now that we have everything set up the way we need,
- open Chrome on your development machine and go to Chrome inspect. Make sure the site you want to debug is open on your mobile device and then connect your laptop to your mobile device via USB.

Then confirm that you want to allow USB debugging. Back in our development machine, we can see a list of the attached devices and the Chrome tabs that are open on the devices. You can even open other tabs. We can also focus on specific tabs, you can reload them And you can even close a tab.

Using Development Tools on Mobile

Remote inspection connects the mobile browser to desktop developer tools, allowing the page running on the device to be examined and adjusted from the development machine.

- The best part of course is that you can inspect the pages that are running on your mobile device, from your development machine let's take a look.

One of my favorite new features is the new screen cast mode. This allows you to drive the experience on your mobile device from your development machine. You can click on links, and see them update simultaneously on the device. As well as on your desktop.

While continuing to cast a video in full screen mode to your TV. Now you can! With Chrome 35.0.1900.0 (or later), when you go full screen (with both HTML content/video and Flash) when you're already casting a tab, we'll size things correctly for TV, but show that content within the tab. On the TV, content should fill the entire screen. Using Development Tools on Mobile

To try this feature, you do need to be running the very latest and greatest version of Chrome. It's available today if you install Chrome Canary (<https://www.google.com/intl/en/chrome/browser/canary.html>) as a secondary browser on your computer, and it works on Windows, Mac, Chrome OS, and Linux.

- As you can see, you have all the familiar features from the development tools available for mobile now.

Reverse Port Forwarding

Port forwarding makes a server running on the development computer available to the connected device, so a mobile browser can test work that has not been published online.

Now all of these examples, we're accessing a live site. But you can also setup Port Forwarding to allow your mobile device to access a local server on your development machine over USB. Let's take a look.

To do this, you want to make sure you have a server running on your local development machine. In this case, I'm going to use Python's simple HTTP server on Port 9999. Now to verify that it is actually working, I'm going to access that on the local machine and it's working fine. If I want to now access that same page on my mobile device, I need to set up Port Forwarding. I can go back to the Chrome Inspect page, click on Port Forwarding and set up a port forwarding rule.

In this case, port 9999 on local host or IP address 127.0.0.1. I enable Port Forwarding and click on Done. When I refresh this page, you'll see that Port Forwarding is now running on Port 8080, and 9999.

Mobile Tools for iOS

Safari provides the corresponding inspection workflow for iOS devices, while the underlying objective remains the same: observe the real mobile page with desktop debugging tools.

At this point, we're all set up. So let's try it out. I'll open a new tab, navigate to localhost 9999 on the mobile device. And the page is ready to go.

- Now, that was really easy and it's also possible to do this on mobile Safari with the web inspector using the iOS web kit debug proxy. Now, that's a little bit harder to set up.
- Now that you are equipped with the right tools we can get to work. In the next lesson, we're going to get started with a mobile user experience.

Mobile First

Mobile first means beginning with the constrained mobile experience, prioritizing the core user experience, and progressively adding detail for larger screens.

Starting with the smallest practical experience forces the design to identify essential content and actions. Additional space can then be used to enhance the layout instead of being required for basic operation. The result is a clearer base that can grow progressively.

- We will discuss now the tools which build designs that scale across multiple devices. And one of the first challenges with developing for the mobile web, is the display screen.

The typically small display screen compared to your desktop or laptop device really forces you to focus on what's critical for your users, in fact this is what's led to the recent adoption of the mantra mobile first. Now, mobile first isn't about designing for the mobile web before you even think about the desktop experience.

It's really a design philosophy of stripping down to your core user experience and then layering additional gravy on top if you have the appropriate space. For example you might put your company info links inside a menu rather than cluttering up your home screen with them.

- What is Mobile First?

Mobile first design is an approach outlined in 2009 by Luke Wroblewski. Simply, mobile first is an approach to responsive design: design for smaller screens first, then add more features and content for bigger and bigger screens. This design approach is also known as "progressive enhancement."

Constraints of the Canvas

A design created for one fixed screen size can require zooming or scrolling elsewhere; flexible layouts reduce that dependency.

The available canvas changes with device size and orientation. A layout tied to one set of pixel dimensions may either overflow or be scaled until its text becomes difficult to read. Relative dimensions and reflow allow the same content to use the changed space more effectively.

When creating a visual design layout, it's tempting to think of your design surface like a canvas, a fixed surface. Now web design used to be like this. You drew on this canvas, it was a fixed size and shape, just like an artist draws on a canvas. What do you think the problems with this will be, as you move to a mobile first design.

Fundamentally, the problem is all of these. With a layout that's designed for a particular screen size and aspect ratio, the user is going to have to zoom and scroll around in order to see everything and there will frequently be blank space left on the screen.

Screen Size for Mobile

The virtual viewport helped desktop-width pages fit on early phones, but it also created scaling and readability problems that viewport settings must address.

When Mobile browsers first came along the content on the Web wasn't designed for narrow small screen devices. It was designed for Windows that were around 1,000 pixels wide and wider than they were tall with easy scrolling. To show this content into a tiny Mobile screen since rendering a Web Page designed for 1,000 pixels across and a 320 pixel wide screen would mean you'd be scrolling a lot. Mobile browsers basically lied about the Window width. They made the Window act as if it were 980 pixels wide, even though the original iPhone was only 320 pixels across. This enabled sites that were designed for a 1024 by 768 screen, that is, that were around 980 pixels wide to fit on the Mobile screen. Although you needed to do a lot of Zooming to read the text sometimes. Unfortunately if

your site did not happen to match that 980 pixel width you were either going to overflow or underflow the screen. Either wasting space or forcing the User to Zoom. In order to control this, Apple provided a viewport meta tag to be added to your HTML to control the default. For how big should my screen act on this page? The default is 980 pixels. So, if you put 980 pixels here, it would have no effect. The Mobile browser already defaults to 980. But setting a viewport tells the browser how wide the content is intended to be, and then the browser scales to make that size fit on the device's screen. There are two ways to use this tag.

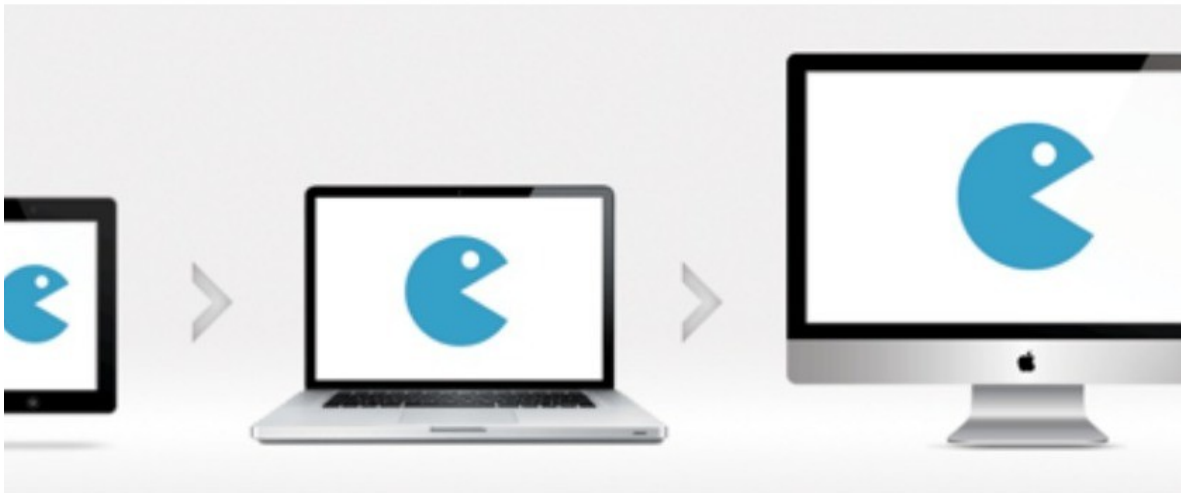


Figure 4. Content across multiple device sizes.

The first way lets us take a page that was designed to be precisely one size. This is an application I built that's using a fixed layout, and it's a bit smaller than default. It's 916 pixels wide. Notice that it's not using all the space on the screen because the browser by default is assuming the

- layout wants to be 980 pixels wide.
- So let's a viewport that tells the browser it was designed for 916 pixels across. What would you put in the viewport to tell the browser that this should be 916 pixels?
- Now the mobile browser will automatically scale that width to fit on the screen, no matter what size screen the user has, or whatever device width the user has.

Fixed Width

A fixed viewport can make a page fit, but the result is still scaled and may produce very small text and controls.

Let us take a page that was designed to be precisely one size. This is an application I built that's using a fixed layout, and it's a bit smaller than default. It's 916 pixels wide. Notice that it's not using all the space on the screen because the browser by default is assuming the layout wants to be 980 pixels wide.

- Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen. This was not perfect!! But a quick fix. So let's a viewport that tells the browser it was designed for 916 pixels across. What would you put in the viewport to tell the browser that this should be 916 pixels? The viewport is the user's visible area of a web page. The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen. Now the mobile browser will automatically scale that width to fit on the screen, no matter what size screen the user has, or whatever device width the user has.

Rescaling On Devices

Rescaling changes how a desktop-width page is fitted into a narrow screen. It can make the whole page visible, but it can also make text and controls too small.

Let's take a look at an app given below across a few devices with different width screens, an iPhone at 640 pixels across and a nexus 7 tablet in landscape, at double that width 1280 and note that it rescales properly on both of these devices. Note this takes into account orientation as well.

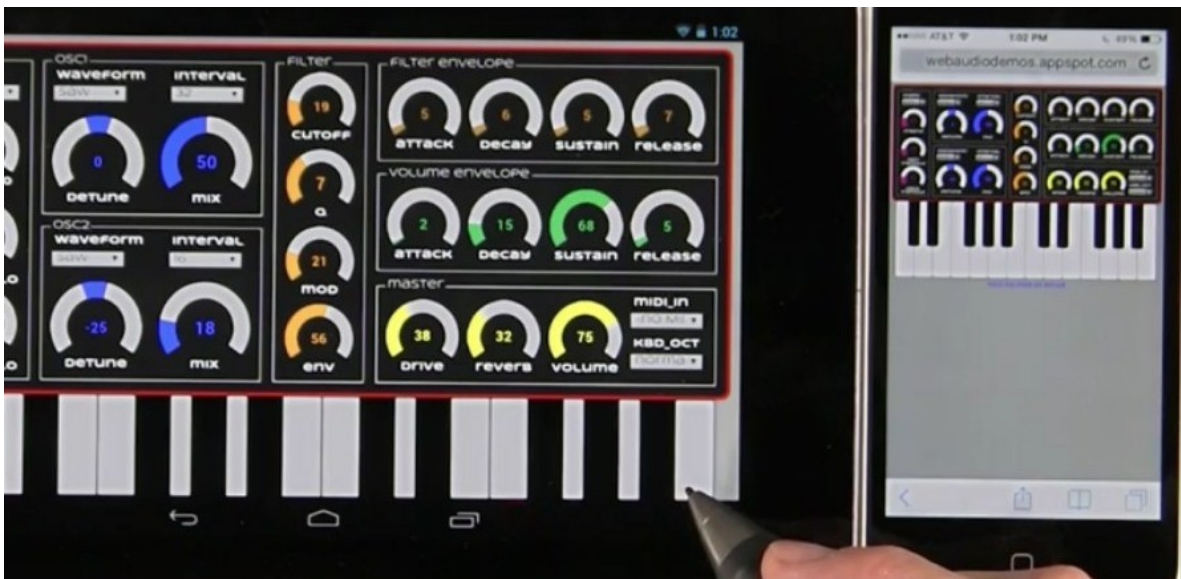


Figure 5. Application rescaling across devices.

The default scaling is to available screen width, so it will properly adjust when you flip the screen. There's also a height property on viewport which you could use instead of width, to control the scaling based on the height, but of course, most designs are built as width primary. That is, they're designed to scroll up and down.

Changing Viewport Settings

The viewport meta element tells the browser how the page width and scale should relate to the device, replacing the browser's assumed desktop-sized viewport.

The width setting determines the layout viewport used by CSS, while the scale settings determine how that layout is initially presented. These controls must be treated together: a correct viewport enables reflow, whereas scale alone merely magnifies or reduces an unsuitable fixed design.

Although typically viewport is only being set on Load, you can actually play around with the viewport settings in the development tools to tweak it, and get it just right. If I go into the page and set the Viewport meta element contents from the mobile dev tools It will change the page, as if it had been refreshed, however, and this is where it gets a bit confusing, the zoom level is maintained by the browser across pager refreshes, so when you change viewport settings in the source code, and you're reloading, be sure to actually close the tab first. Don't just hit reload or it won't necessarily show the effects on the screen.

Viewport settings can be tested interactively in development tools before the final value is placed in the page source. Changing the viewport meta content allows the developer to observe how width and scale affect layout, reflow, and readability on the connected device.

A mobile browser may preserve its current zoom level when a page is reloaded. If an updated viewport value does not appear to take effect, closing the tab and reopening the page provides a fresh viewport state. This avoids mistaking a retained zoom level for a failure of the new viewport declaration.

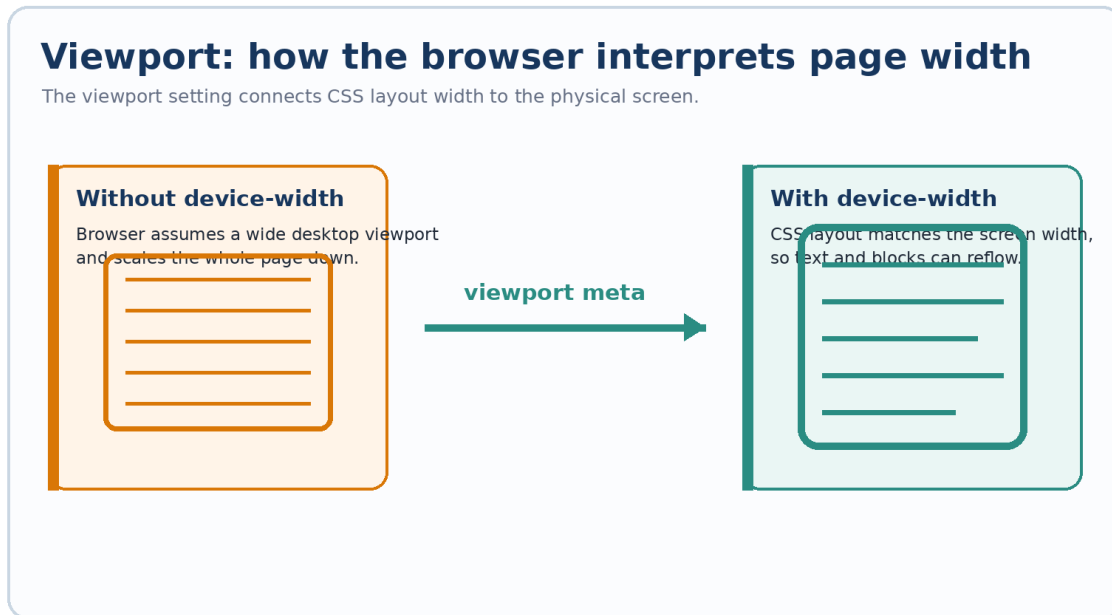


Figure 6. Viewport configuration and page reflow.

Downsides of Fixed Widths

A fixed width may fit one target but can overflow, underflow, or require zooming on another, which is why flexible sizing is preferred for portable devices.

