Extra Notes - CSS - Grid Layout Guide

A general guide to CSS Grid Layout.

Contents

intro

Grid layout with CSS is useful for structure and organisation in a HTML page. Its usage may be considered akin to previous layout options with tables, thereby enabling a developer to add columns and rows to a page.

However, a grid layout enables more complex and interesting layout options, including overlap and layers.

general concepts and usage

A grid may be composed of rows and columns, thereby forming an intersecting set of horizontal and vertical lines.

Elements may be added to the grid with reference to this structured layout.

Grid layout in CSS includes the following general features,

- additional tracks for content
  - option to create more columns and rows as needed to fit dynamic content
- control of alignment
  - align a grid area or overall grid
- control of overlapping content
  - permit partial overlap of content
  - an item may overlap a grid cell or area
- placement of items - explicit and implicit
  - precise location of elements &c.
  - use line numbers, names, grid areas &c.
- variable track sizes - fixed and flexible
  - e.g. specify pixel size for track sizes or use flexible sizes with percentages or new `fr` unit

grid container

We may define an element as a grid container using `display: grid` or `display: inline-grid`.

Then, any children of this element become *grid items*.

e.g.

```
.wrapper {
  display: grid;
}
```

We may also define other, child nodes as a `grid` container. Any direct child nodes to a grid container are now defined as grid items.

what is a grid track?

In a grid, rows and columns are defined with `grid-template-columns` and `grid-template-rows` properties respectively.

In effect, these define *grid tracks*.

As MDN notes,

- a *grid track* is the space between any two lines on the grid.

(https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout)

So, we can create both row and column tracks. e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```

The `wrapper` class now includes three defined columns of width 200px, thereby creating three tracks.

n.b. a track may be defined using any valid length unit, not just `px`.

`fr` unit for tracks

CSS Grid now introduces an additional length unit for tracks, `fr`.

`fr` unit represents fractions of the space available in the current grid container.

e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

However, we may also apportion various space to tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 2fr 1fr 1fr;
}
```

This still creates three tracks in the grid, but the overall space effectively now occupies four parts. Two parts for `2fr`, and one part each for the remaining two `1fr`.

We may also be specific in this sub-division of parts in tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;
}
```

The first track will occupy a width of `200px`, whilst the remaining two tracks will each occupy 1 fraction unit.

`repeat()` notation for `fr`

For larger, repetitive grids, it's easier to use the `repeat()` notation to define multiple instances of the same track.

e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
}
```

This will now create four separate tracks, each defined as `1fr` unit's width.

Such `repeat()` notation may also be used as part of the track definition. e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px repeat(4, 1fr) 100px;
}
```

This example will create one track of `200px` width, then four tracks of `1fr` width, and finally a single track of `100px` width.

`repeat()` may also be used with multiple track definitions, thereby repeating multiple times

e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
}
```

This will now create eight tracks, the first four of width `1fr` and the remaining four of `2fr`.

implicit and explicit grid creation

In the above examples, we simply define tracks for the columns, and CSS grid will then apportion content to required rows.

However, we may also define an explicit grid of columns and rows, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: 150px;
}
```

This slightly modifies an implicit grid to ensure each row is `200px` tall.

track sizing

A grid may require tracks with a minimum size, and the option to expand to fit dynamic content.

This might be as simple as ensuring that a track does not collapse below a certain height or width, and that it has the option to expand as necessary for the content.

Grid provides a `minmax()` function, which we may use with rows, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: minmax(150px, auto);
}
```

This example ensures that each rows will occupy a minimum of `150px` in height, and still be able to stretch to contain the tallest content.

However, the whole row will expand to meet the `auto` height requirements, thereby affecting each track in the row.

grid lines

A grid is defined using *tracks*, and not lines in the grid.

However, the created grid also helps us with positioning by providing numbered lines.

e.g. in a three column, two row grid we have the following,

- four lines for the three vertical columns
- three lines for the two horizontal rows

Such lines start at the left for columns, and at the top for rows.

n.b. line numbers start relative to written script, e.g left to right for western, right to left for arabic...

positioning against lines

When we place an item in a grid, we use these lines for positioning, and not the tracks.

This is reflected in the usage of `grid-column-start`, `grid-column-end`, `grid-row-start`, and `grid-row-end` properties.

In effect, items in the grid may be position from one line to another, e.g. column line 1 to column line 3.

n.b. default span for an item in a grid is one track, e.g. define column start and no end - default span will be one track...

e.g.

```
.content1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

grid cell

A *cell* is the smallest unit on the defined grid layout.

In effect, it is conceptually the same as a cell in a standard table.

So, as content is added to the grid, it will be stored in one cell.

grid area

However, we may also store content in multiple cells, thereby creating *grid areas*.

Grid areas must be rectangular in shape.

e.g. a grid area may span multiple row and column tracks for the required content.

add some gutters

Gutters may be created using the gap property for either column or row, `column-gap` and `row-gap`.

e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
  column-gap: 5px;
  row-gap: 10px;
}
```

n.b. any space used for gaps will be determined prior to assigned space for `fr` tracks.

Demos

- grid basic - page zones and groups
- grid basic - article style page
- grid layout - articles with scroll