CSS Grid Workshop

beyond tellerrand Munich

Melanie Richards / [Melanie.Richards@microsoft.com](mailto:Melanie.Richards@microsoft.com)

[aka.ms/bt-grid-questions](http://aka.ms/bt-grid-questions) (ask anonymously, will be unmonitored after workshop)

Contents

[Quick-Reference Sheet 2](#_Toc503349758)

[Defining grid tracks 2](#_Toc503349759)

[Handy track sizing values 4](#_Toc503349760)

[Placing and sizing grid items 6](#_Toc503349761)

[Aligning grid items within their “slots” 8](#_Toc503349762)

[Aligning the grid as a whole 8](#_Toc503349763)

[Property values for item alignment 9](#_Toc503349764)

[Property values for grid alignment 10](#_Toc503349765)

[Workshop Projects 11](#_Toc503349766)

[Grid FAQ 12](#_Toc503349767)

[Resources 14](#_Toc503349768)

[Additional Projects 15](#_Toc503349769)

Standards vocab:

* **CSSWG:** W3C CSS Working Group, which designs CSS “standards” so that new features are as "interoperable" (work the same browser to browser) as possible
* **Specification:** the standardized design of a CSS feature

# Quick-Reference Sheet

**Grid is a 4-step process:**

1. Declare grid context on a parent element with display: grid or inline-grid
2. Define the grid tracks
3. Place, size, and align direct-child elements (grid items) on the grid
4. Size and align the grid itself

## Defining grid tracks

Follow links in the “approach” column to see docs / full syntax on MDN.  
In shorthands, **rows always come before columns**.

|  |  |
| --- | --- |
| **Approach** | **Examples** |
| Explicit tracks  ([grid-template-rows](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-rows), [-columns](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-columns);  [grid-template](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template)) | /\* Three rows of equal width; columns of 6em, then one column 2x as wide as the next \*/ grid-template-rows: repeat(3, 1fr); grid-template-columns: 6em 2fr 1fr;  /\* Same grid in the grid-template shorthand \*/  grid-template: repeat(3, 1fr) / 6em 2fr 1fr; |
| Explicit tracks w/  named lines | grid-template-rows: **[top header]** minmax(10em, auto) **[aside-start main]** auto **[afterward]** auto **[aside-end footer]** minmax(10em, auto) **[bottom]**;  /\* This child would start at the “afterword” and span 1 line \*/  grid-row: afterword;  /\* This child would span from “aside-start” line to “aside-end” line \*/  grid-row: sidebar; |

*(continued on next page)*

Defining grid tracks, contd.

|  |  |
| --- | --- |
| [Grid areas](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-areas)  Must have defined explicit tracks before defining grid areas. Can use a period as a spacer for slots where you don’t really need a name. | /\* A grid with 3 columns and 4 rows defined \*/  grid-template-areas: “logo nav nav”  “side side main”  “side side social”  “foot foot foot”;  /\* Child placement example \*/  grid-area: side; |
| Auto tracks  ([grid-auto-rows](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-rows),  [-columns](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-columns))  Can be used in conjunction with explicit tracks (Ex: can define 3 explicit rows with grid-template-rows, and then an auto pattern for an unknown number of additional rows) | /\* A couple rows of specific measurements, followed by unknown #s of rows of equal height \*/  grid-template-rows: 10em auto;  grid-auto-rows: 1fr; |
| [Grid gap](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-gap)  Used in conjunction with the track definitions listed above. Avoid using %s in row gaps; due to disagreement as to how these should resolve, there [isn't currently cross-browser compability](https://jsfiddle.net/zk9dtnzk/2/) (FF resolves as % of the grid item, others just don't honor them) | /\* Rows have a 4em gap, columns have a 3% gap \*/  grid-gap: 4em 3%;  /\* Same thing in longhands \*/  grid-row-gap: 4em;  grid-column-gap: 3%;  /\* 20px gaps between rows and columns \*/  grid-gap: 20px; |

## Handy track sizing values

|  |  |  |
| --- | --- | --- |
| **Value** | **What it is** | **Examples** |
| fr unit | Tracks of flexible width, which tell the Grid how to distribute extra space; proportional | /\* The last column will be sized to 20em, and the remaining space will be distributed between the first two columns (twice as much in the 2fr column as in the 1fr column) \*/  grid-template-columns: 2fr 1fr 20em; |
| max-content | Each element in a track has a natural measurement it would take given infinite space; max-content is equal to the largest of these measurements | /\* Used as a track measurement \*/  grid-template-columns: max-content 1fr 20em; |
| min-content | Each element in a track has a smallest measurement that would prevent it from overflowing; min-content is equal to the largest of these measurements | /\* Used as a track measurement \*/  grid-template-columns: min-content 1fr 20em; |
| fit-content() | Computes to the max-content value (see above), or the given measurement, whichever is smaller. Takes a “length” (px, em, etc) or a percentage. | /\* First track will be as wide as necessary to fit the grid items in the track, but with a max of 20em \*/  grid-template-columns: fit-content(20em) 1fr 20em; |

*(continued on next page)*

Handy track sizing values, contd.

|  |  |  |
| --- | --- | --- |
| minmax([min], [max]) | A functional value that sizes a track to at least [min] and at most [max]. You can use fr units, max-content, min-content, lengths (px, em, etc), or percentages within it. If max computes to less than min, it is ignored and the measurement, at minimum, is the min given. | /\* Used as a track measurement \*/  grid-template-columns: min-max(min-content, 2fr) 1fr 20em; |
| auto-fill | Fits as many tracks as possible within the axis, preserving empty tracks (those without Grid items) | See repeat() notation |
| auto-fit | Fits as many tracks as possible within the axis, but *collapses* empty tracks | See repeat() notation |
| repeat() | Repeats a sequence of tracks (rows or columns) | /\* Creates 12 columns alternating between 10 characters, and an equal distribution of  remaining space \*/  grid-template-columns:  repeat(12, 10ch 1fr);  /\* Creates as many cols as will fit of 20em wide, then distributes any extra space evenly between these cols \*/  grid-template-columns:  repeat(auto-fill, minmax(20em, 1fr)); |

## Placing and sizing grid items

|  |  |
| --- | --- |
| **Approach** | **Examples** |
| [Auto flow](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-flow)  By default, grid items flow into the grid row-wise. You can manipulate this with grid-auto-flow, set on the grid parent. | /\* Fills each row, adding new rows as needed (also the default behavior) \*/  grid-auto-flow: row; |
| /\* Fills each column, adding new columns as needed \*/  grid-auto-flow: column; |
| /\* Tries to fill smaller slots in the grid as items which fit in them come up (this may result in grid items appearing out of order; this is not a perfect 1:1 to a “masonry” layout, see FAQ) \*/  grid-auto-flow: dense; |
| /\* Dense behavior can be used in the column direction \*/  grid-auto-flow: column dense; |
| [grid-row-start](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-row-start),  [grid-row-end](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-row-end),  [grid-column-start](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column-start),  [grid-column-end](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column-end)  By setting where a grid item starts and ends, you’re essentially placing and sizing it at the same time | /\* Starts grid item on 4th column line \*/  grid-column-start: 4; |
| /\* Starts grid item on line named “foo” or “foo-start” (in second case, spans to "foo-end" line if available \*/  grid-column-start: foo; |
| /\* Starts grid item on fifth line named “col” \*/  grid-column-start: 5 col; |
| /\* Span item to 2 grid lines from the column-start value \*/  grid-column-end: span 2; |
| /\* Span item to next grid line named “foo” \*/  grid-column-end: span foo; |
| /\* Span item to the last line in the grid \*/  grid-column-end: -1; |

*(continued on next page)*

Placing and sizing grid items, contd.

|  |  |
| --- | --- |
| [grid-row](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-row),  [grid-column](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column)  Shorthands for -start and  -end | /\* Places grid item on 4th row line, spans to 7th line \*/  grid-row: 4 / 7; |
| /\* Places grid item on 3rd row line, spans to line named “colophon” or “colophon-end” \*/  grid-row: 3 / span colophon; |
| [grid-area](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-area) | /\* Places item in grid area name “foo” \*/  grid-area: foo; |
| /\* Places item in next available grid area \*/  grid-area: auto; |
| /\* Places item on 2nd row line, 4th column line \*/  grid-area: 2 / 4; |
| /\* Item spans 2nd row line to 5th row line; 4th to 7th column line \*/  grid-area: 2 / 4 / 5 / 7; |
| /\* Item spans in row direction from the 2nd line to the line named “foo”; spans in column direction from the 4th line to the line named “bar” \*/  grid-area: 2 / 4 / foo / bar; |

## Aligning grid items within their “slots”

These properties align the entire grid item (say, a div) within the slot created by the intersection of rows and columns. They don’t align the child elements within the div), you'd want to use flexbox or a nested grid for that. For the accepted values of these properties, see the “Property values for item alignment” table.

|  |  |
| --- | --- |
| **Property** | **Usage** |
| justify-self | Used on grid item to control the alignment in the “inline” axis (in Latin writing systems, the inline axis is horizontal) |
| justify-items | Used on grid parent to control the alignment of grid items within their slots, in the “inline” axis |
| align-self | Used on grid item to control the alignment in the “block” axis (in Latin writing systems, the block axis is vertical) |
| align-items | Used on grid parent to control the alignment of grid items within their slots, in the “block” axis |

## Aligning the grid as a whole

After track sizing and item placement is done, you can align the rows and columns within the grid itself. This tells the tracks how to distribute themselves in any leftover space within the grid parent. See “Property values for grid alignment” table.

|  |  |
| --- | --- |
| **Property** | **Usage** |
| justify-content | Used on grid parent to control the alignment of the grid as a whole in the “inline” axis (in Latin writing systems, the inline axis is horizontal) |
| align-content | Used on grid parent to control the alignment of the grid as a whole in the “block” axis (in Latin writing systems, the block axis is vertical) |

## Property values for item alignment

|  |  |
| --- | --- |
| **Value** | **What it Does** |
| auto | Inherits from the justify/align-items property on the grid; otherwise behaves as “normal” |
| normal | Basically behaves as “stretch”, unless the element has an intrinsic ratio (like an image), in which case it behaves as “start” |
| center | Aligns to physical center |
| left | Aligns to physical left |
| right | Aligns to physical right |
| start | Aligns to the “start” edge of the alignment container. This value can change based on the writing mode of the parent. |
| end | Aligns to the “end” edge of the alignment container. This value can change based on the writing mode of the parent. |
| self-start | Aligns to the “start” edge of the item. This value can change based on the item’s writing mode. |
| self-end | Aligns to the “end” edge of the item. This value can change based on the item’s writing mode. |
| stretch | Fills the space in this axis |

## Property values for grid alignment

|  |  |
| --- | --- |
| **Value** | **What it Does** |
| normal | Basically behaves as “stretch”, unless the element has an instrinsic ratio (like an image), in which case it behaves as “start” |
| center | Aligns to physical center |
| left | Aligns to physical left |
| right | Aligns to physical right |
| start | Aligns to the “start” edge of the grid element. This value can change based on the writing mode. |
| end | Aligns to the “end” edge of the grid element. This value can change based on the writing mode. |
| stretch | Fills the space in this axis |
| space-between | Evenly distributes extra space between tracks |
| space-around | Evenly distributes extra space, placing it before the first track and after the last track |
| space-evenly | Evenly distributes extra space between tracks, before the first track, and after the last track |
| safe | A keyword which can be added to another of these property values. If a particular item were to overflow the grid container, then it would be treated as “start” |
| unsafe | A keyword which can be added to another of these property values, which tells the grid to honor the specified alignment, even if the item would overflow. This is not currently needed as the “unsafe” behavior is default, but would be needed in the future if the CSSWG decides to have the “smart” behavior be the default. |

# Workshop Projects

* Recreate a print layout   
  A modern-classic CSS Grid challenge! Choose one of the print layouts by [Matt Willey](http://mattwilley.co.uk) in /project-materials/project-a/, and recreate it in CSS the best you can. Links for sample content to us as placeholders: [text](http://lipsum.org), [images](https://placeholder.com/), [photos](http://pexels.com). If you share your rendition of the layout elsewhere, please be sure to include proper attribution (name and link) for the designer.
* Blog redux   
  Reimagine the [W3C blog index](https://www.w3.org/blog/) with CSS Grid. You may bring in new assets, but at least keep: the W3C logo and some type of header, the post previews with all info in tact, and the content from the right-hand sidebar. If you have time, try reimaging an individual blog post, as well. For bonus points, incorporate overlapping grid items somewhere into your design!
* Recreate an old project   
  Redo a layout or two on a previous project with CSS Grid. How might you be able to simplify the markup? What becomes much easier with Grid?
* Make a classic arcade game UI   
  Create one frame of a game in the style of a classic video game, using sprites from a site like [The Spriter’s Resource](https://www.spriters-resource.com/), or your own assets. How might you position pieces of the environment with Grid? Some ideas for games to emulate: Pacman, a platformer (like Super Mario Bros.), Frogger, Bejeweled, Mahjongg, brick breaker (with your own unique layouts!)

If you’d like your project included in a summary blog post, send a Codepen, Github, or other public link to [Melanie.Richards@microsoft.com](mailto:Melanie.Richards@microsoft.com)!

# Grid FAQ

Can I align child elements of Grid items to the main Grid?

Not yet, but the CSSWG knows this is important to web devs and is working on a solution. The feature that will solve this is called “subgrid”; it was dropped from the [first iteration of the CSS Grid specification](http://www.w3.org/TR/css-grid-1/), because the working group didn’t think the current design of subgrid adequately solved the problem. Back to the drawing board!

Can I style Grid tracks?

Grid tracks are not style-able, but there are some ways you can get around it. I used a single linear gradient background to create the illusion of styled grid tracks—sand, water, sky—in my team’s [grid demo](https://developer.microsoft.com/en-us/microsoft-edge/testdrive/demos/css-grid/) *(designed by Stephanie Drescher, mostly built by Greg Whitworth)*. See also: [“Faux Grid Tracks”](http://alistapart.com/article/faux-grid-tracks) by Eric Meyer.

Can I make a Masonry-style layout with CSS Grid?

The current design of Grid is such that a “Masonry” layout (a la Pinterest boards) is not possible without Javascript. This is a little confusing because there is a property/value set that seems like it would do this: grid-auto-flow: dense. What this feature does, however, is try to fill empty gaps in rows by cherry-picking grid items out of order. What you actually want with a Masonry layout is for everything to be in order and to fill empty vertical gaps of unknown height. The problem is that Grid and auto-flowing items deals with placing items on whole tracks, whereas with Masonry layouts, it’s like you need Grid to place items on partial tracks, create new auto-sized rows based on only the fraction of the content that was not placed on the previous partial track, etc. It’s a known, common desire that needs a little thinking to do more performantly in the browser engine than can be done by an web dev in Javascript.

# Grid FAQ, contd.

How can I set different gaps between tracks?

The way grid-gap is currently designed is that column gaps all take one measurement, and row gaps all take one measurement. A way to force a larger gap in one place would be to create an empty grid track (say, of 2em wide, if you want a 6em gap instead of the 4em gap you have elsewhere), and then just not place any items in that track. [There is a discussion going](https://github.com/w3c/csswg-drafts/issues/1659) on how to better solve this in future levels of the CSS Grid specification. Feel free to jump in there to make a suggestion or upvote a suggestion that makes sense to you!

# Resources

Reference Materials

* [Grid Specification](http://www.w3.org/TR/css-grid-1/)
* [Alignment Specification](https://www.w3.org/TR/css-align-3/)
* [Docs for original -ms prefixed Grid](https://msdn.microsoft.com/en-us/library/hh673533(v=vs.85).aspx)
* [CSS Grid support](https://caniuse.com/#feat=css-grid)
* [Test Microsoft Edge on BrowserStack for free](https://www.browserstack.com/test-on-microsoft-edge-browser)
* [Grid by Example](https://gridbyexample.com/): lots of examples and patterns with fallbacks, by Rachel Andrew
* [Mozilla’s CSS Grid Playground](https://mozilladevelopers.github.io/playground/css-grid)

Explorations of Grid

* [Labs by Jen Simmons](http://labs.jensimmons.com/): fun with other layout techniques, too
* [More ideas for Grid projects](https://hiddedevries.nl/en/blog/2017-12-06-what-to-use-grid-layout-for-some-ideas)
* [Realizing common layouts using CSS Grid Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Realizing_common_layouts_using_CSS_Grid_Layout)
* [Design Systems and CSS Grid](https://24ways.org/2017/design-systems-and-css-grid/)
* [Breaking Out With CSS Grid Layout](https://cloudfour.com/thinks/breaking-out-with-css-grid-layout/) ([explainer](https://rachelandrew.co.uk/archives/2017/06/01/breaking-out-with-css-grid-explained/))
* [Using CSS Grid as a mask](https://codepen.io/andybarefoot/pen/wrXvLj)
* Share your experiments on [Codepen](https://codepen.io/search/pens?q=css+grid&limit=all&type=type-pens)

# Additional Projects

Make a block-style dashboard

Think about something that interests you, and what a dashboard related to that interest might look like. For example, a coin collector might have an overview of their recent inventory, charts on profit margins, data trends in coin values, articles about coin collecting news, etc.  
  
Things you might consider:

* How do the blocks in this dashboard adjust to variable amounts of content?
* How might you use Grid to make this dashboard responsive to viewport measurements, and keep the blocks in a logical order?
* Where might you be able to create symmetry throughout the design?

Make a collage

Embrace the overlapping capabilities of CSS Grid to make some collages using images, text, and CSS effects!

Recreate your favorite tabletop game

If it's a board game, recreate the board. If you like card games, is there a certain structure to how the cards get laid out on the table? Include that in your design.

Design a building

Create a blue print of a floor plan, or the façade of a building, using only CSS.

Replicate a news site

Take the index of a busy news site, like nytimes.com or faz.net, and recreate it with as few wrapper elements as possible.