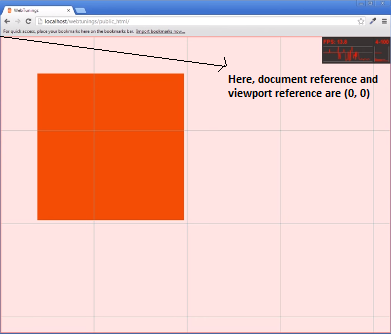
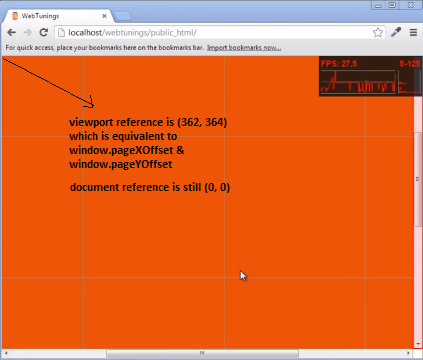
**Desktop browsers:**



After zoom-in of element,



**A tale of two viewports — part one**

**This mini-series explains how viewports and the widths of various important elements work, such as the <html> element, as well as the window and the screen.**

This is about the desktop browsers, and its sole purpose is to set the stage for a similar discussion of the mobile browsers. Most web developers will already intuitively understand most desktop concepts. On mobile we’ll find the same concepts, but more complicated, and a prior discussion on terms everybody already knows will greatly help your understanding of the mobile browsers.

### Concept: device pixels and CSS pixels

The first concept you need to understand is CSS pixels, and the difference with device pixels.

Device pixels are the kind of pixels we intuitively assume to be “right.” These pixels give the formal resolution of whichever device you’re working on, and can (in general) be read out from screen.width/height.

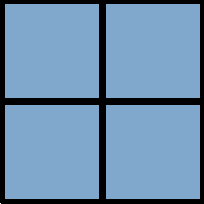
If you give a certain element a width: 128px, and your monitor is 1024px wide, and you maximise your browser screen, the element would fit on your monitor eight times (roughly; let’s ignore the tricky bits for now).

If the user zooms, however, this calculation is going to change. If the user zooms to 200%, your element with width: 128px will fit only four times on his 1024px wide monitor.

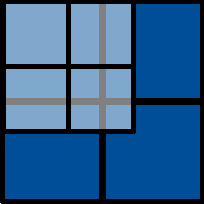
Zooming as implemented in modern browsers consists of nothing more than “stretching up” pixels. That is, the width of the element is not changed from 128 to 256 pixels; instead the actual pixels are doubled in size. Formally, the element still has a width of 128 CSS pixels, even though it happens to take the space of 256 device pixels.

In other words, zooming to 200% makes one CSS pixel grow to four times the size of one device pixels. (Two times the width, two times the height, yields four times in total).

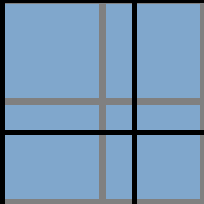
A few images will clarify the concept. Here are four pixels on 100% zoom level. Nothing much to see here; CSS pixels fully overlap with device pixels.



Now let’s zoom out. The CSS pixels start to shrink, meaning that one device pixel now overlaps several CSS pixels.



If you zoom in, the opposite happens. The CSS pixels start to grow, and now one CSS pixels overlaps several device pixels.



The point here is that *you are only interested in CSS pixels*. It’s those pixels that dictate how your style sheet is rendered.

Device pixels are almost entirely useless to you. Not to the user; the user will zoom the page in or out until he can comfortably read it. However, that zooming level doesn’t matter to you. The browser will automatically make sure that your CSS layout is stretched up or squeezed in.

#### 100% zoom

I started the example by assuming a zoom level of 100%. It’s time to define that slightly more strictly:

At zoom level 100% one CSS pixel is exactly equal to one device pixel.

The concept of 100% zoom is very useful in the explanations that are going to follow, but you shouldn’t overly worry about it in your daily work. On desktop you will generally test your sites in 100% zoom, but even if the user zooms in or out the magic of CSS pixels will make sure that your layout retains the same ratios.

