

CSS GRID
“The future depends on what you do today.”

Bruxelles Formation | CEPEGRA | Olivier Céréssia | 2018

webdesigner.cepegra-labs.be

# Préambule

Flexbox vient à peine de s’installer confortablement dans nos habitudes qu’il est déjà question de remettre tout le système de positionnement en cause.

Même sl Flex semble convenir à beaucoup de monde, une nouvelle façon d’aborder la disposition de nos différents éléments sur nos pages est en train de débarquer sur le marché : la disposition en grille (CSS Grid Layout).

Même si elle ne porte pas ce nom, cette approche serait encore bien plus flexible (et permettrait des choses bien plus créatives) que … Flex ☺

Ça ne veut absolument pas dire que Flex est à jeter à la poubelle. Ça veut juste dire qu’on a une arme supplémentaire pour pouvoir positionner les choses en CSS.

Notre boite à outil « moderne » devrait être constituée des propriétés suivantes : grid, flexbox, multicolumn, flow, shapes, object fit, viewport units, etc.

Souvenez-vous (en fait vous ne vous en souvenez peut-être même pas), les positionnements en float et inline-block étaient tellement plus compliqués et moins logiques que ces solutions émergentes. Il y a fort à parier que Flex et Grid seront les solutions qui seront les plus populaires dès demain.

Voyons ensemble comment prendre en main ces fameux Grid Layouts.

Table des matières

[Préambule 2](#_Toc348513155)

[html5 : la révélation Erreur ! Signet non défini.](#_Toc348513156)

[ Nouvelle grammaire **Erreur ! Signet non défini.**](#_Toc348513157)

[Simplification du !DOCTYPE **Erreur ! Signet non défini.**](#_Toc348513158)

[Une syntaxe permissive **Erreur ! Signet non défini.**](#_Toc348513159)

[ Nouvelles balises sémantiques **Erreur ! Signet non défini.**](#_Toc348513160)

[<header> et <hgroup> **Erreur ! Signet non défini.**](#_Toc348513161)

[<footer> **Erreur ! Signet non défini.**](#_Toc348513162)

[<nav> **Erreur ! Signet non défini.**](#_Toc348513163)

[<aside> **Erreur ! Signet non défini.**](#_Toc348513164)

[<section> **Erreur ! Signet non défini.**](#_Toc348513165)

[<article> **Erreur ! Signet non défini.**](#_Toc348513166)

[<figure> **Erreur ! Signet non défini.**](#_Toc348513167)

[ Nouvelles balises multimédia **Erreur ! Signet non défini.**](#_Toc348513168)

[<audio> **Erreur ! Signet non défini.**](#_Toc348513169)

[<video> **Erreur ! Signet non défini.**](#_Toc348513170)

[ Des formulaires améliorés **Erreur ! Signet non défini.**](#_Toc348513171)

[De nouveaux types de champs **Erreur ! Signet non défini.**](#_Toc348513172)

[Range **Erreur ! Signet non défini.**](#_Toc348513173)

[date, datetime, month, week, time **Erreur ! Signet non défini.**](#_Toc348513174)

[search **Erreur ! Signet non défini.**](#_Toc348513175)

[De nouveaux attributs intelligents **Erreur ! Signet non défini.**](#_Toc348513176)

[Placeholder **Erreur ! Signet non défini.**](#_Toc348513177)

[Autofocus **Erreur ! Signet non défini.**](#_Toc348513178)

[Autocomplete **Erreur ! Signet non défini.**](#_Toc348513179)

[Required **Erreur ! Signet non défini.**](#_Toc348513180)

[ Des attributs puissants **Erreur ! Signet non défini.**](#_Toc348513181)

[Draggable **Erreur ! Signet non défini.**](#_Toc348513182)

[Contenteditable **Erreur ! Signet non défini.**](#_Toc348513183)

[ Vers un HTML5 « transitionnel » ? **Erreur ! Signet non défini.**](#_Toc348513184)