- Your content is almost always going to be scaled.
- Users of some devices will have to zoom.

Text & UI will look small on a large desktop screen. A fixed width lets you tell the browser what width your webpage was designed for and the browser will automatically scale that width to fit to the screen no matter what size screen the user has or what device width the user has. Now what would be some of the downsides or side effects of using a fixed width viewport? It is true, that using effects with viewport, means that your content, is almost always going to be scaled, and not match the native resolution. And definitely, in some cases, users are going to have to zoom, to be able to see or read anything, particularly text.

Since, everything, all scaled depending on the size of the screen, fitting the current layout, precisely, into the users screen. Whether, it's a four inch phone or a 12 inch tablet, but actually, your text in UI will not change at all on the desktop. Because, currently, at least, none mobile browsers, don't respond to the view port at all. In some cases of course, it's the right thing to do, if you just want to get the UI to fit on the screen, to use, fix with view port, like in my sync app, but of course those dials are awfully small, which makes the UI very hard to use.

Text Reflow

Reflow allows text and layout to respond to the actual screen width instead of shrinking an entire fixed design.

When the browser uses the actual device width, lines of text can wrap within the available measure and blocks can move according to the layout rules. This is different from shrinking a complete desktop page, where the relationships remain fixed and every part simply becomes smaller.

Let's talk about how to make truly scalable pages. Now, HTML, by default, is supposed to reflow text anyway. And text sizes are supposed to be consistent. But the worst side effect of this viewport stuff, by default, is that, even with nothing set, the mobile browser is going to reflow text and render as if the window were 980 pixels wide. And then scale it to fit it on to the screen.



Figure 7. Text reflow.

This may mean that the text looks really small on my mobile browser by default. So, the second way to use viewport is if your page knows how to adapt to width. For example, if it knows how to wrap the contents based on the screen width. You can simply set the width to device width which tells the browser, my website knows how to adapt to your width. This is really the best approach. To build applications that scale their own layout and make intelligent decisions about how to do so, rather than just trying to scale a fixed layout to fit the screen.

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

Device Width

Setting the viewport to device width tells the browser to use the screen width as the page width.

The device-width value removes the assumed desktop-width canvas and makes the CSS viewport correspond to the portable screen. It does not make a rigid layout fluid by itself, but it gives flexible CSS the correct width on which to operate.

So let's try this out on this page. Let's add a device with meta element to a page, and refresh it on the mobile browser. This is what our page looked like before. Now with the

meta tag in place. Let's try reloading it. And you can see, the page now chooses a better size because it's reflowing at the native size of the screen.

Other Viewport Controls: Initial Scale

Initial scale controls the zoom level used when the page first appears; it should work with the chosen viewport width rather than compensate for a rigid layout.

On iOS, if you only set width to device width and you don't set the initial scale, like in this page, when you rotate the screen the iOS web engine will keep the same view port width and rescale it to fit across the landscape screen. It's just stretching the portrait layout, to fit, across the landscape width. Even though I've set width to device width, iOS is still scaling the landscape width. In fact, the interesting bit is that even if you load this page initially in landscape mode, it still thinks it's the portrait width, it just rescaling it to fit in the landscape screen. Now, if you have the same page, but you set the initial scale to 1 in the viewport meta-element along with setting device width, it'll change the viewport size when you rotate, instead of rescaling. You can see now the window size is 480 pixels across. It's the landscape width, not the portrait width. Other Viewport Controls: Initial Scale

Initial Scale

An initial scale of 1 normally presents CSS pixels at their intended size and gives the page a predictable starting point after loading or rotating.

I did want to mention at this point that there are other viewpoint controls. There's an initial scale property that lets you set what the browser's initial scaling factor will be. It defaults to one and usually, you don't really want to mess with it. If you change it to another number, this changes the initial zoom factor. And the user will probably have to pan or zoom. There is one very critical use to initial scale though. So, in short, this is what you really need to use as your default boiler plate viewport meta-element. You'll need the initial scale, so that IRS, when flipping from portrait to landscape mode will still scale correctly. An interesting side note I discovered is that on iphones - Although they are changing the viewport size properly if an initial scale is that, they are also changing the default font size for the document and orientation change, effectively zooming up the text when you go to landscape mode. This means you should probably set a default font size of the page, not just use an percentages. You may want to use a reset style sheet to do this, if you aren't already.

Fixed VS Device

A fixed viewport preserves an assumed width, whereas device width lets the layout use the actual screen and reflow its content.

So really, fixed viewport widths are historical. Resizing by default was an attempt to shoe horn the desktop web into a mobile device. Fixed viewport sizes were a quick way to provide some minimal controls on that resizing, with device width, gets us back to the same scalable sized canvas that the desktop web has, so the right way to do fluid flexible design, in the modern mobile web starts with this tag. This marker let's the browser know your one of the cool kids and you know what you are doing. There are also minimum scale and maximum scale properties too. Which is a way to limit the extents, that the system is allowed to scale the page on the device. That seems a little esoteric at first, but, there is one use that, I wanted to mention I don't want, you to think, that I'm biased.

Min and Max Scale

Minimum and maximum scale place limits on zoom, but overly restrictive limits can prevent users from enlarging content they need to read.

So, after describing how you need to add initial scale, for iOS to work as expected. I wanted to give equal time, for, Android. If your page, with device width happens to forcibly, overflow the page, like this page, where I've added an element that's very wide, Android will actually, do some rescaling, when you change screen orientation, to, try to get the whole page to fit onto the screen. Now, unfortunately, it gets it wrong, and it tries to zoom it the wrong way. Now, the only way that I found, to get around this, other than not overflowing the screen to begin with, is to set the minimum scale and maximum scale properties, to one, which will prevent the user from zooming altogether. As well as, disabling automatic scaling. Let's do that in the dev tools now. Now let's try rotating again. We're back to normal here, and now, you see that even in landscape mode, we get the proper width. Now, the problem with this, of course, is i can't zoom anymore. We really don't want to disable zooming, for, your users; this is an accessibility problem. So, don't do this in production pages, but, it can help your testing, in, general just don't overflow the page and you'll be good.

User Scalable

Disabling user scaling removes pinch-to-zoom and can create an accessibility problem, so it should not be used merely to hide layout defects.

There is one more property you can set. You can disable users zooming entirely by setting user scalable to no. Fundamentally, you really shouldn't do this. It's bad for accessibility and you're more likely to just annoy your users by preventing them from zooming than to make their experience any better.

Viewport Units

Viewport units size elements relative to the visible screen and can therefore respond directly to changes in viewport dimensions.

There is one more thing, on viewports. You're naturally, going to want to lay out, elements on the page, relative to the size of the viewport. Particularly, when you're sizing columns on the page, for example. So, we have a new unit type, in, CSS called, viewport units. You can use these units, to size things, in percentage of width of viewport or percentage of height of viewport. Without having to push percentage sizing, everywhere, which makes things a little easier. The really exciting thing, is, these unit types even work in desktop browsers, as well as in mobile browsers.

VMin and VMax

vmin follows the smaller viewport dimension and vmax follows the larger one, which makes them useful when orientation can change.

There are also vmin and vmax units, which let you size things based on the smallest or largest units of width and height. This helps make layouts that stay consistent across portrait and landscape mode. Using this feature, it's pretty easy to create a button that takes up 1/3rd of the biggest square that will fit in the screen. Now using these units, the button stays as large as it can be and still fit in a third of the space.

Separate Desktop and Mobile

A separate mobile experience increases maintenance and can create inconsistent features; a shared responsive experience usually keeps access more consistent.

So you've had about wraps it up, for how to design for a particular view port. With all of this complexity though, you might have noticed, some developers decide to build two separate applications, a desktop app. And a separate mobile web app, but there are some problems with doing this, which of these problems might you encounter with having separate mobile and desktop sites?

Rules for Mobile Only

If a separate mobile page is used, users should still be able to reach the complete feature set and move between mobile and desktop versions.

First, you really need all your features to be available and fully usable on the mobile version. If you try to guess what content or features your users won't miss on the mobile, you'll probably get it wrong. It might seem logical, right up until the point where I try to browse store inventory for my mobile device, only to find that feature is not available. Secondly, with two apps or sites, there's a natural tendency to focus on one or the other and to get out of sync between them. Maintaining two separate applications and keeping them fundamentally the same is hard to do. And finally, there's the problem that it's difficult to identify when you really want the mobile version. My tablet for example has nearly as big a screen as my laptop, and it's actually higher resolution than my laptop. Mobile design is actually more of a spectrum that spans all the way from low-resolution devices up into desktop design. At the very least, there are three simple rules if you want to offer a mobile-only site. First, you want to make sure a user can still get to the full site from the mobile site if they want to, in case you forgot any features. Secondly, you should put canonical URL in meta-information. This is commonly done for search engine optimization, but it's also important to keep your mobile site and your desktop site together. Finally, if someone links to a specific page on your desktop site, you shouldn't transfer them to your top-level mobile page. You need to transfer them to that specific page or its equivalent, on the mobile site. It's really frustrating when you get redirected to the top-level page.

Don't Block Access

Device detection should not become a barrier. A user may need a feature that the site has incorrectly assumed is unsuitable for that device.

It's great if you've sunk development money into developing a native mobile app, and it's fine if you want to make sure your users know it's available, but you shouldn't block access to your web application with a full page ad for your native mobile app. Make this offer unobtrusive, and make my decision to ignore it sticky. Now, you can use smart app banners, in iOS, to make this offer a little less intrusive, but if I open a URL in a browser, I probably want to stay in the browser.

Building Fluid Designs

Building a fluid design is a sequence of related changes rather than a single CSS declaration. The page first needs a viewport that represents the screen correctly. Fixed wrapper and column widths can then be replaced with relative or automatic dimensions, and images can be constrained so that they remain inside the flexible content area.

Fluid Layouts As we discussed already that one of the first challenges of developing for mobile devices is the screen. As mobile developers, we need to design pages that can adapt across various screen sizes and scenarios. This will prepare you for building fluid layouts, designs that can adapt across different screen sizes. And hopefully we'll break the bad habits of designing to the fixed page.

Blog Layout

The blog example exposes the limitations of a desktop-oriented fixed layout and provides a concrete design to improve progressively.

Now, if you resize this window, nothing really changes. Nothing adjusts other than the amount of margin space. It's really a fixed width layout. Now, let's take a look at this on the phone. Now here, I get a very different version of a blog. It fits well on the phone, but I'm only getting this because blog is running on Wordpress.

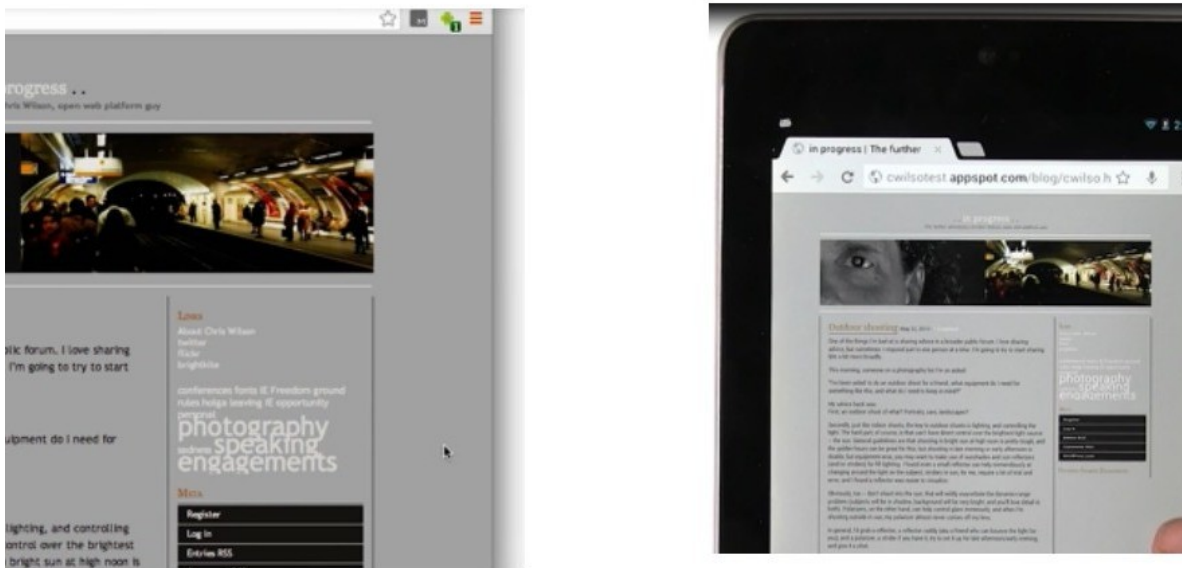


Figure 8. Blog layout case study.

And, Wordpress is configured to give a totally different version to phones. But, that's not actually what I want. Because I really want to have a shared layout between the two. Let's take a look at this on a intermediate device. My seven inch Android tablet. Now here, what I'm getting is essentially the desktop version. I can still scroll through the site but you can tell all the text is pretty small. Although interestingly, if I rotate to landscape mode, it adapts. Now why does it do that? Well it turns out it's our old friend viewport. There's now viewport set in this page. So, what does that mean? Does layout happen if the device flipped? Or does layout happen at the magic 980 pixel width and then get rescaled to fit on the screen?

Digging Around

Inspecting the existing markup and CSS reveals which fixed dimensions and layout assumptions prevent the page from adapting.

So if we dig around the page with the developer tools, we'd see a lot of pixel widths. Including the main content of the page, which is inside this page element. And you'll notice it has a width set in pixels. If I turn it off, well, the page is still not working properly. But at least we've started. Instead of this fixed code let's use a different technique, Fluid Layout. Fluid Layout means I should stop fixing all those widths in absolute terms like pixels or points and start thinking in terms of areas of screen real estate. Percentages of the widths, for example, and sizes based on the initial font or Em units.

Fixed Viewport Width

A fixed viewport may make the example appear to fit, but it does not solve the underlying dependency on fixed dimensions.

With all this complexity, you might ask, why don't I just set the viewport width to 760 pixels? The width of that column of text I showed you in the Dev tools, and call it a day. Is it because that would be too easy? Or because, then my site would scale the same layout across all devices? Or because the fonts would be wrong in some way? Well, the answer isn't because this would be too easy there really is a good reason for why you don't want to set viewport width to a fixed size. Namely that then your site would scale the same layout across all devices, you'd get the same layout of text and images on a four inch cell phone as you do on a ten inch tablet. And of course this means the fonts would be are consistent size with that layout. You'd get the same size fonts rescaled on to a four inch phone as you would on a ten inch tablet. And fundamentally this would result in a poor user experience

across mobile devices. Well, the answer isn't because this would be too easy there really is a good reason for why you don't want to set viewport width to a fixed size. Namely that then your site would scale the same layout across all devices, you'd get the same layout of text and images on a four inch cell phone as you do on a ten inch tablet. And of course this means the fonts would be are consistent size with that layout. You'd get the same size fonts rescaled on to a four inch phone as you would on a ten inch tablet. And fundamentally this would result in a poor user experience across mobile devices.

Set Viewport Width

Using device width makes the browser report the usable screen width, allowing the following fluid CSS changes to take effect correctly.

So let's go fix my page. Now the first thing we need to do in order to make a mobile friendly page, is to set the viewport. The mobile browser needs to know that this webpage knows how to format itself on devices with varying width. It doesn't need the mobile browser to pretend that it's a desktop screen with a given width. Once you set the viewport correctly, it will be able to resize properly and lay out properly. So what should my viewport meta say? Set Viewport Width That's right, you need to set the width to device width, and optionally, you can set the initial scale to one if you want, to help avoid some iOS device formatting issues.

