
Technologies ▼

References & Guides ▼

Feedback ▼

Sign in 

 Search

Supporting older browsers

Previous

Overview: CSS layout

Next

In this module, we recommend using Flexbox and Grid as the main layout methods for your designs. However, there will be visitors to your site who use older browsers, or browsers which do not support the methods you have used. This will always be the case on the web — as new features are developed, different browsers will prioritise different things. This article explains how to use modern web techniques without locking out users of older technology.

Prerequisites: HTML basics (study [Introduction to HTML](#)), and an idea of how CSS works (study [Introduction to CSS and Styling boxes](#).)

Objective: To understand how to provide support for your layouts on older browsers that might not support the features you want to use.

What is the browser landscape for your site? 

Every website is different in terms of target audience. Before deciding on the approach to take, find out the number of visitors coming to your site using older browsers. This is straightforward if you have an existing website which you are adding to or replacing, as you probably have analytics available which can tell you the technology people are using. If you have no analytics or this is a brand new site, then there are sites such as [Statcounter](#) that can provide statistics filtered by location.

















You should also consider the type of devices and the way people use your site, for example, you may expect a higher than an average number of mobile devices. Accessibility and people using assistive technology should always be considered, but for some sites that may be even more critical. In my experience, developers are often very worried about the experience of 1% of users in an old version of Internet Explorer, while not considering at all the far greater number who accessibility needs.

What is the support for the features you want to use?

Once you know the browsers that come to your site, you can assess any technology that you want to use against how well it is supported and how easily you can provide an alternative for visitors who do not have that technology available. We are trying to make this easy for you at MDN, by providing browser compatibility information on each page detailing a CSS property. For example, take a look at the page for `grid-template-columns`. At the bottom of this page is a table, which lists major browsers along with the version they began supporting this property.

Browser compatibility

New compatibility tables are in beta ▾

														
														
Basic support	57 ▼	16	52 ▼	No	44 ▼	10.1	57	57 ▼	16	52 ▼	44	10.3	6.0	
<code>minmax()</code>	57 ▼	12	52 ▼	?	44 ▼	10.1	No	?	12	52 ▼	No	10.3	?	
<code>repeat()</code>	57 ▼	16	52 * ▼	No	44 ▼	10.1	?	?	No	52 * ▼	?	10.3	?	
<code>fit-content()</code>	29	16	51	No	44	10.1	57	57	16	51	44	10.3	7.0	

-  Full support
-  No support
-  Compatibility unknown
-  See implementation notes.
-  User must explicitly enable this feature.

Another popular way to find out about how well a feature is supported is the [Can I Use](#) website. This site lists the majority of Web Platform features with information about their browser support status. You can view usage statistics by location — useful if you work on a site that has users mostly for a specific area of the world. You can even link your Google Analytics account to get analysis based on your user data.

Understanding the technology your users have, and the support for things you might want to use puts you in a good place to make all of your decisions and to know how best to support all of your users.

Support doesn't mean "looks the same"

A website can't possibly look the same in all browsers, because some of your users will be viewing the site on a phone and others on a large desktop screen. Similarly, some of your users will have an old browser version, and others the latest browser. Some of your users might be hearing your content read out to them by a screen reader, or have zoomed in on the page to be able to read it. Supporting everyone means serving a version of your content that is designed defensively, so that it will look great on modern browsers, but will still be usable at a basic level for users of older browsers.

A basic level of support comes from structuring your content well so that the normal flow of your page makes sense. A user with a very limited feature phone might not get much of your CSS, but the content will flow in a way that makes reading easy. Therefore, a well-structured HTML document should always be your starting point. *If you remove your stylesheet, does your content make sense?*

One option is to leave this plain view of the site as the fallback for people using very old or limited browsers. If you have a tiny number of people coming to the site in these browsers it may not make commercial sense to pour time into trying to give them a similar experience to people on modern browsers. It would be better to spend the time on things which make the site more accessible, thus serving far more users. There is a middle ground between a plain HTML page and all the bells and whistles, and CSS has actually made creating these fallbacks pretty straightforward.

Creating fallbacks in CSS

CSS specifications contain information that explains what the browser does when two layout methods are applied to the same item. This means that there is a definition for what happens if a floated item, for example, is also a Grid Item using CSS Grid Layout. Couple this information with the knowledge that browsers ignore CSS that they don't understand, and you have a way to create simple layouts using the legacy techniques we have already covered, which are then overwritten by your Grid layout in modern browsers that understand it.

In the below example, we have floated three `<div>` so they display in a row. Any browser that does not support CSS Grid Layout will see the row of boxes as a floated layout. A floated item that becomes a grid item loses the float behaviour, which means that by turning the wrapper into a Grid Container, the floated items become Grid Items. If the browser supports Grid Layout it will display the grid view, if not it ignores the and `display: grid` related properties and the floated layout is used.

```
1  * {box-sizing: border-box;}
2
3  .wrapper {
4      background-color: rgb(79,185,227);
5      padding: 10px;
```

```

6    max-width: 400px;
7    display: grid;
8    grid-template-columns: 1fr 1fr 1fr;
9  }
10
11  .item {
12    float: left;
13    border-radius: 5px;
14    background-color: rgb(207,232,220);
15    padding: 1em;
16  }

```

```

1  <div class="wrapper">
2    <div class="item">Item One</div>
3    <div class="item">Item Two</div>
4    <div class="item">Item Three</div>
5  </div>

```

Item One

Item Two

Item Three

Note: The `clear` property also has no effect once the cleared item becomes a grid item, so you could have a layout with a cleared footer, which is then turned into a Grid Layout.

Fallback Methods

There are a number of layout methods which can be used in a similar way to this float example. You can choose the one that makes the most sense for the layout pattern you need to create.