# Les 25 nouvelles propriétés CSS

À placer sur les parents

* display : grid | inline-grid | subgrid ;
* grid-gap : <grid-row-gap> <grid-column-gap>;
* grid-column-gap : <line-size>;
* grid-row-gap : <line-size>;
* grid-template-columns : <track-size> … | <line-name> <track-size> ;
* grid-template-rows : <track-size> … | <line-name> <track-size> ;
* grid-auto-flow : row | column | row dense | column dense ;
* grid-template-areas : <grid-area-name> | . | none | … ;
* grid-template : grid-template: none | subgrid | <grid-template-rows> / <grid-template-columns>;
* justify-items : start | end | center | stretch;
* align-items : start | end | center | stretch;
* justify-content : start | end | center | stretch | space-around | space-between | space-evenly;
* align-content : start | end | center | stretch | space-around | space-between | space-evenly;
* grid-auto-columns : <track-size> ...;
* grid-auto-rows : <track-size> ...;
* grid : none | <grid-template-rows> / <grid-template-columns> | <grid-auto-flow> [<grid-auto-rows> [/ <grid-auto-columns>]];

À placer sur les enfants

* grid-column-start : <number> | <name> | span <number> | span <name> | auto ;
* grid-column-end : <number> | <name> | span <number> | span <name> | auto ;
* grid-row-start : <number> | <name> | span <number> | span <name> | auto ;
* grid-row-end : <number> | <name> | span <number> | span <name> | auto ;
* grid-column : <start-line> / <end-line> | <start-line> / span <value>;
* grid-row : <start-line> / <end-line> | <start-line> / span <value>;
* grid-area : <name> | <row-start> / <column-start> / <row-end> / <column-end>;
* justify-self : start | end | center | stretch;
* align-self : start | end | center | stretch;

# Un point sur le support actuel

S’il est bien un point capital à vérifier avant de commencer à utiliser css grid en production, c’est sa compatibilité.

Voyons ce qu’en dit caniuse.com :



Un support plutôt homogène sur les navigateurs modernes alors que IE le supporte de façon partielle en faisant référence à une version antérieure de la spécification. Tout n’est pas perdu donc.

Doit-on dès lors utiliser CSS grid en production ?

La question mérite d’être posée d’autant que Flex fait sensiblement la même chose avec un support meilleur sur IE (malgré quelques bugs).

Mais la réponse devrait être oui. Pour plein de raisons :

* CSS Grid, c’est l’avenir du positionnement sur le Web
* Utiliser CSS Grid et le rendre universel, c’est notre rôle, nous sommes les fers-de-lance de ce genre de nouveautés et nous nous devons de les utiliser quand elles sont déjà majoritairement utilisables comme c’est le cas avec Css Grid
* C’est quand même déjà super bien supporté par les navigateurs modernes.
* Cela permet une flexibilité et des montages structurels difficiles avec Flex
* La majorité des propriétés de Css Grid sont faciles à prendre en main, contrairement à certaines de Flex qui sont plus obscures (flex-grow, flex-shrink, flex-basis).

Donc oui, utilisez CSS Grid et commencez à répandre son utilisation partout où vous allez, et faites comprendre aux personnes qu’il est bon de travailler avec les nouveaux outils mis à notre disposition.

Si, en revanche, votre projet doit absolument être compatible sur IE et certaines de ses versions antérieures, alors peut-être qu’il faudra se « contenter » de Flex, ce qui en soi n’est quand même pas si mal.

# à placer sur les parents

## display

C’est notre bon vieux display qui se voit équipé de nouvelles valeurs. Ce display est à mettre sur le parent et transforme l’élément en question en un élément qui peut accueillir une découpe sous forme de grille.

* **grid** : génère une grille de type block
* **inline-grid** :  génère une grille de type inline
* subgrid : si votre grille est déjà un élément de grille, vous pouvez en faire une sous-grille

##### **Attention**Column, float, clear et vertical-align n’ont aucun effet sur un élément « gridé ».

## grid-gap, grid-column-gap et grid-row-gap

C’est la fonction raccourcie de grid-column-gap et grid-row-gap et cela fonctionne un peu comme un margin (mais pas tout à fait) : cela ajoute de l’espace entre les éléments de la grille, mais pas autour d’eux.

Syntaxe

###### grid-gap : <grid-row-gap> <grid-column-gap>;

Si vous n’indiquez qu’une valeur, vous aurez la même marge entre les colonnes et les lignes, sinon vous pouvez les dissocier de cette façon :

###### grid-gap:50px 20px;

## grid-template-columns et grid-template-rows

Définit le format pour les colonnes ou les lignes de votre grille. Les différentes valeurs sont séparées par un espace.