Making it Fluid

The main changes are to keep containers flexible, remove unnecessary fixed widths, and allow images to scale within their layout.

The conversion begins by locating hard-coded widths that control the main wrapper, columns, and images. Replacing those constraints with percentages, automatic width, or a suitable maximum width lets the existing structure respond without discarding its content.

So now that you know how to support the view port meta, lets poke around in the developer tools to see what we need to do to fix this page. Now, the biggest problem here is that the page has a fixed column width. If I resize the window, you'll notice the column of text doesn't actually change size, just margin position. As we mouse around the elements inside the dev tools, we can pretty quickly find, the first offending element. This page element here actually has a width set on it. Let's disable that and see what happens. Well, it certainly changed things, but I don't think that it actually improved things very much. Let's keep digging inside the content and see what we can find. You will find a wrapper element

having a width of 100% already. Which is good. It means that it's not preventing us from resizing. But this content element does have a width set in pixels. Let's disable that. Now as we resize the page our content is fine but the sidebar is appearing and disappearing. Sure enough, it has a width set. If we set its width as a percentage also, now we're resizing a little bit better. So now let's go back up to the page, and let's try giving it a width but let's give it a width in view port units instead, now our only remaining problem seems to be that the header is actually not resizing as we want it to let's go take a look at the header image again. The image has a max width set, but not just a width. So, let's set its width. And now, everything seems to be resizing well. Now, this isn't perfect, but all that I really wanted to get at was that as you try to transition pages from fixed layout to fluid layout, The core things to look for, are fixed pixel sizes and lack of percentage resize.

Use Fluid Layouts

Percentages, automatic sizing, and flexible media form the base of a layout that works across phones, tablets, and desktops.

So there's really two rules for building fluid layouts. Use percentages or viewport units, something that'll size as you resize the window. And then, test while you're resizing the window. Let's take a look at how this worked out on the mobile device. We went from this layout without any changes to my blog. To this layout, definitely an improvement. I don't have to scroll left or right to see everything on the screen. And, you may notice the font size is actually a bit larger now. The font size was always set in M units, although it's actually still a bit small for a mobile device. So, we should probably bump that up. Just take a look on the tablet. In the tablet layout, you will get a definite improvement. This is actually a pretty readable experience. And, even when you rotate the screen, you end up with a nice reformatting. All this is really just a long winded way of saying, be sure to create fluid layouts. Be sure to reflow to use all the space on the screen and take advantage of every bit you can. And be sure you adapt to different screens as well. One tool that makes reflow a lot easier, particularly across very different screen sizes, is the new flex box layout in CSS. This new tool lets you stack elements in flexible rows or columns.

FlexBox Intro

Flexbox is designed for arranging items along an axis while distributing available space and adapting item sizes more naturally than rigid positioning.

FlexBox has been kicking around for a couple of years now, but it's gone through some pretty major revisions during that time. It's just recently shipped in iOS 7, and it's been shipping in Android for a little while, but you should test your FlexBox layouts thoroughly, across browsers. This may be a little too bleeding edge for you, but, let's take a look. We're going to walk through an interactive demo, talking you through each feature in FlexBox, as you change the page, live, in the developer tools.

FlexBox Compatibility

Compatibility and vendor prefixes must be checked because support can differ across the mobile browsers targeted by the application.

Okay, that was a ton of detail. Really I recommend you, just sketch out your design, and then party on it with FlexBox. Flex Box is super powerful, and it's really super easy to use. Incidentally, I did mention, Flex Box is still making its way into all the browsers out there. So, there are some Flex Box polyfills out there, that can help you, like Flexy.

Fluid to Responsive Design

A fluid layout supplies continuous resizing; responsive design adds deliberate structural changes where continuous resizing is no longer enough.

So these tools give us the ability to build designs that are fluid. But as it turns out, that's not enough. As you may have noticed as you went through this lesson, the experience of re-flow from a very small window size to a very large window size is not really a great one for the user. Now that you know how to build fluid designs, you need to learn to adapt those designs to radically different scenarios, like a four inch phone screen, and a 30 inch desktop screen. Fluid to Responsive Design

Fixed, Fluid, Adaptive, and Responsive web design

These models differ in how they use fixed dimensions, relative widths, breakpoints, and media queries to respond to available space.

Model	How it behaves
Fixed	Uses a set width; resizing or changing devices does not change the basic layout.
Fluid	Uses percentage widths so columns remain relative and scale with the browser.
Adaptive	Uses media queries to target specific device sizes.

Model	How it behaves
Responsive	Uses a fluid grid plus media queries to control design and content as the browser or device scales.

Different devices has different design

The same content may need different arrangements on a phone, tablet, and monitor because the usable space and interaction context are different.

- we can't use reflow across all devices.

Why we can't use reflow across all devices?

Reflow can rearrange content, but one uninterrupted arrangement may still become awkward at extreme sizes; some layouts need a breakpoint-driven structural change.

Reflow across all devices

Reflow remains valuable as the flexible base, while responsive rules refine the design when the content or controls need a different arrangement.

- New form factors come out all the time Reflow across all devices

Changing Layouts

A responsive layout can change the placement and proportion of components instead of only reducing their width.

Mobile Tablet Monitor

Responsive Web Design

Responsive design combines a fluid base with conditional layout changes so the page remains usable across different form factors.

- Responsive web design is the practice of building a website suitable to work on every device and every screen size, no matter how large or small, mobile or desktop.
- Responsive web design makes your web page look good on all devices.
- Responsive web design uses only HTML and CSS.
- Responsive web design is not a program or a JavaScript. Responsive Web Design

Why Responsive Web Design?

A fluid page can resize, but resizing alone does not guarantee a good experience. Responsive design adds deliberate layout changes, flexible media, and conditional CSS so that content remains clear and controls remain usable at different sizes and resolutions.

The topic first distinguishes fixed, fluid, adaptive, and responsive layouts. It then explains how media queries, Flexbox, device pixel ratio, vector graphics, raster-image quality, and responsive-image selection work together. Each technique solves a different part of the same problem: delivering an appropriate layout and visual resource before unnecessary scaling or downloading occurs.

Responsive Web Design

Responsive design combines a fluid base with conditional layout changes so the page remains usable across different form factors.

- Monitor Tablet Mobile

Different form factors



Three layout sizes



Responsive result

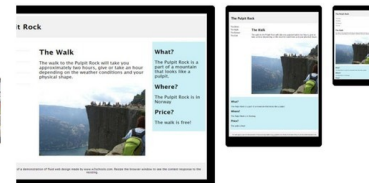


Figure 9. Responsive layouts across different form factors.

Media Queries

A media query applies CSS only when its conditions match, allowing layout and presentation to change at selected widths or device characteristics.

Each media query combines a condition with a group of CSS rules. The base layout remains available to every device, and the conditional rules alter it only when the stated expression matches. Breakpoints should therefore be chosen where the content needs a different arrangement.

- Media Queries is a W3C Candidate Recommendation—a widely reviewed document which is ready for implementation by browser vendors.

- It's an extension of media dependent stylesheets tailored for different media types (i.e. screen and print) found in CSS2.
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.
- A media query consists of a media type and an expression to check for certain conditions of a particular media feature. The most commonly used media feature is width.

Media Queries Examples

The examples show that a query connects a condition to a block of CSS; the styles apply only while that condition is true.

```

• Example 1: If the browser window is smaller than
500px, the background color will change to light blue:
• @media only screen and (max-width:500px)
{
body {
background-color: lightblue;
}
}
Media
Media Q
Queries Examples
ueries Examples

```

Media Queries Examples (Example 1)

The first example demonstrates the basic relationship between a condition and the rules protected by that condition. When the viewport satisfies the expression, the browser applies the enclosed declarations; when it no longer satisfies the expression, the base declarations continue to control the layout. This makes the change reversible as the available width changes.

Media Queries Examples (Example 1)

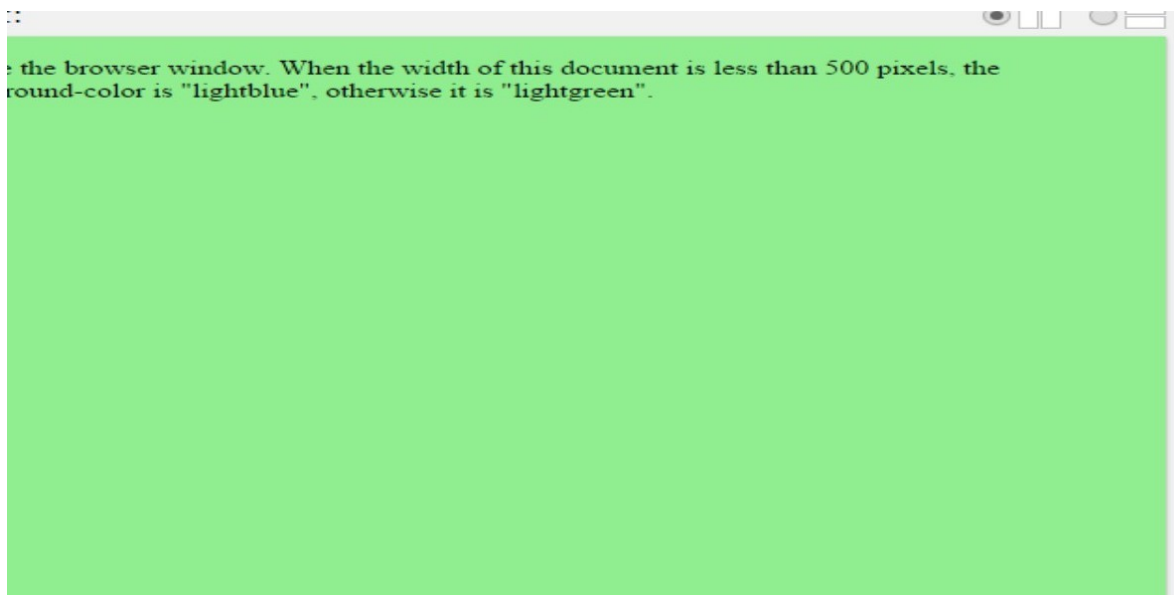


Figure 10. Media query output.

Media Queries Examples 2

A second example extends the same principle to another layout decision. Multiple queries can describe different useful states, but they should not create arbitrary device categories. Each condition should correspond to a point where the content, navigation, or columns need a more suitable arrangement.

```

• Example 2: The following example shows a menu that
will float to the left of the page if the viewport is 480
pixels wide or wider (if the viewport is less than 480
pixels, the menu will be on top of the content):
• @media screen and (min-width: 480px) {
#leftsidebar {width: 200px; float: left;}
#main {margin-left: 216px;}
}
Media
Media Q
Queries Examples 2
Queries Examples 2

```



Figure 11. Responsive menu output.

Available Media Query Expression

Media-query expressions can test characteristics such as width, height, orientation, resolution, and related minimum or maximum values. The chosen feature should match the design decision being made. For example, a width condition is appropriate when a row of content no longer fits, while an orientation condition is useful when the arrangement depends on portrait or landscape space.

Width Orientation Color Height Aspect-ratio Color-index Device-width Device-aspect-ratio Monochrome Device-height Grid Resolution

Flexbox

Flexbox controls the direction, alignment, wrapping, and order of flexible items inside a container.

The flex container establishes the layout direction, while its items can be aligned, wrapped, reordered, and allowed to consume available space. These properties are especially useful when a row of controls or columns must adapt without relying on fixed positioning.

- Flexible boxes, or flexbox, is a new layout mode in CSS3.
- Use of flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and different display devices.
- For many applications, the flexible box model provides an improvement over the block model in that it does not use floats, nor do the flex container's margins collapse with the margins of its contents.

Flexbox (Example)

The example begins by declaring a flex container and placing several flex items inside it. Once the parent establishes the flex formatting context, the children participate in that layout automatically. Their size, direction, spacing, and alignment can then be controlled from the container instead of positioning every item independently.