### Screen size

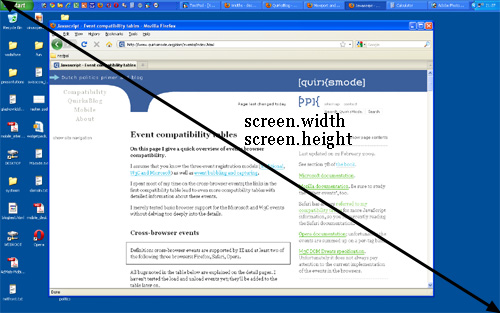
*screen.width/height*

Meaning - Total size of the user’s screen.

Measured in Device pixels

Browser errors - IE8 measures it in CSS pixels, in both IE7 and IE8 mode.

Let’s take a look at some practical measurements. We’ll start with screen.widthand screen.height. They contain the total width and height of the user’s screen. These dimensions are measured in device pixels because they never change: they’re a feature of the monitor, and not of the browser.



Fun! But what do we do with this information?

Basically, nothing. The user’s monitor size is unimportant to us — well, unless you want to measure it for use in a web statistics database.

### Browser window size

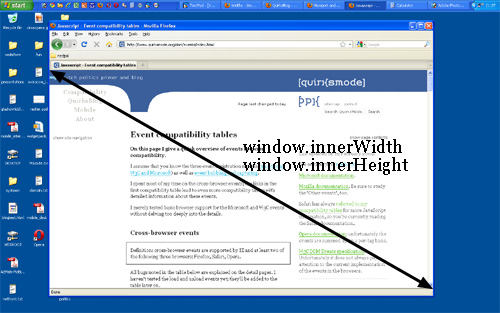
*window.innerWidth/Height*

Meaning - Total size of the browser window, including scrollbars.

Measured in CSS pixels

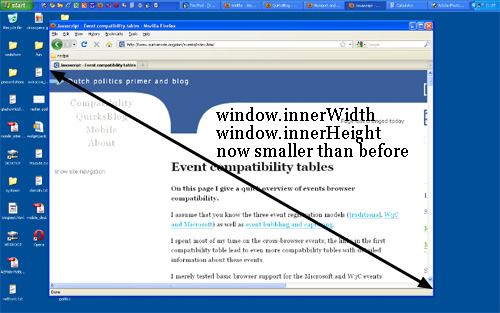
Browser errors - Not supported by IE. Opera measures it in device pixels.

Instead, what you want to know is the inner dimensions of the browser window. That tells you exactly how much space the user currently has available for your CSS layout. You can find these dimensions in window.innerWidth andwindow.innerHeight.



Obviously, the inner width of the window is measured in CSS pixels. You need to know how much of your layout you can squeeze into the browser window, and that amount decreases as the user zooms in. So if the user zooms in you get less available space in the window, and window.innerWidth/Height reflect that by decreasing.

(The exception here is Opera, where window.innerWidth/Height do not decrease when the user zooms in: they’re measured in device pixels. This is annoying on desktop, but fatal on mobile, as we’ll see later.)



Note that the measured widths and heights include the scrollbars. They, too, are considered part of the inner window. (This is mostly for historical reasons.)

### Scrolling offset

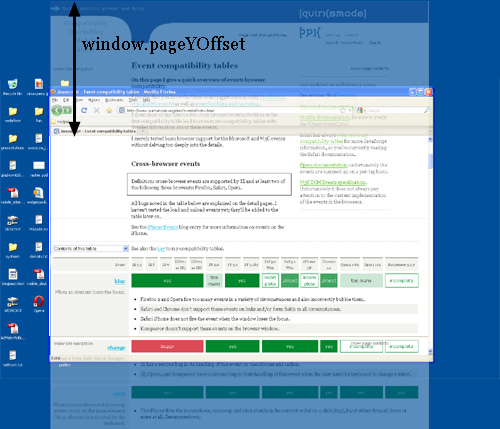
*window.pageX/YOffset*

Meaning - Scrolling offset of the page.

