

www.agilityio.com

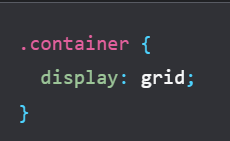
Grid Notes

**Chapter 0: Overview**

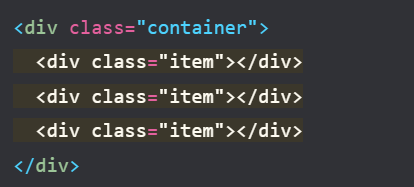
* **What can you do with grid ?**
* A grid can be defined with rows and columns. You can choose how to size these row and column tracks or they can react to the size of the content.
* Direct children of the grid container will be automatically placed onto this grid.
* Or, you can place the items in the precise location that you want.
* Lines and areas on the grid can be named to make placement easier.
* Spare space in the grid container can be distributed between the tracks.
* Grid items can be aligned within their area.

**Chapter 1: Grid terminology**

* **Grid lines**
* A grid is made up of lines, which run horizontally and vertically. If your grid has four columns, it will have five column lines including the one after the last column.
* Lines are numbered starting from 1, with the numbering following the writing mode and script direction of the component
* **Grid tracks**
* A track is the space between two grid lines. A row track is between two row lines and a column track between two column lines.
* **Grid cell**
* A grid cell is the smallest space on a grid defined by the intersection of row and column tracks. It's just like a table cell or a cell in a spreadsheet. If you define a grid and don't place any of the items they will automatically be laid out one item into each defined grid cell.
* **Grid area**
* Several grid cells together. Grid areas are created by causing an item to span over multiple tracks.
* **Gaps**
* A gutter or alley between tracks. For sizing purposes these act like a regular track. You can't place content into a gap but you can span grid items across it.
* **Grid container**
* the HTML element which has display: grid applied, and therefore creates a new grid formatting context for the direct children.

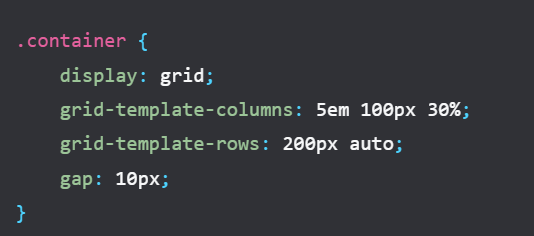


* **Grid item**
* A grid item is an item which is a direct child of the grid container.

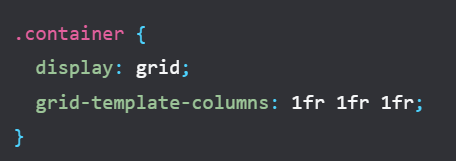


**Chapter 2: Rows and columns**

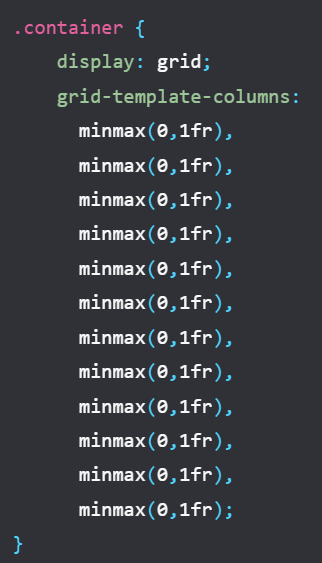
* To create a basic grid you can define a grid with three column tracks, two row tracks and a 10 pixel gap between the tracks as follows.



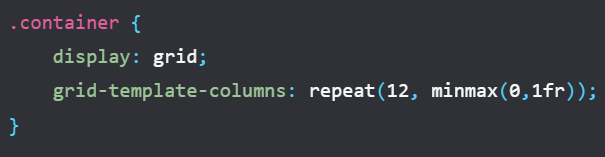
* If the element with a class of .container has child items, they will immediately lay out on this grid
* **Intrinsic sizing keywords**
* In addition to the length and percentage dimensions as described in the section on sizing units, grid tracks can use intrinsic sizing keywords. These keywords are defined in the Box Sizing specification and add additional methods of sizing boxes in CSS, not just grid tracks
* The min-content keyword will make a track as small as it can be without the track content overflowing. Changing the example grid layout to have three column tracks all at min-content size will mean they become as narrow as the longest word in the track.
* The max-content keyword has the opposite effect. The track will become as wide enough for all of the content to display in one long unbroken string. This might cause overflows as the string will not wrap.
* The fit-content() function acts like max-content at first. However, once the track reaches the size that you pass into the function, the content starts to wrap. So fit-content(10em) will create a track that is less than 10em, if the max-content size is less than 10em, but never larger than 10em.
* **The fr unit**
* The fr unit is flexible in length, describing a share of the available space in the grid container.
* The fr unit works similarly to using flex: auto in flexbox. It distributes space after the items have been laid out. Therefore to have three columns which all get the same share of available space:\



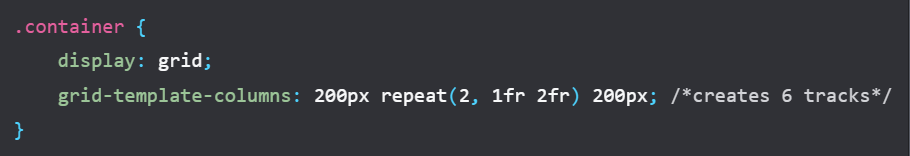
* As the fr unit shares out available space, it can be combined with a fixed size gap or fixed size tracks.
* **The minmax() function**
* This function means that you can set a minimum and a maximum size for a track.
* If we take the example of the fr unit above which distributes remaining space, it could be written out using minmax() as minmax(auto, 1fr). Grid looks at the intrinsic size of the content, then distributes available space after giving the content enough room. This means that you might not get tracks that each have an equal share of all space available in the grid container.
* **repeat() notation**
* If you want to create a 12 column track grid with equal columns, you could use the following CSS.



* Or, you could write it out using repeat():



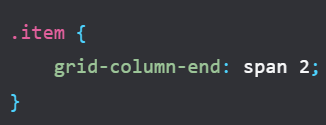
* The repeat() function can be used to repeat any section of your track listing. For example you can repeat a pattern of tracks. You can also have some regular tracks and a repeating section.



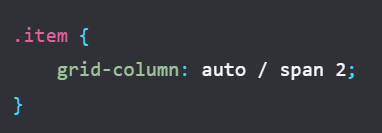
* **auto-fill and auto-fit**
* There is a subtle difference between auto-fill and auto-fit. Using the auto-fill keyword you can see that empty tracks have been created. Change the keyword to auto-fit and the tracks collapse down to 0 size. This means that the flexible tracks now grow to consume the space
* The auto-fill and auto-fit keywords otherwise act in exactly the same way. There is no difference between them once the first track is filled.

**Chapter 3: Auto-placement**

* **Placing items in columns**
* The default behavior of grid layout is to place items along the rows. You can instead cause the items to place into columns using grid-auto-flow: column. You need to define row tracks otherwise the items will create intrinsic column tracks, and layout out all in one long row.
* A row always runs in the direction a sentence runs in the writing mode of the document or component. In the next demo you can change mode the value of grid-auto-flow and the writing-mode property.
* **Spanning tracks**
* You can cause some or all of the items in an auto-placed layout to span more than one track. Use the span keyword plus the number of lines to span as a value for grid-column-end or grid-row-end.



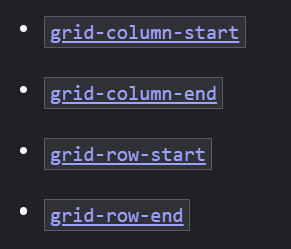
* As you have not specified a grid-column-start, this uses the initial value of auto and is placed according to the auto-placement rules. You can also specify the same thing using the shorthand grid-column:



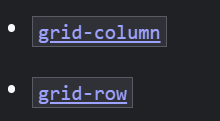
* **Filling gaps**
* The items will be placed according to the order they are in the source, or any modification with the order property. If there is not enough space to fit an item, grid will leave a gap and move to the next track.

**Chapter 4: Placing items**

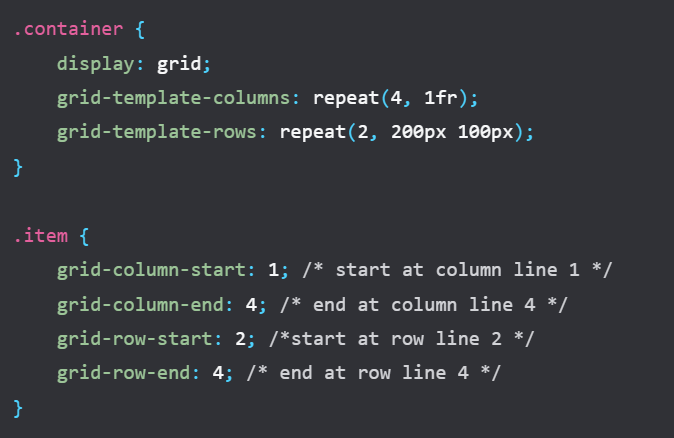
* The CSS Grid Layout is based on a grid of numbered lines. The simplest way to place things onto the grid is to place them from one line to another.
* The properties that you can use to place items by line number are:



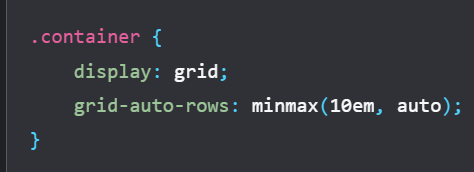
* They have shorthands which allow you to set both start and end lines at once:



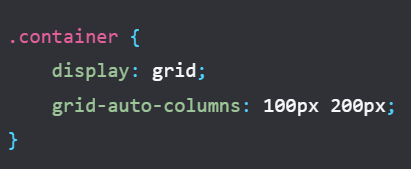
* To place your item set the start and end lines of the grid area that it should be placed into.



* **Stacking items**
* Using line-based positioning you can place items into the same cell of the grid. This means that you can stack items, or cause one item to partly overlap another. Items which come later in the source will be displayed on top of items that come earlier. You can change this stacking order using z-index just as with positioned items.
* **Negative line numbers**
* When you create a grid using grid-template-rows and grid-template-columns you create what is known as the explicit grid. This is a grid that you have defined and given size to the tracks.
* **Sizing implicit tracks**
* The tracks created in the implicit grid will be auto-sized by default. However if you want to control the sizing of the rows, use the grid-auto-rows property, and for columns grid-auto-columns.
* Example:
* To create all implicit rows at a minimum size of 10em and a maximum size of auto:

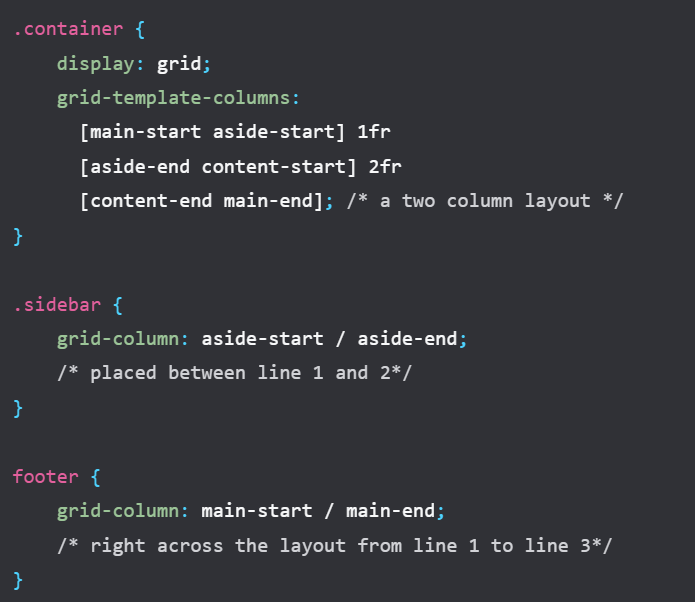


* To create implicit columns with a pattern of 100px and 200px wide tracks. In this case the first implicit column will be 100px, the second 200px, the third 100px, and so on.



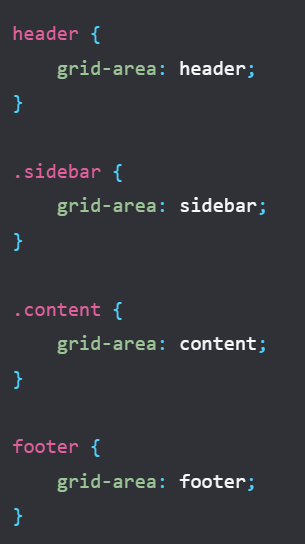
**Chapter 5: Named grid lines**

* It can make it easier to place items into a layout if the lines have a name rather than a number. You can name any line on your grid by adding a name of your choosing between square brackets. Multiple names can be added, separated by a space inside the same brackets. Once you have named lines they can be used instead of the numbers.

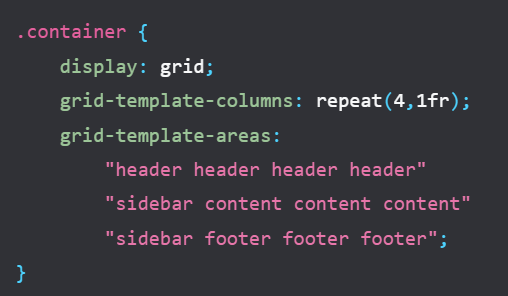


**Chapter 6: Grid Template Areas**

* To start, give the direct children of your grid container a name using the grid-area property:



* The name can be anything you like other than the keywords auto and span. Once all of your items are named, use the grid-template-areas property to define which grid cells each item will span. Each row is defined within quotes.

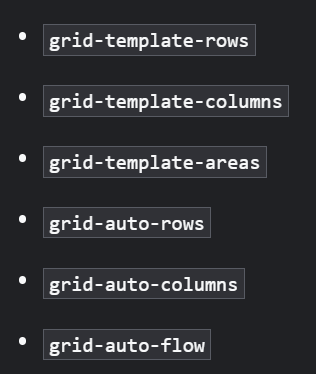


**Chapter 7: Shorthand properties**

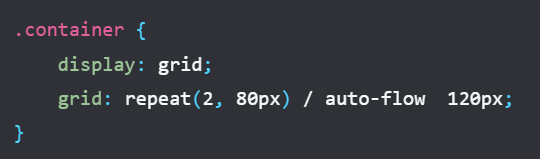
* There are two shorthand properties which allow you to set many of the grid properties in one go. These can look a little confusing until you break down exactly how they go together
* **grid-template**
* The grid-template property is a shorthand for grid-template-rows, grid-template-columns and grid-template-areas. The rows are defined first, along with the value of grid-template-areas. Column sizing is added after a /.



* **grid property**
* The grid shorthand can be used in exactly the same way as the grid-template shorthand. When used in this way it will reset the other grid properties that it accepts to their initial values. The full set being:



* You can alternately use this shorthand to define how the implicit grid behaves, for example:



**Chapter 8: Alignment**

* The properties which begin with align- are used on the block axis, the direction in which blocks are laid out in your writing mode.
* [justify-content](https://developer.mozilla.org/docs/Web/CSS/justify-content) and [align-content](https://developer.mozilla.org/docs/Web/CSS/align-content): distribute additional space in the grid container around or between tracks.
* [justify-self](https://developer.mozilla.org/docs/Web/CSS/justify-self) and [align-self](https://developer.mozilla.org/docs/Web/CSS/align-self): are applied to a grid item to move it around inside the grid area it is placed in.
* [justify-items](https://developer.mozilla.org/docs/Web/CSS/justify-items) and [align-items](https://developer.mozilla.org/docs/Web/CSS/align-items): are applied to the grid container to set all of the justify-self properties on the items.
* **Distributing extra space**
* As with flexbox, these properties will only work if there is additional space to distribute. If your grid tracks neatly fill the container then there will be no space to share out.
* **Moving content around**

Items with a background color appear to completely fill the grid area they are placed in, because the initial value for justify-self and align-self is stretch.