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: -webkit-flex;
  display: flex;
  width: 400px;
  height: 250px;
  background-color: lightgrey;
}
.flex-item {
  background-color: cornflowerblue;
  width: 100px;
  height: 100px;
  margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
</html>
```

Flexbox (Example continue)

The continuing code adds the dimensions and presentation needed to make the behavior visible. The important structural idea is that the container controls the flexible relationship between the items. This makes the example easier to adapt when the available width changes or when another item is added.

Flexbox (Example out put)

The output shows the items arranged inside the flex container according to the declared direction and spacing. It should be read together with the code: the visual result is produced by the parent-child flex relationship, not by fixed coordinates assigned separately to each box.



Figure 12. Flexbox output.

Flexbox Properties

Container properties establish direction, wrapping, and alignment, while item properties influence order and the way individual items use space. A responsive component normally begins with the simplest useful values and adds wrapping or alternative alignment only when the content requires them.

Property	Description
display	Specifies the type of box used for an HTML element.
flex-direction	Specifies the direction of flexible items inside a flex container.
justify-content	Horizontally aligns items when they do not use all available main-axis space.
align-items	Vertically aligns items when they do not use all available cross-axis space.
flex-wrap	Controls whether items wrap when one flex line has insufficient room.
align-content	Aligns flex lines and modifies the behavior of flex-wrap.
flex-flow	Shorthand for flex-direction and flex-wrap.
order	Specifies the order of a flexible item relative to other items in the container.

The Device Pixel Ratio

Device pixel ratio describes how physical screen pixels correspond to logical CSS pixels.

A CSS pixel is a layout unit rather than a guaranteed single hardware pixel. At a ratio of two, one CSS pixel is represented by two physical pixels in each dimension. The additional hardware pixels can improve sharpness when an appropriately detailed resource is available.

The device pixel ratio is the ratio between physical and logical pixels The Device Pixel Ratio

What is Pixel

A pixel is the smallest addressable picture element in a digital display or image, but web layout does not always map one CSS pixel to one hardware pixel. The distinction becomes important on portable devices because screens of similar physical size can contain very different numbers of physical pixels.

A pixel is simply the individual point of color on a digital image. A pixel doesn't have a particular size. It is an abstract representation of a specific coordinate, like a point on a map.

- Pixel is used to describe the number of discrete points that can be captured by digital cameras, and because most cameras can actually recognize millions of points, the term Megapixel is used to shorten the number of zeros needed.

Difference Between Physical and Logical Pixel

Physical pixels belong to the display hardware, while logical pixels provide the coordinate system used by the page layout.

- Physical pixel : The actual number of pixels that mobile or computer device support e.g. physical pixel size of iphone6 plus is 1080*1920 pixels.
- logical pixel: Device independent pixels are called logical pixel e.g. 1 Logical pixel may be equal to 2 or more physical pixels in digital image.

The Device Pixel Ratio

Device pixel ratio describes how physical screen pixels correspond to logical CSS pixels.

A CSS pixel is a layout unit rather than a guaranteed single hardware pixel. At a ratio of two, one CSS pixel is represented by two physical pixels in each dimension. The additional hardware pixels can improve sharpness when an appropriately detailed resource is available.

The device pixel ratio is the ratio between physical pixels and logical pixels. For instance, the iPhone 4 and iPhone 4S report a device pixel ratio of 2, because the physical linear resolution is double the logical resolution. Physical resolution: 960 x 640 Logical resolution: 480 x 320

Other devices report different device pixel ratios, including non-integer ones. For example, the Nokia Lumia 1020 reports 1.6667, the Samsung Galaxy S4 reports 3, and the Apple iPhone 6 Plus reports 2.46 (source: dpilove). But this does not change anything in principle, as you should never design for any one specific device.

- 800 pixels 1080 pixels Nexus 7 Tablet GalaxyS4



Nexus 7 Tablet



GalaxyS4

Figure 13. Physical resolution comparison.



Figure 14. Device pixel density comparison.

Mobile Device Pixels

Mobile screens differ in both total resolution and the number of pixels placed within each inch. A dense screen can draw finer detail, yet the page must continue to use usable logical sizes for text and controls. Device pixel ratio provides the connection between those layout sizes and the physical display.

- Mobile devices not only have different numbers of actual pixels, but they fit very different numbers of pixels into each inch of the screen real estate

The Device Pixel Ratio

Device pixel ratio describes how physical screen pixels correspond to logical CSS pixels.

A CSS pixel is a layout unit rather than a guaranteed single hardware pixel. At a ratio of two, one CSS pixel is represented by two physical pixels in each dimension. The additional hardware pixels can improve sharpness when an appropriately detailed resource is available.

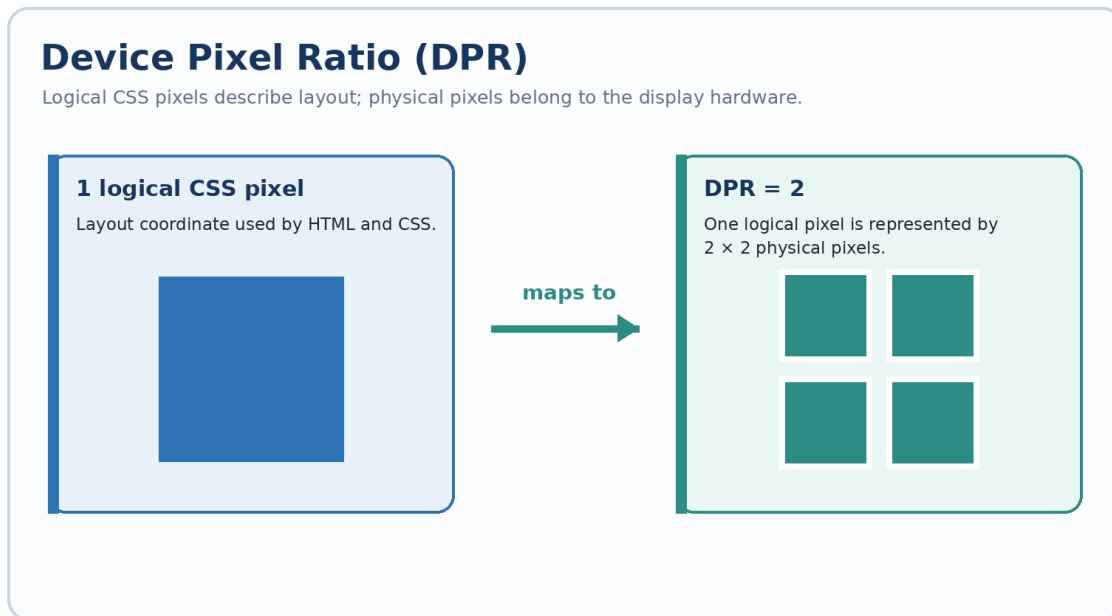


Figure 15. Logical CSS pixels and physical pixels at DPR 2.

Image Quality

Good image delivery balances visual quality across screen densities with the bandwidth cost of downloading larger files.

Sending the largest possible file to every screen is wasteful, while sending a low-resolution file to a dense screen may produce visible softness. A useful strategy considers the displayed size, pixel density, compression, and transfer cost together.

- Web applications need to deliver good across a broad range of devices, but they need to do so at a minimum cost.

Removing the Default Zoom

Default zoom is related to the browser's assumed viewport and the scale needed to fit a desktop-oriented page. Supplying an appropriate device-width viewport and a suitable initial scale gives the page a predictable starting size. The layout must still be flexible; removing zoom does not repair content that remains wider than the screen.

- If you have used to browsing the Web on a smartphone you'll notice how websites are scaled out to fully display within the screen.

- When you get into building a responsive mobile design, the auto-zoom can really mess up your layout elements. Specifically, images and navigation content may appear small or too large in your layout.
- There is a special meta tag you can append into the document header which resets this in most Android and iPhone devices.

```
<meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0; user-scalable=0;" />
```

This is known as the viewport meta tag which sets up some custom variables within the content. Apple has a documentation page regarding a few other meta tags you should look into, although these are geared specifically toward websites on iOS. The initial-scale value is important as this defaults your website to a full 100% zoom.

Difference in image Quality

The same raster file can appear different when displayed across screens with different densities or when scaled beyond the detail it contains. A low-detail source may look soft on a dense screen, while a much larger source can look clear but cost more to transfer. Quality must therefore be evaluated together with the displayed dimensions.

Images are important facet of practically every website. Mobile users may not be looking to stream videos, but photos are a whole different story. These are also the biggest problem when it comes to layouts breaking out of the box model. `img { max-width: 100%; }`

The standard rule for CSS is to apply a max-width property to all images. Since they'll always be set at 100% you will never notice distortions. When the user re-sizes their browser window smaller than your image. it'll automatically re-adjust to 100% width scaled down. The problem is that Internet Explorer cannot understand this property, so you'll need to put together an IE-specific stylesheet using `width: 100%;` When we render images to the screen. A pixel layout dimensions may well not equal to pixel in real device terms.



Figure 16. Image rendering at different densities.

Beautiful images, low bandwidth

The objective is to retain the visual information that the design needs while avoiding bytes that the device will not use. Compression, image dimensions, format, and the selected density copy all contribute to that balance. A resource is efficient when it is appropriate for its rendered role, not simply when it is the smallest available file.

There should be our goal to make image as beautiful as they can using as little bandwidth as possible.

- Chromebook pixel screen: 4.1 mPx
- Nexus 4 screen : 0.9 mPx
- Iphone 5 screen :0.7 mPx

Producing Different Quality Content

Vector and programmatically drawn graphics can remain clear at different densities, while raster content usually needs carefully selected resolution and compression.

Different Devices have different display densities. Some of them can make use of pixels and some of them can't . Producing Different Producing Different Q Quality Content uality Content

- What is SVG? SVG stands for Scalable Vector Graphics. SVG is used to define graphics for the Web.
- The HTML <svg> Element The HTML <svg> element (introduced in HTML5) is a container for SVG graphics. SVG has several methods for drawing paths, boxes, circles, text, and graphic images.

```
<!DOCTYPE html>
<html>
<body>
<svg width="100" height="100">
<circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />
</svg>
</body>
</html>
```

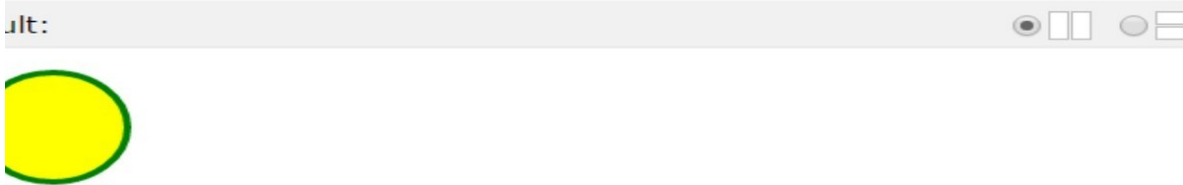


Figure 17. SVG output.

What is HTML Canvas? The HTML <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript). The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics. Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #000000;">
Your browser does not support the HTML5 canvas tag.
</canvas>
</body>
</html>
```



Figure 18. Canvas output.

Comparison of SVG and Canvas

SVG describes graphics as document elements, which supports scaling and element-level interaction. Canvas provides a drawing surface whose pixels are produced by script and is well suited to repeated, graphics-intensive drawing. The comparison helps choose a method according to resolution needs, interaction, rendering area, and the kind of graphic being produced.

Canvas	SVG
Resolution dependent	Resolution independent
No support for event handlers	Supports event handlers
Poor text-rendering capabilities	Suited to applications with large rendering areas
Result can be saved as .png or .jpg	Complex DOM-based rendering can be slow
Suited to graphic-intensive games	Not suited to game applications

Large area, low quality images

A large display area makes insufficient image detail easier to notice, but increasing the source dimensions also increases the possible download cost. The appropriate response is not automatically to use the maximum resolution. The image should contain enough detail for its expected rendered size and be compressed to a practical level.

Let's presume, we really need a raster image because some images can't be represented well by vector drawing.

Large area, low quality images

For a large visual area, compression and resolution must be balanced so that the image remains useful without consuming unnecessary bandwidth.



Figure 19. Raster image used in the quality discussion.

Large area, low quality images

A large display area makes insufficient image detail easier to notice, but increasing the source dimensions also increases the possible download cost. The appropriate response is not automatically to use the maximum resolution. The image should contain enough detail for its expected rendered size and be compressed to a practical level.

Deliver high resolution but highly compressed images

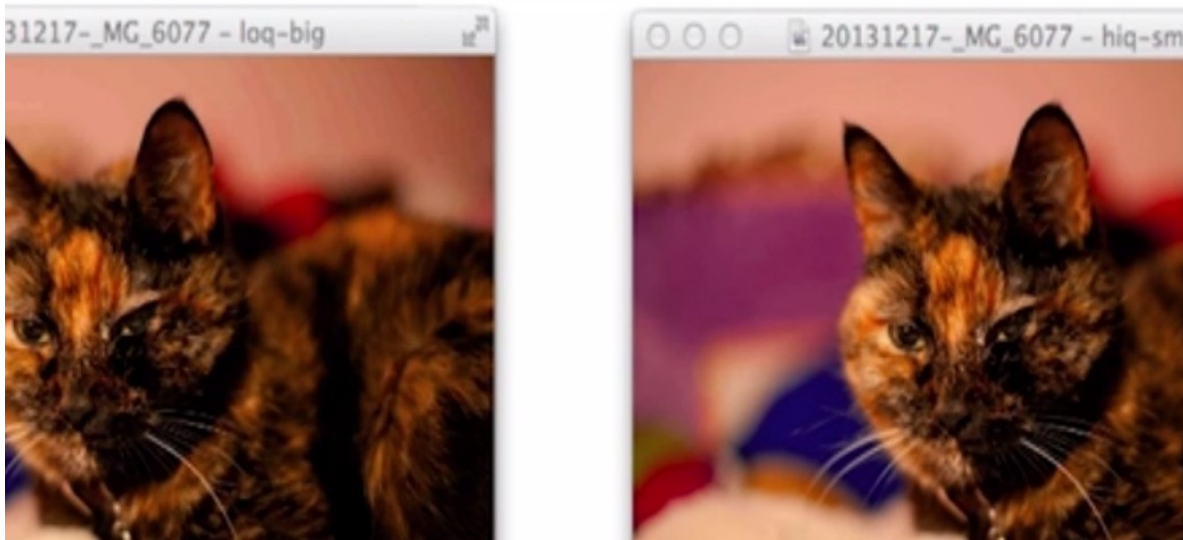


Figure 20. Resolution and compression.

Do the right selection of quality image

Selection should take place before the request whenever possible. The browser or server can use available information about density and viewport size to choose among prepared copies. This avoids downloading one file only to replace it with another after the page has already spent time and bandwidth.

- Using a higher resolution image to every device is not proper solution.
- q Should we use a low quality source image ,replace it with JavaScript?
- q Should we expect the server to give us the proper pixel density image magically?
- q Should we make several copies of the image and conditionally request the right one from the client?
- q Should we use a low quality source image ,replace it with JavaScript?
- q Should we expect the server to give us the proper pixel density image magically?

HTTP Client Hints

Client hints allow the request to carry device and viewport information that can help a server choose a more appropriate image response.

This specification defines a set of HTTP request header fields, known as Client Hints, that are intended to be used as input to proactive content negotiation; just as the accept header allows clients to indicate what formats they prefer, Client Hints allow clients to indicate a list of device and agent specific preferences.

Available hints Current list includes DPR (device pixel ratio), Width (resource width), Viewport-Width (layout viewport width), and Dnlink(maximum downlink speed) request headers, and Content-DPR response header that is used to confirm the DPR of selected image resources.

- Delivering DPR-aware images DPR hint automates device-pixel-ratio-based selection and enables delivery of optimal image variant without any changes in markup. For example, given the following HTML markup:

```
  
GET /img.jpg HTTP/1.1  
User-Agent: Awesome Browser Accept: image/webp, image/jpg  
DPR: 2.0  
CH-RW: 160  
HTTP/1.1 200 OK  
Server: Awesome Server  
Content-Type: image/jpg  
Content-Length: 124523  
Vary: CHoDPR, CH-RW  
DPR: 2.0
```

Do the right selection of quality image

Selection should take place before the request whenever possible. The browser or server can use available information about density and viewport size to choose among prepared copies. This avoids downloading one file only to replace it with another after the page has already spent time and bandwidth.

- q Should we make several copies of the image and conditionally request the right one from the client?

Responsive Images

Responsive image techniques allow the browser or layout to use image sources suited to display density, rendered size, and other conditions.

The browser should receive enough information to choose a suitable source before downloading. If selection happens only after a large default image has already arrived, the layout may look correct but the bandwidth benefit has been lost.

Responsive images is a method for providing the browser with multiple image sources depending on display density, size of the image element in the page, or any number of other factors to make possible flexible approach to images. Graphics are often a problem in responsive layouts designed to adapt automatically to different device sizes from desktop

monitors to hand-held smartphones. Traditionally, graphics have fixed dimensions that break responsive layouts on screens that are smaller than the graphic. Length is not a problem, because pages can always scroll vertically. However, you never want the page to have to scroll horizontally on mobile devices with fluid layouts.

Problems in Responsive Images

A complete responsive-image solution must consider semantics, art direction, validity, bandwidth, server support, and JavaScript dependence.

The image must retain its meaning, allow a different crop when art direction requires it, use valid markup, and avoid unnecessary downloads. A practical solution also has to work with server behavior and should not depend entirely on JavaScript.

What Are The Problems? One major factor in the need for responsive images is overall website size - even today, a huge percentage of mobile-version sites are as large (or even larger) than their desktop versions, thereby affecting performance negatively

Problem 1 - Semantics Making responsive images work cleanly and reliably across multiple platforms sometimes involves using techniques that aren't 'semantic'. Why? Because when the source of an image points to a real image, with an alt text to describe it, even if the source is the smallest image possible, it means you're downloading unnecessary data.

Problem 2 - Art Direction A picture tells a thousand words ... most of the time. Many responsive image techniques enable you to provide several resolution versions of an image, which can then be used accordingly for suit a given platform. However, this can sometimes have a negative effect where the message of the image is diluted or lost altogether. What's the minimum size for an image? The answer, of course, depends on context.

Problem 3 - Validity If you are customizing markup, checking markup validity using a validation service such as W3C ensures that a construct follows the correct syntax of whatever language it works with (for example, HTML). While most of the time browsers are able to think ahead to present Web pages that aren't strictly 'valid' without any problems, these results can sometimes vary between one browser and another. In rare cases, it can cause confused layouts or even crashes. 'Validating' every page is a bulletproof way of identifying these problems before they occur.

Problem 4 - Bandwidth Capabilities Much as we'd all like to say otherwise, a world of super- speed connections and lightning-fast load times for everyone is still some way off. That's why in many developers' opinions,

responsive images ought to consider the bandwidth available on any given device, and adjust the image size accordingly to cut out excess downloads.