Measured in CSS pixels

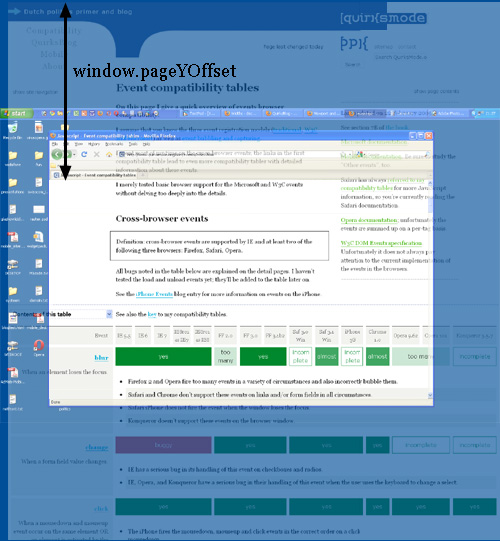
Browser errors - None

window.pageXOffset and window.pageYOffset, contain the horizontal and vertical scrolling offsets of the document. Thus you can find out how much the user has scrolled.



These properties are measured in CSS pixels, too. You want to know how much of the document has already been scrolled up, whatever zoom state it’s in.

In theory, if the user scrolls up and then zooms in, window.pageX/YOffset will change. However, the browsers try to keep web pages consistent by keeping the same element at the top of the visible page when the user zooms. That doesn’t always work perfectly, but it means that in practice window.pageX/YOffsetdoesn’t really change: the number of CSS pixels that have been scrolled out of the window remains (roughly) the same.



### Media queries

##### Media queries

Browser errors - IE doesn’t support them.

For device-width/height Firefox uses the values screen.width/height would have if they are measured in CSS pixels.

For width/height Safari and Chrome use the values documentElement.clientWidth/ Height would have if they are measured in device pixels.

Finally, some words about media queries. The idea is very simple: you can define special CSS rules that are executed only if the width of the page is larger than, equal to, or smaller than a certain size. For instance:

div.sidebar {

width: 300px;

}

@media all and (max-width: 400px) {

// styles assigned when width is smaller than 400px;

div.sidebar {

width: 100px;

}

}

Now the sidebar is 300px wide, except when the width is smaller than 400px, in which case the sidebar becomes 100px wide.

The question is of course: which width are we measuring here?

There are two relevant media queries: width/height and device-width/device-height.

1. width/height uses the same values as documentElement.clientWidth/Height (the viewport, in other words). It works with CSS pixels.
2. device-width/device-height uses the same values as screen.width/height (the screen, in other words). It works with device pixels.



Which should you use? That’s a no-brainer: width, of course. Web developers are not interested in the device width; it’s the width of the browser window that counts.

So use width and forget device-width — on desktop. As we’ll see, the situation is much more messy on mobile.

**Mobile browsers:**

We are going to talk about the mobile browsers.

### *The problem of mobile browsers*

When we compare the mobile browsers to the desktop ones, the most obvious difference is screen size. Mobile browsers display significantly less of a desktop-optimised website than desktop browsers; either by zooming out until the text is unreadably small, or by showing only the small part of the site that fits in the screen.

A mobile screen is far smaller than a desktop screen; think about 400px wide at maximum, and sometimes a lot less. (Some phones report larger widths, but they’re lying — or at the very least giving us useless information.)

An intermediate layer of tablet devices such as the iPad or the rumoured HP webOS-based one will bridge the gap between desktop and mobile, but that doesn’t change the fundamental problem. Sites must work on mobile devices, too, so we have to get them to display well on a small screen.

The most important problems center on CSS, especially the dimensions of the viewport. If we’d copy the desktop model one-to-one, our CSS would start to misfire horrendously.

Let’s go back to our sidebar with width: 10%. If mobile browsers would do exactly the same as desktop browsers, they’d make the element about 40px wide at most, and that’s far too narrow. Your liquid layout would look horribly squashed.

One way of solving the problem is building a special website for mobile browsers. Even apart from the fundamental question of whether you should do that at all, the practical problem is that only very few site owners are sufficiently clued-in to cater specifically to mobile devices.

Mobile browser vendors want to offer their clients the best possible experience, which right now means “as much like desktop as possible.” Hence some sleight of hand was necessary.

The layout viewport's dimensions are set by the meta-viewport rule.

<https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag>