

Responsive Web Design (RWD) Best Practices Guide

Version: 2013.11.20

This document includes best practices around responsive web design (RWD) when developing hybrid applications. Details on each checklist item are [discussed](#) later in the document.

Checklist

RWD Practitioner

- Use CSS3 media queries rather than JavaScript `onorientationchange` [?](#)
- Use flexible font units [?](#)
- Define a fluid grid system for layout [?](#)
- Use Flexible Margins and Padding [?](#)
- Make Images Responsive Where possible [?](#)
- Use CSS3 Media Queries [?](#)
- Accept limitations of CSS3 Media Queries [?](#)
- Consider using CSS Flexbox Layout [?](#)
- Design for Orientation Changes [?](#)
- Start and stay with simple layout [?](#)
- Define key horizontal (and vertical) breakpoints you need to support [?](#)

Common RWD Problems

- ❑ Avoid `onorientationchange` events to trigger DOM manipulation [↗](#)
- ❑ Manage complex web components with different size-based structures [↗](#)
- ❑ Managing graphically drawn components which require recalculation/redraw [↗](#)
- ❑ Repaint/flicker due to use of Dojo's `portrait` and `landscape` classes in CSS [↗](#)

Discussion

RWD Practitioner

This section contains technical information for implementing responsive design relevant to practitioners such as CSS experts and web developers. Many of the topics discussed in this documents are also covered in and/or related topics in the following documents and should be used in parallel.

- *CSS Best Practices*
- *JavaScript Best Practices*
- *Images Best Practices*

Use CSS Media Queries rather than JavaScript orientationchange events

When you rotate a device's orientation, the browser engine first reflows the content to the new orientation/screen dimensions (using CSS rules). After the reflow occurs, onorientationchange events are emitted to JavaScript. This means that if you are doing positioning in JavaScript in response to these events, you will incur multiple repaints which has a very negative effect on performance.

The onorientationchange event occurs after the browser has already reflowed content. CSS is used during the automatic reflow. By the time JavaScript receives these events, there has already been a redraw of the screen. Avoid any layout/resizing or content changes in response to these events. Wherever possible use CSS classes to quickly enable/disable cached instances of DOM content instead.

To get the fastest layout with minimal repainting possible, always use CSS media queries.

Use Flexible Font Units

- Set body font size to 100%, so later can use ems to size the font.
- Use **em** for font-size
 - Most browser's default size is 16px, use that as base for em calculation. Browser converts em to pixel internally
 - Compounding effect: because it is relative to the font-size of the parent
- Use **rem** units
 - Introduced in CSS3
 - Root em – relative to the root "html" element

```
html { font-size: 100%; }
h1 { font-size: 1.4rem; }
```

Define a Fluid Grid System for layout

- How to design a responsive layout
 - First float the content
 - Define content as `` elements
- Each section uses percentage width (relative)

1-Column

1/1

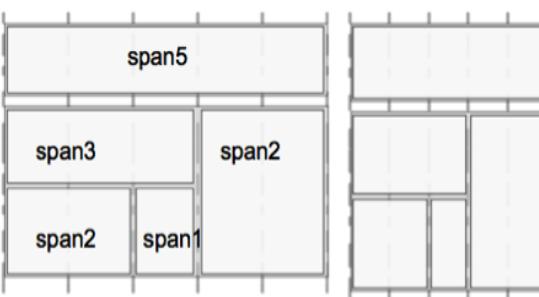
Etiam factum non deponit quid pro quo hic esset. Olypian quamvis et gortia conglutium sic ad nauseam. Souviali ignitus carbonandum e pluribus unam. Defacto lingua est iggyi abntay. Marquee selectus non provisio incongruous ferre note contentia. Duis autem vel eum inure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugiat nulla facilisis. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

2-Column Grid

1/2	1/2
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.	Duis autem vel eum inure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugiat nulla facilisis.

3-Column Grid

1/3	1/3	1/3
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.	Duis autem vel eum inure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugiat nulla facilisis.	Etiam factum non deponit quid pro quo hic esset. Olypian quamvis et gortia conglutium sic ad nauseam. Souviali ignitus carbonandum e pluribus unam. Defacto lingua est iggyi abntay. Marquee selectus non provisio incongruous ferre note contentia.
1/3	2/3	
Duis autem vel eum inure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugiat nulla facilisis.	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.	

	<pre> .span5 { width: 100%; } .span3 { width: 60%; } .span1 { width: 20%; } ... </pre>
	<pre> <div class="row"> <div class="span3"> ... </div> <div class="span1"> ... </div> <div class="span1"> ... </div> </div> </pre>

Use Flexible Margins and Padding

- It is important to understand the box model of CSS in order to get most of the box model size to relative dimensions(percentage) rather than absolute dimensions.

- When setting flexible **margins** on an element, your context is the width of the element's container.
- When setting flexible **padding** on an element, your context is *the width of the element itself*. This makes sense, if you think about the box model: we're describing the padding in relation to the width of the box itself.
- You can use negative margins to assist with the positioning

Make Images Responsive Where possible

- Why? It prevents images from exceeding the width of their container
- Max-width
 - Discovered by Richard Rutter
 - Use max-width : 100%
 - Can be applied to: img, embed, object, video
 - Alternative: overflow: hidden
- Make flexible background image
 - background-size
 - Immature browser support
- For information/content rich images
- Multi-resolution images

Use CSS3 Media Queries



- Limit the style sheets' scope by using media features, such as width, height or orientation.
- Apply rendering rules by inspecting devices and browser physical characteristics

- Two parts in a Media query:
 - Type (screen)
 - Query (orientation: portrait)
- It's all about test your browser and device:
 - width, height, device-width, device-height (pay attention on differences between the browser viewport and device rendering surface)
 - orientation
 - aspect-ratio, device-aspect-ratio
 - color, color-index, monochrome
 - resolution
 - scan
 - grid

- Media Query can be chained!

`@media screen and (min-width: 480px) and (orientation: portrait)`

- Set the viewport

`<meta name="viewport" content="width=320" />`

- First introduced by Apple in Mobile Safari
- Use it to control the rendering canvas size and override default behavior
- For responsive design you can use resolution-agnostic viewport settings:

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

Note: Setting height=device-height is broken on iOS5/6, but now works on iOS7

Examples of applying media queries:

- CSS Links and Imports
 - `<link rel="stylesheet" media="(max-width: 800px)" href="example.css" />`
 - `@import url("portrait.css") screen (orientation: portrait);`
- CSS files
 - `@media (max-width: 600px) { .sidebar { display: none;}}`
- Javascript
 - `if(window.matchMedia("(min-width:400px)").matches) {`
 . . .
 }

Example Media Query for Phone form factor:

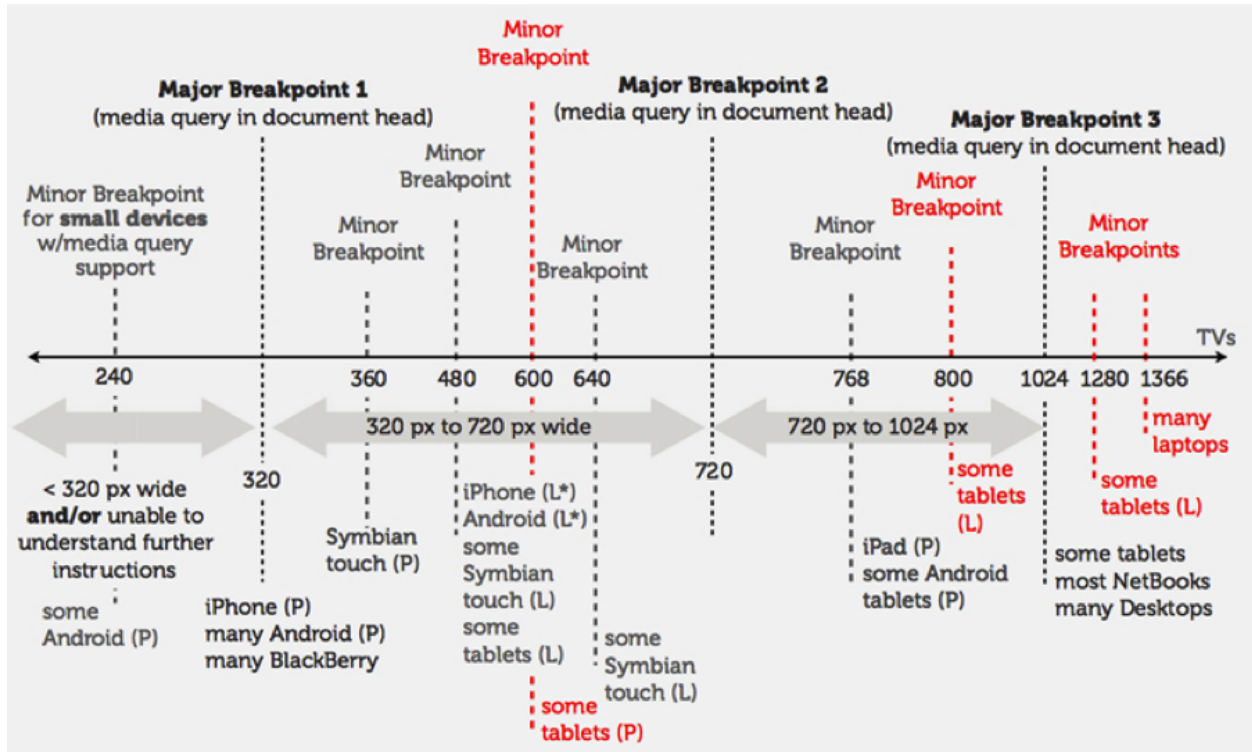
```
// Phone Layout
@media (max-width: 600px) {
```

```
.span5 {  
  width: 100%;  
}  
.span3 {  
  width: 100%;  
}  
.span1 {  
  width: 100%;  
}  
...  
}
```

Accept limitations of CSS3 Media Queries

- Media queries rules are limited to the whole screen. They cannot match to a subpart of the screen
- Component design: there are no perfect media query breakpoints. Take existing examples and adjust to your application requirements.