Problem 5 - Server Side Components The bulk of Adaptive Images' work is done through .htaccess and PHP 5.x, which offers a great solution without complete dependency on JavaScript. The problem with .htaccess is that it assumes your website is running on an Apache server. If you're using something else, such as Nginx, you can port the .htaccess over to Nginx syntax - which is similar to Apache's but this still involves some work. Additionally, adding PHP files to your site's root directory may not be possible if you're running on other technologies, such as Ruby or Python. Clearly this isn't ideal for everyone, so be sure to check first before you dive in.

Problem 6 - JavaScript As you've probably noticed, many of the responsive image techniques above require some JavaScript. If you'd prefer to keep things as simple as possible and want to avoid using JavaScript you can use the third-party Sencha.io solution.

Responsive images

A responsive image must remain flexible in the layout and must also use a source with appropriate detail. CSS can prevent overflow and control presentation, while source-selection techniques address which file is transferred. Treating these as separate concerns makes it easier to diagnose whether a problem comes from layout or resource choice.

Responsive images is a method for providing the browser with multiple image sources depending on display density, size of the image element in the page, or any number of other factors to make possible flexible approach to images.

Responsive Images using CSS3

CSS can make an image scale with its container through relative width and a limiting maximum width. Background images can also be changed by media conditions or supplied as density variants. These methods are particularly suitable when the image is part of presentation rather than essential document content.

Using The width Property: If the width property is set to 100%, the image will be responsive and scale up and down: `img { width: 100%; height: auto;`

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
img {
width: 100%;
height: auto;
}
</style>
</head>
<body>

<p>Resize the browser window to see how the image will scale.</p>
</body>
</html>
```

- Using The max-width Property If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size: `img { max-width: 100%; height: auto;`

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
img {
max-width: 100%;
height: auto;
}
</style>
</head>
<body>

<p>Resize the browser window to see how the image wil scale when the width is less
than 460px.</p>
</body>
</html>
```

width: 100%**max-width: 100%**

user window to see how the image wil scale when the width is less than 460px.

Figure 21. Comparison of width and max-width responsive image behavior.

Use of high quality and different sized copies of image

Preparing several copies creates useful candidates for different screens, but the copies provide a benefit only when the correct one is selected early. The candidate set should represent meaningful differences in rendered size or density instead of offering several files that are nearly identical in cost and quality.

In order to do responsive images, we need to request different images based on the device pixel ratio, but also possibly based on the layout width. These images may be the same image transcoded to different sizes. They may be actually different images depending on the layout size. Use of high quality and different sized copies of image Use of high quality and different sized copies of image We shall need to provide multiple copies of this image so that low pixel count devices can save bandwidth or high pixel count devices can get a crisp image. It is possible to manage on server side in some cases. In fact, some content systems let you simply specify what size you want a given source image app. Use of high quality and different sized copies of image If you want the images to actually represent different art, different crops for example you'll have to do that manually. For each of these images though you should remember to tune the quality parameters to minimize file size while maximizing quality. We'll need at least 1x and 2x versions of images as 1x or 2x for device pixel ratio purpose. For responsive images, We may request different images based on available layout width. Use of high quality and different sized copies of image

Request Right Image

The objective is for the client to request an image that matches its pixel ratio, layout size, and practical bandwidth needs.

- Client should request for the "right" image.
- We need to get the client to ask for the right image to be downloaded.
- It needs to know what device pixel ratio the device is using and potentially the layout size too, if it's responsive to the layout width and then request only the appropriate image.

This last part is the hard part, because the web browser tries to download image as early as possible which of course is a good thing for getting content on the screen quickly, but bad if the client asks for more than one version particularly on a mobile device with limited bandwidth. So we should try to find the solution for the problem of right image at the right time.

Ideal Image Solution

An ideal solution selects the right file before download by considering display density, rendered size, and practical bandwidth.

The selection process first observes the viewport, rendered image size, and device density. It then compares the available candidates and fetches only the source that offers a practical match. The important saving occurs because the unsuitable copies are never requested.

- What we really need is the ability for our client to selectively choose what image to use.
- Ideal solution would be semantic and validate properly. It would only load one copy of the image and work and be accessible across all browsers.
- Unfortunately that solution does not exist.

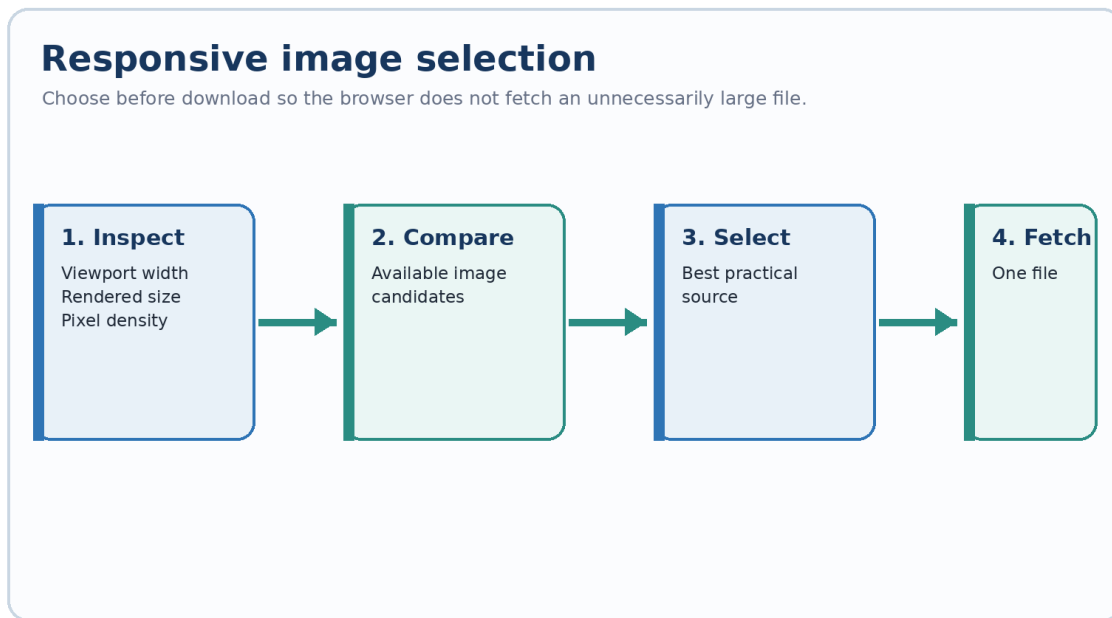


Figure 22. Responsive image selection before download.

Background Image

Background-image rules separate decorative presentation from document content and provide control over source, repetition, and position.

The CSS background properties are used to define the background effects for elements. We shall discuss following properties of background images:

- Background Image - Repeat Horizontally or Vertically

- Background Image - Set position and no-repeat
- Background Image -background-position: Background Image
- Background Image: The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element.

```
Example:
<!DOCTYPE html>
<html>
<head>
<style>
body {
background-image: url ("paper.gif");
}
</style>
</head>
<body>
<h1>Hello World!</h1>
<p>This page has an image as the background!</p>
</body>
</html>
```

Background Image - Repeat Horizontally or Vertically By default, the background-image property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange, like this:

```
• Background Image - Repeat Horizontally or Vertically:
<!DOCTYPE html>
<html>
<head>
<style>
body {
background-image: url("gradient_bg.png");
}
</style>
</head>
<body>
<h1>Hello World!</h1>
<p>Strange background image...</p>
</body>
</html>
```

- Background Image - Set position and no-repeat:

```

Example:
<!DOCTYPE html>
<html>
<head>
<style>
body
{
background-image: url("img_tree.png");
background-repeat: no-repeat;
}
</style>
</head>
<body>
<h1>Hello World!</h1>
<p>Background image example.</p>
<p>The background image is only showing once, but it is disturbing the reader!</p>
</body>
</html>

```

background-position: In the example above, the background image is shown in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much. The position of the image is specified by the **background-position** property: Example

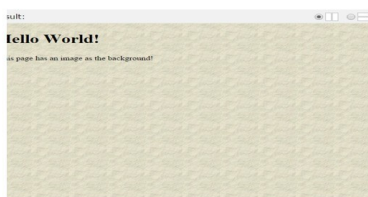
```

body {
background-image: url("img_tree.png");

```

background-repeat: no-repeat; background-position: right top;

Background image



Repeat



Position + no-repeat



Figure 23. Background image, repeat, and positioned no-repeat behavior.

Image-set

image-set provides density variants of a CSS image and lets the browser select the appropriate source.

The function associates several CSS image sources with density descriptors. A supporting browser can select the source that corresponds to the display environment, allowing a decorative image to remain clear without requiring every device to download the densest copy.

Why we use image-set instead of media queries? I. The problem with media queries for higher resolution displays is they don't give the browser any discretion. They say explicitly that if the pixel density equals 1 or 2 or whatever, that the browser must use a particular image source. II. The various resolutions of the image are all in the same place in css. When we use media queries to specify different image resolutions, we can end up with the image sources separated by dozens of lines of css which makes tracking down the different image sources more challenging.

- The emerging trend on HD screens for computer devices has changed the way we build websites, including how we add images. In the past, we can simply use background-image and set the image URL.

Today, this is no longer relevant, since we also need to provide an image that is optimized for higher resolution displays or the image will look pixelated or blurred. So, technically we need two sets of image, one in regular size and another twice the size.

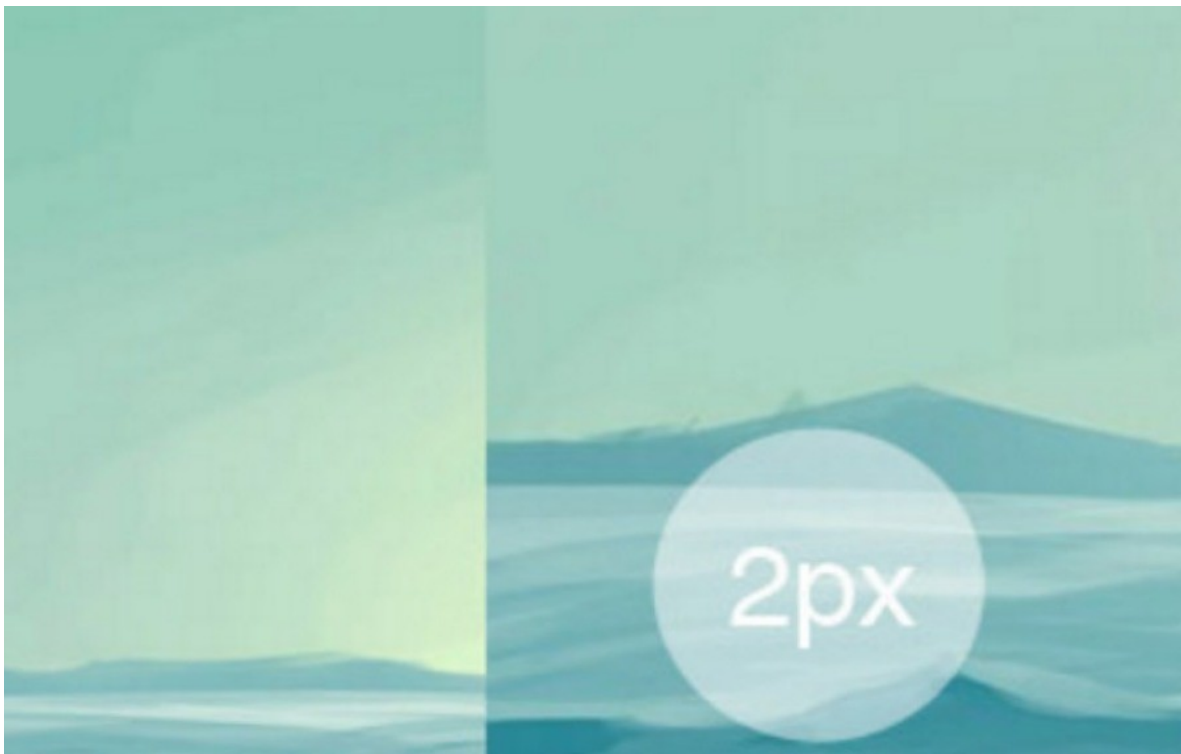


Figure 24. Density selection with image-set.

CSS Image Set Function: By using this function, we can insert multiple images which will be set for normal and high-res display. Apart from that, this function is also trying to deliver the most appropriate image resolution based on the connection speed. So, regardless of the screen resolution, if the user accesses the image through a slow Internet connection.

CSS Image Set is experimental. It is only supported in Safari 6 and Google Chrome 21 and is prefixed — `-webkit-image-set()`. In action, the smaller-sized image will be delivered.

- This function is declared within the `background-image` property, while the background URL is added within the function followed by the resolution parameter (1x for normal display and 2x is for high-res display), like so.

```
.selector {
background-image: -webkit-image-set(url('../img/image-
1x.jpg') 1x, url('../img/image-2x.jpg') 2x);
}

body {
background-image: url('../img/bg.png');
}

.demo-wrapper {
width: 600px;
margin: 0 auto;
}

.demo-wrapper .theimage {
margin-top: 100px;
border: solid 8px #f3f3f3;
width: 100%;
height: 360px;
background-size: 100% auto;
background-repeat: no-repeat;
background-position: center center;
background-image: url('../img/image-1x.jpg');
background-image: -webkit-image-set(url('../img/image-1x.jpg')
1x, url('../img/image-2x.jpg') 2x);
}
```



Figure 25. Image-set output.

Finally: `image-set()` function seems to be a promising solution for serving images in high-res. But, with the current limited browser support, `image-set()` is not ready to be implemented in a live site. In addition, the implementation could be changed along the way in the near future.

Importance of Performance

Performance directly affects whether users stay engaged with a page. Even a small delay can reduce interaction and income, so speed must be treated as a design requirement from the beginning rather than as a final adjustment.

On a portable device, a slow response is especially noticeable because network quality, processing power, and battery use can vary. Performance influences both the user experience and the business result of a page. For that reason, the cost of layout choices, scripts, and resources should be considered while the page is being designed and developed.

Why performance

Small delays can reduce engagement and income, so performance must be treated as a core requirement.

Users experience performance as waiting: waiting for content to appear, waiting for an interaction to respond, or waiting for the page to become stable. Even when the final design is visually correct, excessive waiting can prevent the user from reaching it.

- A slow site means that you will have less user engagement.

Example

Let's say you accidentally introduce a performance issue, that causes your page to take an extra half second to load. That's just 500 milliseconds. That's a pretty small difference. What kind of difference, in user engagement, do you think this would cause? Even such a small change, causes a slightly more than one percent drop in user engagement, and income from your site.

It is a requirement

Performance should be stated as a requirement because it affects design choices before implementation is complete. A requirement can guide the size and number of resources, the amount of computation, and the complexity of rendering. Without that expectation, the page may accumulate costs that are difficult to remove at the end.

Many well publicized studies from Google, Microsoft, Amazon, Facebook, and others, all show that UI performance translates directly to profit. A drop in performance directly drives a drop in user engagement. In short, speed is not a feature, it's a requirement. Users won't use your app if it's not fast.

Analyzing Performance

Performance work begins with measurement. The page should be observed under repeatable conditions so that network transfer, rendering work, and computation can be examined separately and the real bottleneck can be identified.

A useful investigation separates what the browser is doing into three broad areas. Network analysis examines requests and transfer time, rendering analysis examines the work required to display the page, and computation analysis examines JavaScript and other processing. Looking at these areas separately prevents an attractive but ineffective optimization from hiding the actual cause of delay.

When to consider Performance

Performance should be considered during design and development, because late optimization cannot always repair expensive structural decisions.

Of course, it's nearly impossible to graph performance into your application after it's already been built, but before we get into good performance patterns, the first step to insuring great performance, is being able to analyze your performance

#perfmatters

Chrome Development Tools

The development tools provide observable evidence about requests and page work, making them the starting point for a performance investigation.

Understanding Network

Pillars of Performance

The course groups performance work into network, rendering, and computation.

The three pillars provide a diagnostic order. First observe the transfers, then the browser work required to display the result, and finally the processing performed by scripts or other computation. Measurements can then be compared after a change to confirm that it produced a real improvement.

- Three pillars of performance are
- Network
- Render
- Compute

Network

Network analysis examines what is requested, how large each transfer is, and how long the browser waits before useful content becomes available.

- Examples
- 1. www.vu.edu.pk
- 2. <http://playbiolab.com/>
- 3. <https://www.eecs.mit.edu/>

Touch Interaction

Touch is the primary pointing method on most portable devices. A touch interface must respond naturally to fingers while remaining compatible with mouse input and normal browser behavior.

A finger is less precise than a mouse pointer and does not provide a permanent hover state. Controls must therefore be easy to see and large enough to tap, and the interface must not hide essential actions behind hover-only behavior. At the same time, ordinary browser actions such as scrolling should remain available unless the application has a clear reason to replace them.

One of the most important factors in an immersive user experience on a mobile device is touch interaction. You expect a responsive interaction from a device that you hold in the palm of your hand, and with most mobile screens, your primary pointing device is your

finger. As a mobile web developer, you need to be sure that you're building great touch enabled user devices

Simple rules for touch UI

A reliable touch interface follows a few practical rules: do not depend on hover, make targets large enough to tap, support both touch and mouse, and avoid replacing browser behavior unless the application genuinely needs a custom gesture.

These rules are practical design constraints rather than optional decoration. A page that uses adequately sized controls, visible interaction states, and shared touch-and-mouse support is easier to operate and simpler to maintain. The rules also reduce the need for device-specific code because the same interface remains understandable across input methods.

Rule 1

Essential actions and information must not depend on hover. Touch screens do not provide the same continuous pointer position, so the user needs a visible control or a tap-based way to reach the same result.

Do not rely on hover effects Buttons Source:

http://www.w3schools.com/css/css3_buttons.asp



Figure 26. Hover-only behavior.

Rule 2

Interactive targets should be large enough and separated enough for a finger. This reduces accidental activation and allows the interface to be used without requiring the precision of a mouse cursor.

Size of Touch Area should be at-least 10mm X 10 mm

Rule 3

Touch support should not eliminate mouse support. Portable and hybrid devices can expose more than one input method, and the same page may also be used on a desktop computer.

Touch and mouse are not mutually exclusive



Figure 27. Touch and mouse together.

Rule 4

Default browser behavior should be preserved unless the custom interaction genuinely requires control. Scrolling and other familiar actions are part of how users navigate a mobile page, so cancelling them unnecessarily makes the interface feel broken.

Provide Mouse support as well. Don't do anything It is important to understand that you may not even need to do anything in order to support touch. Touch events emulate mouse clicks already, and the web browser builds in some gestures like scrolling and zooming. So,

you only need to implement something here in cases where the gesture support Or mouse emulation does not give an optimal user experience.

Example

- http://www.w3schools.com/css/css3_buttons.asp

Click Delay

Some mobile browsers delay a click briefly while deciding whether the user intends to double-tap. Understanding why the delay exists helps in choosing an appropriate remedy without damaging zooming, scrolling, or accessibility.

The delay was introduced so the browser could wait for a possible second tap used for zooming. A solution should remove the delay only when the page has a suitable viewport and the user can still read and navigate comfortably. Disabling zoom merely to make clicks feel faster can exchange a small timing issue for a serious usability problem.

Touch supporting mobile browsers, usually interpret double tapping, as a zoom into this element gesture. Unfortunately, this means the platform, delays firing click of it until it can decide whether, the user is double tapping. This delay is about a third of a second. Chrome has recently checked in some changes, that minimize this behavior. But, pretty much, all mobile browsers, have this click delay feature, to some degree.

Fix

A fix for click delay must be evaluated in the context of the viewport and zoom behavior. The goal is a prompt activation without causing the same action to fire twice or preventing the user from enlarging content that needs to be read.

There are a few ways to fix this problem, and, get clicks without a delay,

- 1) You can set the view port, to be non- scalable. Either, set user scalable, to no, or set minimum scale and maximum scale, to one, which has the same effect. This can cause accessibility problems, though, so be careful jumping, to this solution. You need to make sure, if you do this, that your site will never ,need to be zoomed.
- 2) You can use a fast click library, like the ft labs one, that we've linked down below. But ,you do have to be careful, about how this impacts, your scrolling performance, be sure to read the directions, very carefully. <https://github.com/ftlabs/fastclick>

- 3) Where finally, of course, you can implement tech support yourself directly, if you consume the touch events, you won't have this issue.

Supporting Touch

Touch support is implemented through a sequence of touch events. The important point is not only which event fires, but also how the interaction remains associated with its original target from the beginning to the end of the gesture.

Touch support is based on a sequence rather than a single action. The page observes when contact begins, how it moves, when it ends, and whether it is cancelled. Because the events remain connected to the element that originally received the touch, the handler can continue a gesture even when the finger moves beyond that element.

Go to www.caniuse.com And search for "touch" Or go to <http://caniuse.com/#feat=touch>

Touch Events

Touch interaction is represented through touchstart, touchmove, touchend, and touchcancel events.

touchstart marks initial contact, touchmove reports changing position, touchend reports release, and touchcancel handles an interrupted interaction. Together they describe the complete lifetime of a touch rather than treating it as an isolated click.

- touchstart -> mousedown
- touchmove -> mousemove
- touchend -> mouseup
- touchcancel

Touch Events are Fired on The

Touch events continue to target the element where the gesture began, even if the finger later moves outside its boundaries.

Touch events are always delivered to the element that first received that Touch. They don't walk across boundaries like Mouse Events do. So it's important to hook the right elements. In fact, it's a good idea to add the touch end handler during your touch start handler. And, keep in mind too that even if you remove a DOM element from your tree, it still gets the

events until the touch ends. DOM http://www.w3schools.com/js/js_htmlDOM.asp Original Target

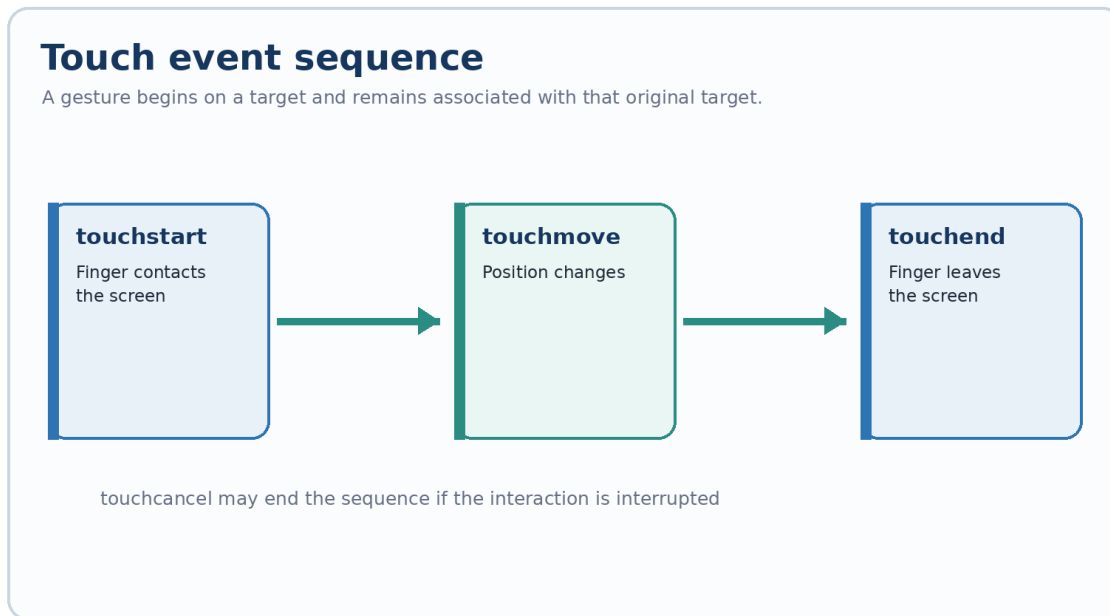


Figure 28. Touch event sequence and the original target.

Mouse Emulation

Browsers often translate touch interaction into mouse-like behavior, but maintaining two separate implementations can duplicate logic. The synthesizer example shows where that duplication becomes difficult during pressing and dragging.

The synthesizer example makes the maintenance problem visible. Pressing a key, moving across keys, and releasing the interaction must behave consistently whether the input began as touch or mouse input. Repeating the same application logic in separate handlers increases the chance that the two versions will behave differently.

Example: <https://webaudiodemos.appspot.com/midi-synth/>



Figure 29. Synthesizer application.

Synthesizer Application

Application Link

<https://webaudiodemos.appspot.com/> <https://github.com/cwilso/midi-synth> We will initially start with touch events. There were two challenges, though. First, you'll notice that on the desktop, if I don't have that touch event emulation turned on, it doesn't actually work at all when I use the mouse. Of course, this is expected. But I didn't really want two code paths, one for mouse and one for touch. Secondly, I really wanted to be able to have a drag across the keyboard. Play each note as you slid across it in turn. But with the way touch delivers all the events to the originally touched down element, I would have had to calculate the hit testing myself for these. And that was kind of a pain. I really wanted these touch events to be delivered to the down element for each key. Instead what happens in this synth, is when I hit one key and then drag, it doesn't actually move where the events are delivered.

Getting Started With Touch Events

A basic touch implementation needs a touch-sensitive element, registered event listeners, and a handler that reads the touch coordinates. The example develops this small structure before adding further behavior.

The basic implementation registers the required listeners on the interactive element and reads the touch information delivered to the handler. From that information, the application can identify the location and target of the contact. Only the events that contribute to the intended behavior should be handled, because unnecessary listeners add complexity and may interfere with normal page movement.

Example - 1

- We define a touch-sensitive div to which we will attach an event listener. We also define a div at the top where we will display the screen coordinates of the most recent touch.
- Next we add some JavaScript to register the event listener, and add a handler function to do something with the touch data:
- So, here we're simply identifying the touchzone div, attaching a listener for the touchstart event, and registering a handler function: touchHandler.

In the touchHandler function we grab the x and y coordinates of the touch, and write them to the coords div:

Adding Mouse Support

Touch support should not remove mouse support. The same application can preserve a familiar desktop interaction while sharing as much handling logic as possible.

Mouse support can use the same application operations that are called by touch handling. The event that starts the interaction may differ, but the work performed by the application should be shared wherever possible. This keeps the visible behavior consistent and prevents two independent implementations from drifting apart.

Pointer Events

Pointer events provide one event model for mouse, touch, and pen input. This reduces the need for parallel handlers and gives the application a consistent way to inspect coordinates, pressure, contact size, tilt, and pointer type.

The pointer model treats mouse, touch, and pen as variations of a common input source. A handler can still determine the pointer type, but it receives a consistent event sequence and a richer set of attributes. Pointer identity is useful for tracking contacts, while pressure, dimensions, and tilt describe how a supported pointer is interacting with the screen.

Problem

Separate mouse, touch, and pen event models require separate handlers and increase implementation complexity.

To handle the mouse events, touch events and pen events we have to write the code separately for each of these.

Solution

Pointer events provide one event model that can represent mouse, touch, and pen input.

To rectify this situation, Microsoft made a standards proposal that reconciles touch events and mouse events and pen input too, into one model called pointer events.

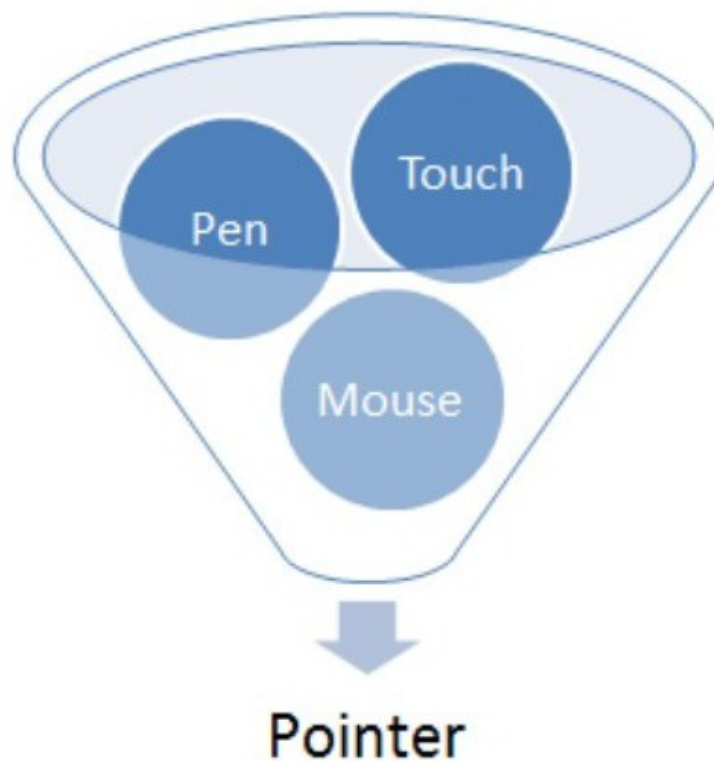


Figure 30. Unified pointer model.

Support in Browsers

Browser support determines whether the native pointer interface can be used directly or whether compatibility support is required. The application should still keep its interaction model focused, because a polyfill can supply an interface but cannot simplify unnecessarily duplicated application logic.

Go to www.caniuse.com Pointer Events Object While the Touch Events API was defined in terms of Touches, the Pointer Events API is defined in terms of Pointers, where a Pointer is defined as: A hardware agnostic representation of input devices that can target a specific coordinate (or set of coordinates) on a screen PointerEvent inherits and extends MouseEvent, so it has all the usual properties that MouseEvent has, such as clientX, clientY etc., as well as a few additional ones, such as tiltX, tiltY, and pressure

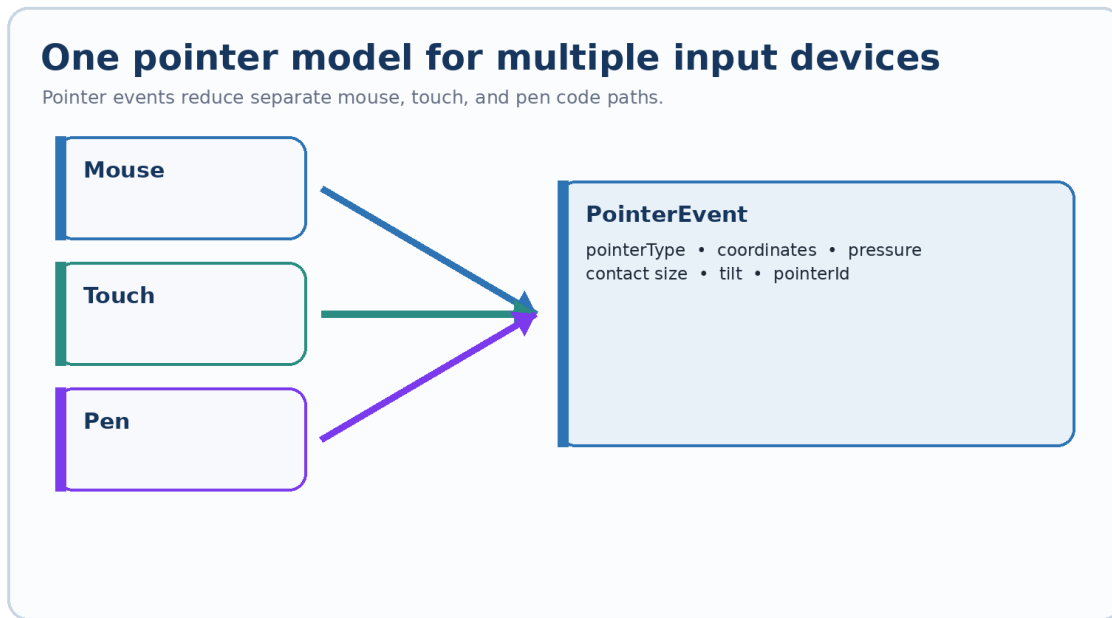


Figure 31. Unified pointer-event model.