Syntaxe

###### grid-template-columns : <track-size> … | <line-name> <track-size>;grid-template-rows : <track-size> … | <line-name> <track-size>;

Faisons pour le moment abstraction de la possibilité que nous avons de les nommer, nous reviendrons sur ce point plus tard.

Partons d’un exemple simple pour observer comment se comporte CSS Grid.

En indiquant ces valeurs pour les colonnes :

###### grid-gap:10px;grid-template-columns: 50px 50px 50px;



On constate plusieurs choses :

* La grille, avec l’inspecteur de Firefox Dev edition, est représentée par des bordures pleines
* Quand on n’indique que 3 colonnes dans notre template, alors il passe automatiquement à la ligne s’il y a plus que 3 éléments
* Le gap (zone pointillée) n’est présent qu’entre les items de la grille et pas entre les éléments de la grille et les autres éléments.

Si on désire que la 3e item prenne « tout l’espace restant », on peut incorporer la valeur auto à la place d’une valeur chiffrée :

###### grid-template-columns: 50px 50px auto;



Gros avantage de ça évidemment : la grille est parfaitement « responsive » sur sa colonne auto.

Le fait que les colonnes passent à la ligne automatiquement même si on ne leur a pas indiqué qu’elles le devaient, c’est ce qu’on appelle des créations de lignes implicites parce qu’on ne les a pas définies. Ce qui est créé de façon implicite, dans Firefox est représenté par des pointillés :



Tandis que les colonnes que j’ai explicitement créées ci-dessous sont représentées par des traits-d’ union :



Avec grid, une nouvelle unité de mesure débarque : les fractions ! (ou fr)

Prenons un exemple concret :

###### grid-template-columns: 50px 1fr 1fr 50px;



L’unité fr prend l’espace libre restant (sans les données fixes etc.) et le redistribue en X où X est égal au nombre de fractions demandées. Donc ici, on demande 2 blocs qui mesurent chacun une fraction soit 2 fractions de la même taille.

Quelle est la différence entre auto et les unités 1fr alors ? Si on les utilise séparément, en réalité, ils fonctionnent de la même façon.

Si, en revanche, vous utilisez les fr et auto en même temps, alors auto ne prend plus, en largeur, que la taille de son contenu et laisse aux fr le loisir de s’occuper de l’espace libre restant.

### Repeat

Avec cette propriété vient également la possibilité de répéter une séquence de cette façon :

###### grid-template-columns: repeat(5, 1fr);

À la gauche de la virgule, le multiplicateur et à droite ce qui va être répété, cela peut être un seul élément comme plusieurs de cette façon :

###### grid-template-columns: repeat(3, 1fr 2fr);



Vous pouvez aussi précéder ou faire suivre le repeat d’un élément qui ne sera pas répété :

###### grid-template-columns: 5fr repeat(3, 1fr 2fr);



### Auto-fill, auto-fit et minmax

Ce sont des paramètres qui ne vont intervenir qu’à l’intérieur d’un élément répété. Alors que auto-fill et auto-fit remplacent le multiplicateur, minmax va remplacer ce qui est répété.

Auto-fill et auto-fit sont là pour « combler » les espaces au seins de lignes ou de colonnes quand on ne spécifie pas combien d’items on désire par ligne ou colonne.

Associé à minmax, ça va nous permettre de faire du responsive sans l’utilisation de media-queries ! Et ça, c’est trop cool.

En réalité, auto-fill et auto-fit sont très proches au niveau de leur fonctionnement mais un élément les différencie.

Auto-fill remplit la ligne avec le plus de colonnes qu’il peut. Donc il crée des colonnes implicite virtuelles (il prévoit la place en CSS pas, il ne va pas créer ces éléments en HTML) tant qu’une nouvelle colonne peut être placée parce qu’il essaye de remplir la ligne avec le plus de colonnes qu’il peut.

Auto-fit adapte les colonnes qui sont déjà présentes en les agrandissant pour qu’elles prennent tout l’espace libre disponible.

La combi gagnante sera souvent d’associer auto-fit avec minmax(valeurmin, valeurmax) comme ceci :

###### grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

## grid-auto-flow

Grid-auto-flow permet d’indiquer à un élément qui n’aurait pas explicitement été créé de se placer soit dans la colonne ou la ligne suivante. Par défaut, la valeur est réglée sur row.

C’est pour ça qu’avec cette définition de template :

###### grid-template-columns: 50px 50px;

On obtient ce résultat-ci :



Alors qu’il y a de la place pour se mettre à côté, nativement, tout élément de grille qui n’a pas été explicitement indiqué, va « couler » dans une nouvelle ligne.

Pour changer ça, il y a grid-auto-flow :

###### grid-auto-flow : column;



De cette façon, les éléments non-explicites coulent dans les colonnes suivantes. Nous verrons comment avoir un impact sur leur taille plus tard.

## grid-auto-columns et grid-auto-rows

Définit la taille des éléments qui sont implicitement placés en row ou en colonne.

## grid-template-areas

Avec grid-template-areas, vous allez définir des zones au niveau textuel et vous allez ensuite pouvoir attribuer un enfant de la grille à cette zone.

Prenons un exemple concret au niveau html nous avons ceci :

###### <div class="gridded">

######  <div class="grid-child grid-child--header">header</div>

######  <div class="grid-child grid-child--2">2</div>

######  <div class="grid-child grid-child--3">3</div>

######  <div class="grid-child grid-child--4">4</div>

######  <div class="grid-child grid-child--footer">footer</div>

###### </div>

On peut définir notre template de la sorte :

###### grid-template-columns: 1fr 10fr 1fr;

Ensuite, on peut identifier des zones regroupées par lignes de cette façon avec grid-template-areas :

###### grid-template-areas:

######  "header header header"

######  "sidebar1 content sidebar2"

######  "footer footer footer"

###### }

Ensuite, il ne nous reste plus qu’à cibler l’enfant et le rattacher à cette zone de cette façon :

###### .header {

######  grid-area:header;

###### }

###### .footer {

######  grid-area:footer;

###### }

On obtient le résultat suivant :



# à placer sur les enfants

## grid-column-start, grid-column-end, grid-row-start, grid-row-end, grid-column, grid-row et grid-area

Ces 6 propriétés CSS sont groupées parce que 2 d’entre elles sont des écritures raccourcies des 2 autres.

Grid-column est la version raccourcie de grid-column-start et grid-column-end.

Grid-row est la version raccourcie de grid-row-start et grid-row-end.

Grid-area est la version raccourcie de grid-row-start, grid-column-start, grid-row-end et grid-column-end (dans cet ordre-là).

Mais à quoi ça sert au fond ?

Ça sert, sur les enfants d’une grille, à les positionner exactement comme vous le désirez. Vous pouvez faire commencer un élément de grille à partir d’une certaine colonne jusqu’à une certaine colonne et c’est pareil pour les lignes.

En plus, vous pouvez faire ce qui ressemble à un « colspan » dans les tableaux à l’aide de span au sein de ces propriétés css.

Concrètement voyons à quoi ça ressemble.

Dessinons une grande grille de 30 cases disposées en 6 lignes de 5 colonnes.

###### grid-template-columns: repeat(5,1fr);

Et puis découvrons ce qu’on peut faire avec ces propriétés.

### Span

Introduisons maintenant la notion de « span » au sein de ces propriétés à partir de cette grille :


Imaginons que nous aimerions que le « 1 » prenne la place de 2 cases. Il existe plusieurs façons d’y arriver :

La première serait de lui dire qu’il commence à la colonne 1 et se termine à la colonne 3 de cette façon :

###### .grid-child-1 {

######  grid-column:1/3;

###### }

À la gauche de / nous avons le grid-column-start et à la droite le grid-column-end.

Il est important de noter qu’on compte les colonnes et les lignes par leur bordure initiale (gauche pour les colonnes et haut pour les lignes).

Autre façon d’envisager le problème qui donnera le même résultat :

###### .grid-child-1 {

######  grid-column:1/-5;

###### }

Quand vous utilisez les valeurs négatives comme c’est le cas ici, ça veut dire que vous « comptez » les items à rebours càd en commençant par le dernier et en remontant.

Dernière façon de faire :

###### .grid-child-1 {

######  grid-column:span 2;

###### }

Ici, on indique simplement qu’on désire fusionner sur une longueur de « 2 colonnes » à partir de la case en question.



Et on peut évidemment effectuer des fusions sur les lignes et les colonnes en même temps :

###### .grid-child-1 {

######  grid-column:span 3;

######  grid-row:span 5;

###### }



Avec grid-area, vous définissez tout d’un coup donc, mais en plus de ça, vous avez la possibilité de superposer 2 grid-area l’une sur l’autre.