Figure 13



Consider using CSS Flexbox Layout

- W3C Candidate Recommendation (stable)
- Full support: iOS 7, BB10, IE11
- Android browser supports the previous syntax
- Direction-agnostic
- Powerful layout capabilities
- **display: flex;**



Design for Orientation Changes

- Need to design for orientation change
 - CSS media query based
 - Proper HTML mockup change (duplicate components for landscape, portrait)
- CSS Media Query is much faster in handling orientation change than JavaScript
 - CSS Media Query – following style will be rendered
@media screen and (orientation: portrait) {
 - JavaScript can subscribe to following event:
window.addEventListener("orientationchange", function() {})
 - Subscribing to orientation change event may still be needed.

Start and stay with simple layout

- Basic HTML5 element
- A reset stylesheet
- Set body font size to 100%, so later can use rems to size the font.

Define key horizontal (and vertical) breakpoints you need to support

Example:

- Smartphone **<480px**
- Tablets portrait **<768px**
- Tablets landscape **>768px**
- Desktop **>1024px**

Common RWD Problems

Avoid `onorientationchange` events to trigger DOM manipulation

It is common when an application supports multiple orientations or form factors to have web components or sections of web content that have very different DOM structure applied at different breakpoints.

Scan code for the use of `onorientationchange` events and inspect the callback handlers associated with these events. Make sure that there is no DOM processing in response to these callback handlers. DOM operations are very expensive and should be avoided. Use CSS media queries to hide/show different DOM elements.

Managing complex web components with different size-based structures

Sometimes for an application's design components such as tables or lists, calendars or other more complex web components will need to have different structure applied in different size breakpoints. An example of this is that different columns in a table may be shown or hidden in landscape vs. portrait.

The components require a JavaScript call to trigger resize/redraw of the components. There is currently no way to have these calls fire at the time of CSS reflow.

One workaround to make repaints as fast as possible when using these types of components is to use multiple instances of the web component (one for each major breakpoint having structural differences). Use CSS media query rules to enable/disable display of the instances appropriate to each breakpoint. This requires keeping multiple instances preallocated in memory for all the breakpoints. In hybrid mobile applications, this usually only means one instance per orientation supported for a given screen; however, for hosted mobile web with desktop browsers this may require instances at all breakpoints (small, medium, large) because the browser can be resized by the user.

It may also be possible to redesign the web components themselves to allow different structures to be associated with different breakpoints that are then shown/hidden using CSS media queries (enabling/disabling display of the structures in the CSS rules).

Managing graphically drawn components which require recalculation/redraw

Sometimes for certain types of complex web components or graphically drawn components such as charts drawn with SVG or Canvas, the components require a JavaScript call to trigger resize/redraw of the components. There is currently no way to have these calls fire at the time of CSS reflow.

A workaround to make repaints as fast as possible when using these types of components is to use multiple instances of the web component (one for each major breakpoint having structural differences). Use CSS media query rules to enable/disable display of the instances appropriate to each breakpoint. This requires keeping multiple instances preallocated in memory for all the breakpoints. In hybrid mobile applications, this usually only means one instance per orientation supported for a given screen; however, for hosted mobile web with desktop browsers this may require instances at all breakpoints (small, medium, large) because the browser can be resized by the user.

A second workaround that may or may not work depending on the application's requirements is to keep the dimensions of the graphical component the same for example in portrait and landscape and only change its relative position. In practice, we have found that it is rare that this workaround is feasible because typically designers want the web components to take advantage of extra whitespace available, requiring a forced repaint of the component.

Known examples of Dojo components requiring JavaScript on resize:

- dojox/Calendar - Calendar events or view type may change in different breakpoints.
- dojox/charting/Chart
- dojox/dgauge/Gauge
- dgrid - Column structure may change in different breakpoints
- gridx- Column structure may change in different breakpoints

Repaint/flicker due to use of Dojo's portrait and landscape classes in CSS

Dojo 1.6-1.9 include special css classes defined for specifying style rules that apply in portrait vs. landscape orientations.

Do not use these classes-- as the classes are not applied using media queries, but instead use onorientationchange events. This means that the css blocks specified with these classes get applied after the browser does it's automatic reflow, resulting in flicker/repaint problems.