The pointer attributes

Pointer attributes describe both position and the physical character of the contact, such as pressure, size, tilt, and pointer type.

Attribute	Description
pointerId	Unique numeric identifier
screenX / screenY	Coordinates relative to the screen
clientX / clientY	Coordinates relative to the viewport, excluding scroll offset
pageX / pageY	Coordinates relative to the page, including scroll offset
width / height	Dimensions of the pointer contact on screen
tiltX / tiltY	Stylus tilt angles relative to the screen axes
pressure	Pressure of contact on the screen
pointerType	Pointer class: mouse, pen, or touch
isPrimary	Whether this is the main pointer for its pointer type

Event types defined by the PointerEvent interface

The pointer-event sequence covers entry, movement, active contact, release, cancellation, and pointer capture.

Event type	Fired when...
pointerover	A pointer moves over an element and enters its hit-test boundaries.
pointerenter	A pointer moves over an element or one of its descendants; unlike pointerover, it does not bubble.
pointerdown	An active button state begins: touch or stylus contacts the screen, or a mouse button is pressed.
pointermove	Coordinates, pressure, tilt, or button state changes without another event firing.
pointerup	The active button state ends: a stylus or finger leaves the screen, or a mouse button is released.
pointercancel	The pointer is determined to have ended, such as after an orientation change, palm rejection, or too many pointers.
pointerout	The pointer leaves an element's hit-test boundaries; it also follows pointerup or pointercancel on devices without hover.
pointerleave	The pointer leaves an element and its descendants.
gotpointercapture	An element becomes the target of the pointer.
lostpointercapture	An element loses pointer capture.

Mouse events, pointer events, and touch events equivalence

The equivalence table shows how the start, movement, and release stages correspond across the three event families. Pointer events cover the shared sequence with pointerdown, pointermove, and pointerup while also defining entry, exit, cancellation, and capture behavior in the same model.

Mouse event	Touch event	Pointer event
mousedown	touchstart	pointerdown
mouseenter		pointerenter
mouseleave		pointerleave
mousemove	touchmove	pointermove

Mouse event	Touch event	Pointer event
mouseout		pointerout
mouseover		pointerover
mouseup	touchend	pointerup

Polyfill Library

Polyfill Library

A polyfill supplies a web feature in browsers that do not yet implement it natively.

It refers to a JavaScript library that implements an HTML5 web standard, either an established standard (supported by some browsers) on older browsers, or a proposed standard (not supported by any browsers) on existing browsers.

Pointer Code Example

The code example applies the pointer-event model in practice. It includes compatibility support, registers the required event, and uses pointer information without taking over more browser behavior than necessary.

The example demonstrates how compatibility support is included, how a pointer listener is registered, and how information from the event is read. The surrounding guidelines are equally important: use the smallest set of handlers that completes the interaction and avoid cancelling default behavior when the browser should continue to scroll or respond normally.

Pointer Code Example

The code registers the pointer event needed by the component and reads the event object inside the handler. Compatibility code may be included before the handler is used. The example remains small so that the relationship between registration, event delivery, pointer information, and application response is clear.

```

<html lang="en">
<head>
<meta charset="utf-8">
<title>PEP (Pointer Events Polyfill)</title>
<meta name="viewport" content="width=device-width">
<!-- include PEP -->
<script src="https://code.jquery.com/pep/0.4.1/pep.js"></script>
</head>
<body>
<button id="b" touch-action="none">Test button!</button>
<p><output id="o"></output></p>
<script>
document.getElementById( "b" ).addEventListener( "pointerdown", function( e ) {
document.getElementById( "o" ).innerHTML = "that was a " +
e.pointerType + " " + e.type + " on a " + e.target.nodeName;
} );
</script>
</body>
</html>

```

Pointer Code Example 2

Touch Handler Guide Lines

Use only the handlers that are necessary and preserve normal scrolling wherever possible.

Touch Handler Guide Lines It is good practice To use as few handlers as necessary in your application and keep the areas with touch handlers as tightly constrained as you can. That way the default scrolling mechanisms can take over as much as possible.

Input Introduction

Data entry is harder on a small touch keyboard, so forms should reduce effort and show suitable controls. HTML5 input types help the browser present keyboards and pickers that match the value being requested.

A suitable input type communicates the expected value to the browser. The browser can then show a relevant keyboard or native control, such as an email keyboard, number pad, date picker, or search field. This reduces typing effort and errors without requiring the page to create a replacement keyboard or picker.

Input Introduction

Mobile input should reduce typing effort, remain visible on small screens, and present a keyboard suited to the expected data.

- Some users can easily type longer text on mobile devices.
- But it is still very difficult to type with thumbs.

- So we must make input entry easy for all the users.

Also while entering data it should be clearly visible. It is much difficult to enter the small font data. Keyboard Input Phones and tablets do not have the benefit most desktops and laptops do, which is a large input device called the "keyboard". Virtual keyboards offered by mobile devices are physically smaller and usually more tedious to operate, but the great news is that HTML5 has already standardized a handful of input types which make them easier to use.

Input Types in HTML5

HTML5 input types help the browser present appropriate controls for email, telephone, numbers, dates, search, and other values.

The semantic type is both a hint and a contract. It tells the browser what kind of value is expected, allowing a more appropriate control, while the page can still validate and process the value through ordinary form behavior.

Input family	Included types
Text and contact	search, email, url, tel
Numeric	number, range
Date and time	date, month, week, time, datetime, datetime-local
Other	color

Text

A text input displays the general keyboard and is appropriate when the value does not belong to a more specific type. On mobile, it should be used deliberately because a specialized type may reduce the number of keyboard changes or taps required from the user.

The most common input type

```
<input type="text">
```

Email

The email type tells the browser that an email address is expected. A mobile browser can expose characters commonly needed in an address and can apply its email validation behavior, making entry more convenient than an undifferentiated text field.

Entering an email on the standard keyboard is very difficult. When we use Input Type as email. We get a @ sign as well as the .com and other handy buttons to just make entering an address as easy as possible.

```
<input type="email">
```

Tel

The telephone type encourages a phone-oriented keyboard while allowing the variety of symbols and formats used by telephone numbers. If the displayed number should initiate a call, it can separately be presented as a telephone URL.

Tel input provides the user with ability to add the numbers , it provides ability to quickly add *, # , + signs as well.

```
<input type="tel">
```

Number

The number type is intended for numeric values and can provide a numeric control or keyboard. Minimum, maximum, and step requirements should reflect the values the application actually accepts rather than forcing every numeric-looking identifier into arithmetic behavior.

Number input provides a key board to quickly add the numbers.

```
<input type="number">
```

Date

The date type allows a supporting mobile browser to present a date-oriented picker. This reduces format ambiguity and typing effort, while an unsupported browser can fall back to a text control and preserve basic input.

Date input types makes it very easy to enter the date on Webpage. Using Calendar can be little difficult for users

```
<input type="date">
```

Date & Time

Date-and-time controls combine calendar and time selection according to the specific type used. The native interface can vary between devices, but the semantic input type continues to describe the expected value to the browser.

Entering Date and Time is very Difficult but for mobile devices, using Date & Time input it becomes very easy to enter data.

```
<input type="datetime">
```

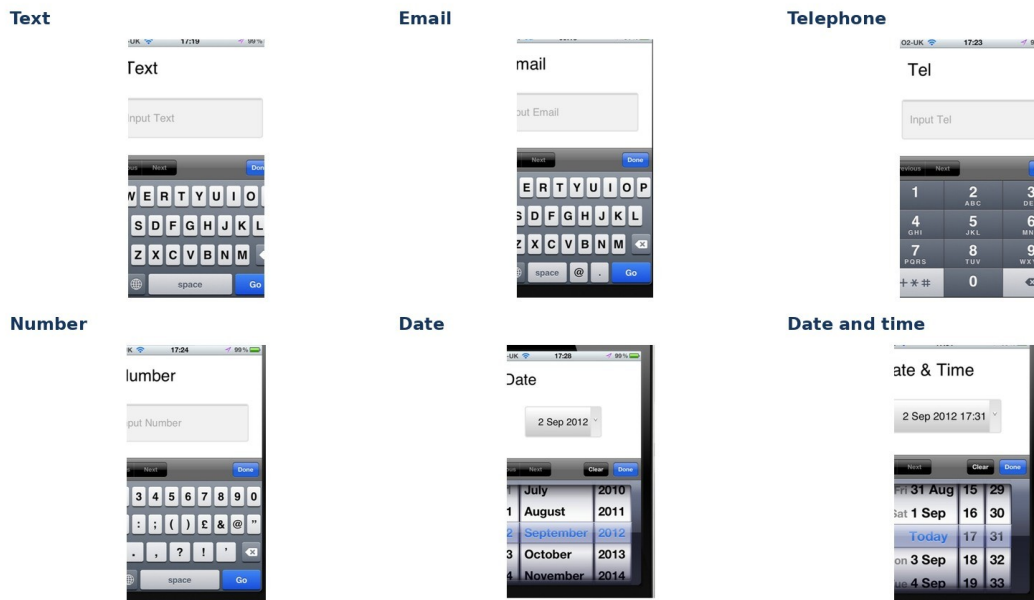


Figure 32. Mobile keyboards and date controls by input type.

- Month
- Search

Regular Expression Validation

Client-side validation can check whether entered text follows an expected pattern before submission. The pattern attribute uses a JavaScript-style regular expression and works alongside the browser's normal validation behavior.

A pattern expresses the allowed structure of the value, and the browser compares the entered text with that expression during validation. The expression should match the format the form actually expects and should be paired with a label or instruction that tells the user what to enter. Validation is most helpful when its requirement is understandable before submission.

For some types like e-mail, there's some basic validation. But this is not very smart validation. If you want to do better client side validation though, HTML-5 supports that too. There's also a pattern attribute on the input element, that you can use to validate on the client side. The pattern element takes a regular expression, as expressed by the java script regex syntax.

Example

For example, let's presume we want to only allow Country code in a specific field in our mobile web application like PAK

I can put in a regular expression for this, and now when I try to input a Country code of more than three letters and then submit, it pops up an error message that says I have to match the requested format.

Example 2

An `<input>` element with `type="password"` that must contain 6 or more characters:

To Learn more about Regular Expressions

Regular expressions can describe much more than the short validation example, but the form should use only the complexity necessary for its expected value. A smaller expression that matches the documented format is easier to test and maintain than a broad pattern whose behavior is difficult to explain.

- http://www.w3schools.com/jsref/jsref_obj_re_gexp.asp To Learn more about Regular Expressions

Two Important Features of Input

The required and placeholder attributes solve different problems. Required makes a value mandatory, while placeholder gives a short hint; neither removes the need for a clear and accessible label.

Required controls whether an empty value is acceptable, while placeholder presents a temporary example or hint inside the field. The placeholder disappears as the user types, so it cannot carry the full meaning of the control. A visible label remains necessary to identify the field throughout the interaction.

Two Important Features of Input

Required and placeholder are independent attributes and may be used together. Required participates in validation, whereas placeholder affects the hint shown inside the empty field. Neither attribute establishes the permanent accessible name supplied by a label.

- There are two more features I wanted to mention on inputs.
- The required attribute and the place holder attribute.
- Required just lets me say, this field is required before you can submit the form. So,

Example

For example, if I want to make sure that a user enters the username,

Now if user will try to submit this form with out entering the username he will get the error message

Placeholder Text

Placeholder text gives a short hint about the expected value, but it does not replace a proper label.

A placeholder is best used for a brief example, not for instructions that the user must remember. Once typing begins, the hint is no longer visible; the permanent label continues to provide context.

The other feature, is the placeholder text. You may want to give hints to your users. In some fields, they go away when they start typing. You can do that pretty easily by just setting a placeholder attribute.

- The placeholder attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format).
- The short hint is displayed in the input field before the user enters a value.
- Note: The placeholder attribute works with the following input types: text, search, url, tel, email, and password.

Example

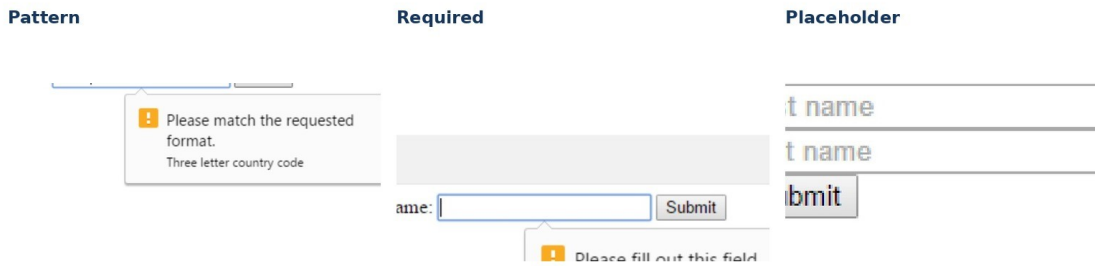


Figure 33. Pattern, required-field, and placeholder behavior.

Input in older browsers

An older browser may not recognize a newer input type. The specification allows it to fall back to an ordinary text field, so the form remains usable even when the enhanced control is unavailable.

Fallback behavior allows a form to adopt newer input types without losing basic usability. When the browser does not implement the specialized type, the control behaves as a text field. The enhanced keyboard or picker may be absent, but the user can still enter and submit the value.

- The HTML spec says that unrecognized input types are to be
- treated as text. So, new input types will still work even if they are not supported on a particular browser.
- They'll just function like text inputs. Not quite as useful, but still functional.

Wrapping in Label

A label explains the purpose of an input and also improves touch usability. Wrapping the control inside its label makes the text and the control part of the same larger touch target.

The relationship between a label and its input benefits both meaning and interaction. On a touch device, the label text becomes an additional area that can activate or focus the control. This is particularly useful for small controls because the user does not need to touch the control with pixel-level precision.

You should always wrap label elements around your input elements, and their associated labeling texts, because it increases the touchable area of the control.



Figure 34. Mobile input controls.

- The only difference between these two controls is that one of them has a label element around it.
- But you'll notice, it's much easier to touch the one with the label. The one without a label, you have to actually touch the check box itself, in order to activate the control.

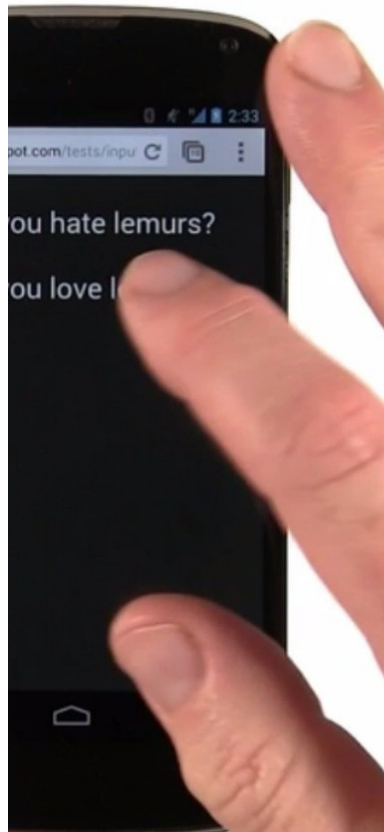


Figure 35. Touchable label area.

telecolon URL

A telephone URL lets a mobile device open its dialer with a number already filled in. Because desktop systems may not provide the same behavior, the page should present the number in a form that still makes sense there.

The telephone scheme turns a displayed number into an actionable mobile link. Selecting it can open the dialer with the number already entered, leaving the user to confirm the call. The number should also remain readable as text so the content continues to make sense where a dialer is not available.

On mobile, you can use a telecolon URL to create a URL that when activated will actually dial a telephone number. This is a really handy feature, particularly for business websites. Customers can instantly dial you.

- Of course, on desktop, these URLs will typically fail, so you probably want to deactivate them.

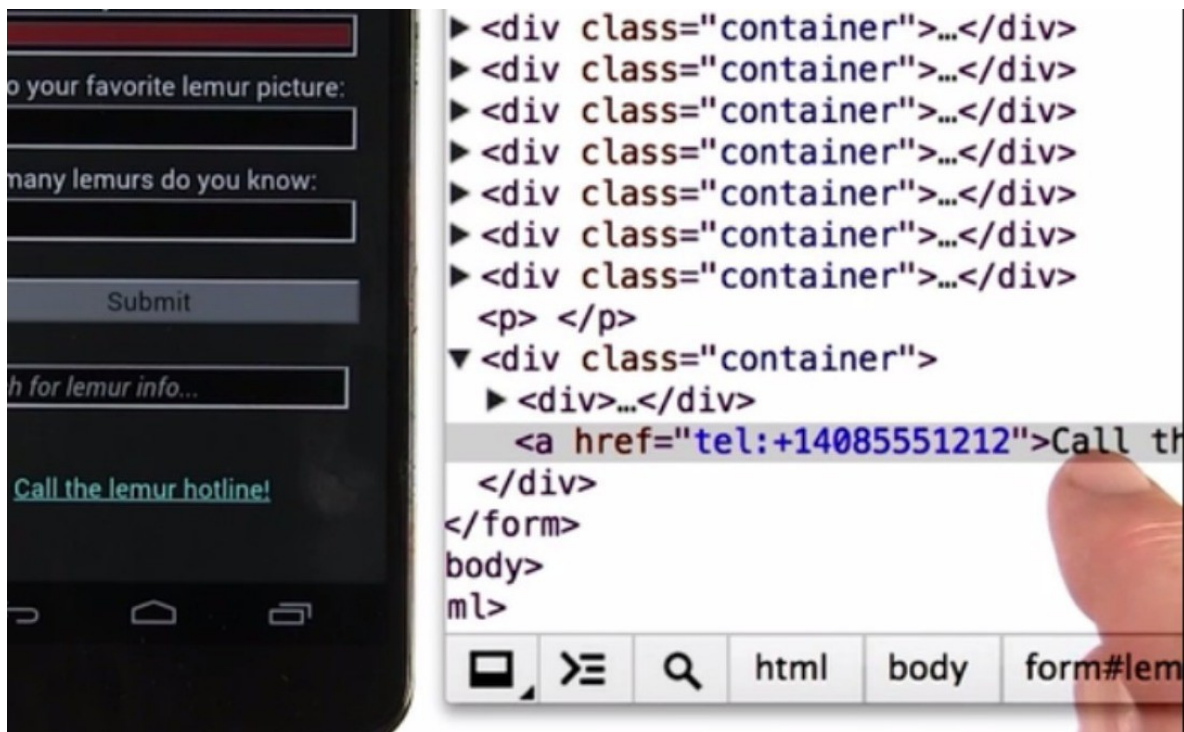


Figure 36. Telephone URL markup.

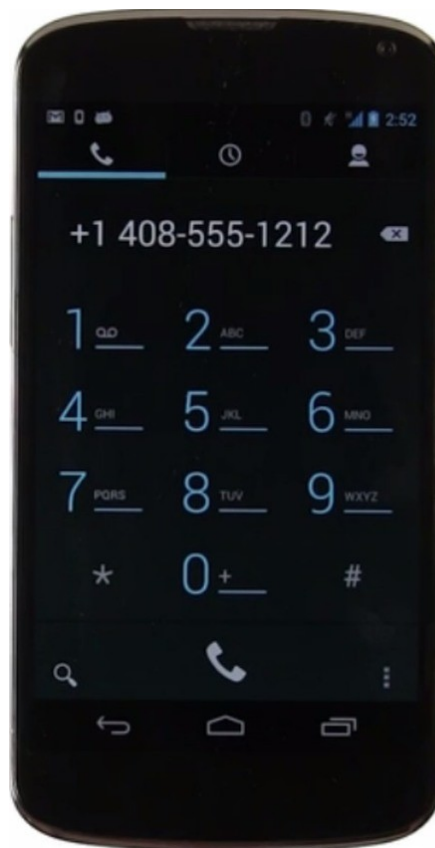


Figure 37. Telephone URL dialer.

Sensors on mobile

Portable devices expose more than a screen and keyboard. Cameras, microphones, location, compass, and motion sensors can become inputs to a web experience when used through the appropriate browser features.

The mobile browser can serve as a bridge between a web page and selected device capabilities. Camera and microphone access are discussed in detail, while location, compass, and motion illustrate the broader range of available sensor input. Access must follow the browser mechanism designed for the particular capability.

One of the most amazing things about the mobile platform is the incredible array of sensors these devices typically have. This phone has a camera that can take stills and video, has audio input and output, it has a GPS, a compass, a tilt sensing accelerometer.



Figure 38. Sensor-rich mobile device.

A Mobile device has far more sensors than a desktop or laptop typically has. And all this, in a device that one can typically hold in the palm of his/her hand. This enables the mobile platform to play host to incredibly engaging and interactive user experiences.

Camera Access

Camera input can support tasks such as profile pictures, QR-code capture, maps, and business cards. The application should choose an access method that matches whether it needs a saved file or a live stream.

A still image may be sufficient when the application needs a profile picture or a captured document, while a live stream is more appropriate when the page must show or process the camera continuously. The choice affects how much control the page has over the interface and whether the user leaves the current application during capture.

The camera is not just for taking pictures and making videos. The camera can be a great way to get quick input into the mobile device. For example you can snap a new profile picture using your mobile camera and upload it

- You capture a QR code to transfer a hyperlink
- You can capture a map or a business card.

QR code



Business card



Figure 39. Camera capture examples: QR code and business card.

Capture Extension

The capture extension uses a file input to request an image, video, or audio recording from the device. It is simple to add, but its interface and behavior are largely controlled by the mobile operating system.

The capture approach builds on a file input and lets the operating system provide the recording interface. The page receives the resulting file after the external capture step is

complete. This keeps the web code small, but it also means that the operating system controls much of the user experience.

There is a simple way to get access to the camera and the audio input, the capture extension on the accept attribute on the file input element in HTML. This will give you a button that opens up the system camera or Audio recorder app on a mobile device.

Accept Attribute

The accept attribute narrows the media or file types offered by the picker. It guides the user toward an image, audio recording, video, or another expected file, while the capture extension can additionally request acquisition from an appropriate device source.

The accept attribute takes a comma-separated list of unique content types of files that are acceptable values for the file input type only. The comma separated values can include file extensions, MIME types without extensions,

```
<input type="file" accept="image/*">
```

(images only)

For Mobile Devices

```
<input type="file"
accept="image/*;capture=camera">
<input type="file"
accept="video/*;capture=camcorder">
<input type="file"
accept="audio/*;capture=microphone">
```

Drawbacks of Capture Extension

The capture approach is convenient but limited: support is mobile-focused, the control is difficult to style, and recording may move the user into another application before returning the result.

The same simplicity creates its limitations. The capture control cannot be styled or managed like an interface designed inside the page, support is concentrated on mobile devices, and the transition to another application can interrupt the flow. These trade-offs motivate the live-media approach.

The first problem is that this feature only works in mobile, not in desktop. On desktop, you only get the normal file open dialogue. Not access to the webcam.

- Secondly, the interface is a file open button, which is hard to style and make look the way you might want it to.
- Third issue is , taking you to a different application to take a picture or record some audio, makes it kind of hard to keep a consistent flow in your application.

Live Input

Live input keeps audio or video inside the page. A video stream can be displayed directly, drawn onto a Canvas as a snapshot, and converted into image data for further use.

With live input, the stream is connected directly to a media element in the page. A frame can then be drawn to Canvas to create a snapshot, and the Canvas data can be converted into an image representation. The page can therefore display and capture camera content without sending the user through a separate recording application.

We can get Live Input, Audio and Video, directly under our App by using an API called `getUserMedia`. Link:- <https://developer.mozilla.org/en/docs/Web/API/Navigator/getUserMedia>

Example

```
var constraints = { video: true };
navigator.getUserMedia = navigator.getUserMedia ||
    navigator.webkitGetUserMedia ||
    navigator.mozGetUserMedia;

function errorCallback(error) {
    console.log("getUserMedia error", error);
}

navigator.getUserMedia(constraints, successCallback, errorCallback);
```

This is a simple example where we just call `getUserMedia`, passing some constraints that say we want video, and then we assign the resulting stream to a video element in the page. Of course, this just gives us a rectangle with live Video Input.

```
function snapshot() {
    var video = document.querySelector("video");
    var canvas = document.querySelector("canvas");
    canvas.width = video.videoWidth;
    canvas.height = video.videoHeight;
    var ctx = canvas.getContext("2d");
    ctx.drawImage(video, 0, 0);
}
```

We probably want to do something with it. If we want to grab Snapshots like if we want to implement a Camera Application, we can do this with a canvas context and `DrawImage`.

Whenever we want to take a Snapshot, we simply grab the image from the video and we draw it to the camera with DrawImage.

```
function snapshot() {  
    canvas.width = video.videoWidth;  
    canvas.height = video.videoHeight;  
    img.width = video.videoWidth;  
    img.height = video.videoHeight;  
    var ctx = canvas.getContext("2d");  
    ctx.drawImage(video, 0, 0);  
    img.src = canvas.toDataURL("image/png");  
}
```

Of course a canvas is different than an image. You can't directly copy a canvas and paste it somewhere else, or download it to your hard drive or post it to your social networks. Or any of the other typical User flows centered around images, but it turns out, canvas has a handy function to encode itself as an image. The canvas toDataURL method. Here, instead of displaying the canvas itself, we're copying the canvas contents as a data URL to the source of an image tag. And if you want to save these images to the Mobile's local storage.

Navigator.getUserMedia()

getUserMedia requests audio or video that satisfies a set of constraints. The success path receives the media stream, while the error path explains why the requested device or permission was unavailable.

The request begins with constraints that describe whether audio, video, or particular media characteristics are needed. The browser checks the available devices and permission. A successful request passes a MediaStream to the success callback; a failed request invokes the error callback with information such as permission denial or the absence of a matching device.

Syntax

The media request is expressed as a call that supplies constraints and handlers for the two possible outcomes. Reading the syntax from left to right shows the complete contract: describe the required media, process the returned stream, and respond when the request cannot be completed.

```
navigator.getUserMedia(constraints,  
    successCallback, errorCallback);
```

Parameters

The parameters separate the request from its results. Constraints specify the desired audio or video tracks; the success callback receives the `MediaStream`; and the error callback receives information about the failure. Each parameter therefore represents a distinct stage of the same operation.

Constraints A `MediaStreamConstraints` object specifying the types of media to request, along with any requirements for each type.

- To require a capability, use the keywords `min`, `max`, or `exact` (a.k.a. `min == max`). The following demands a minimum resolution of 1280x720

successCallback

When the browser grants access and finds media that satisfies the constraints, the success callback receives the stream. The page can attach that stream to a media element for live playback and can use the displayed video as the source for a Canvas snapshot.

When the call succeeds, the function specified in the `successCallback` is invoked with the `MediaStream` object that contains the media stream. You may assign that object to the appropriate element and work with it, as shown in the following example:

```
function(stream) {  
    var video = document.querySelector("video");  
    video.src = window.URL.createObjectURL(stream);  
    video.onloadedmetadata = function(e) {  
        // Do something with the video here.  
    };  
}
```

errorCallback

The error callback is necessary because a camera or microphone request is not guaranteed to succeed. Permission may be denied, a suitable device may be missing, or the requested constraints may not be met. The page should respond without assuming that live media is always available.

When the call fails, the function specified in the `errorCallback` is invoked with a `MediaStreamError` object as its sole argument

Error	Description
PermissionDeniedError	Permission to use a media device was denied by the user or the system.
NotFoundError	No media tracks of the specified type were found that satisfy the constraints.

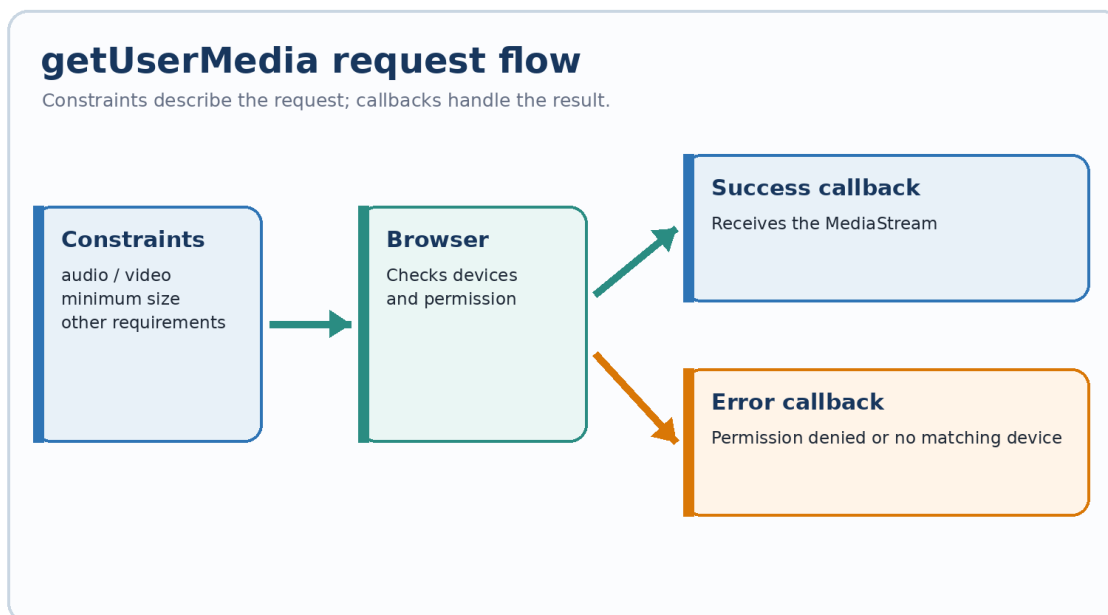


Figure 40. *getUserMedia* request, permission, and callback flow.