Float and **clear**

As shown above, the float and clear properties cease to affect the layout if floated or cleared items to become flex or grid items.

display: inline-block

This method can be used to create column layouts, if an item has `display: inline-block` set but then becomes a flex or grid item, the inline-block behaviour is ignored.

display: table

The method of creating CSS Tables described in the introduction to these lessons can be used as a fallback. Items that have CSS table layouts set on them will lose this behaviour if they become flex or grid items. Importantly, the anonymous boxes created to fix up the table structure are not created.

Multiple-column Layout

For certain layouts you could use `multi-col` as a fallback, if your container has any of the `column-*` properties defined on it and then becomes a grid container, the multicol behaviour will not happen.

Flexbox as a Fallback for Grid

Flexbox has greater browser support than Grid due to being supported by IE10 and 11, although do check out the information later in this lesson explaining the rather patchy and confusing support for Flexbox in older browsers. If you make a flex container into a grid container, any `flex` property applied to the children will be ignored.

For many layout tweaks in older browsers, you may find you can give a decent experience by using CSS in this way. We add a simpler layout based on older and well-supported techniques, then use the newer CSS to create the layout that over 90% of your audience will see. There are cases, however, when the fallback code will need to include something that the new browsers will also interpret. A good example of this is if we were to add percentage widths to our floated items to make the columns more like the grid display, stretching to fill the container.

In the floated layout, the percentage is calculated from the container — 33.333% is a third of the container width. In Grid however that 33.333% is calculated from the grid area the item is placed in, so it actually becomes a third of the size we want once the Grid Layout is introduced.

```
1  * {box-sizing: border-box;}
2
3  .wrapper {
    background-color: rgb(79,185,227);
```

```

4   padding: 10px;
5   max-width: 400px;
6   display: grid;
7   grid-template-columns: 1fr 1fr 1fr;
8 }
9
10  .item {
11    float: left;
12    border-radius: 5px;
13    background-color: rgb(207,232,220);
14    padding: 1em;
15    width: 33.333%;
16  }
17
1  <div class="wrapper">
2    <div class="item">Item One</div>
3    <div class="item">Item Two</div>
4    <div class="item">Item Three</div>
5  </div>

```

Item
One

Item
Two

Item
Three

To deal with this issue we need to have a way to detect if Grid is supported and therefore if it will override the width. CSS has a solution for us here.

Feature queries allow you to test whether a browser supports any particular CSS feature. This means that you can write some CSS for browsers that don't support a certain feature, then check to see if the browser has support and if so throw in your fancy layout.

If we add a feature query to the above example, we can use it to set the widths of our items back to `auto` if we know that we have grid support.

```
1  * {box-sizing: border-box;}
2
3  .wrapper {
4    background-color: rgb(79,185,227);
5    padding: 10px;
6    max-width: 400px;
7    display: grid;
8    grid-template-columns: 1fr 1fr 1fr;
9  }
10
11 .item {
12   float: left;
13   border-radius: 5px;
14   background-color: rgb(207,232,220);
15   padding: 1em;
16   width: 33.333%;
17 }
18
19 @supports (display: grid) {
20   .item {
21     width: auto;
22   }
23 }
```

```
1  <div class="wrapper">
2    <div class="item">Item One</div>
3    <div class="item">Item Two</div>
4    <div class="item">Item Three</div>
5  </div>
```


Item One

Item Two

Item Three

Support for feature queries is very good across modern browsers, however, you should note that it is the browsers that do not support CSS Grid, which also doesn't support feature queries. This means that an approach as detailed above will work for those browsers. What we are doing is writing our old CSS first, outside of any feature query. Browsers that do not support Grid, and do not support the feature query will use that layout information they can understand and completely ignore everything else. The browsers that support the feature query also support CSS Grid and so will run the grid code and the code inside the feature query.

The specification for feature queries also contains the ability to test if a browser does not support a feature — this is only helpful if the browser does support feature queries. In the future, an approach of checking for lack of support will work, as the browsers that don't have feature query support go away. For now, however, use the approach of doing the older CSS, then overwriting it, for the best support.

Older versions of Flexbox

In older versions of browsers, you can find previous iterations of the Flexbox specification. At the time of writing, this is mostly an issue with Internet Explorer 10, which uses the `-ms-` prefix for Flexbox. This also means that there are some outdated articles and tutorials in existence; this useful guide helps you check what you are looking at and can also help if you need flex support in very old browsers.

The IE10 and 11 prefixed version of Grid

The CSS Grid specification was initially prototyped in Internet Explorer 10; this means that while IE10 and IE11 do not have *modern* grid support, they do have a version of Grid layout that is very usable, although different to the modern specification documented on this site. The IE10 and 11 implementations is `-ms-` prefixed, which means you can use it for these browsers and it will be ignored by non-Microsoft browsers. Edge does still understand the old syntax, however, so take care that everything is safely overwritten in your modern grid CSS.

The guide to [Progressive Enhancement in Grid Layout](#) can help you understand the IE version of the grid, and we have included some additional useful links at the end of this lesson. However, unless you have a very high number of visitors in older IE versions, you may find it better to focus on creating a fallback that works for all non-supporting browsers.

Testing older browsers

With the majority of browsers supporting Flexbox and Grid, it can be reasonably hard to test older browsers. One way is to use an online testing tool such as Sauce Labs, as detailed in the [Cross browser testing module](#).

You can also download and install Virtual Machines, and run older versions of browsers in a contained environment on your own computer. Having access to older versions of Internet Explorer is particularly useful, and for that purpose, Microsoft has made [a range of Virtual Machines](#) available for free download. These are available for Mac, Windows and Linux operating systems and so are a great way to test in old and modern Windows browsers even if you are not using a Windows computer.

Summary

You now have the knowledge to confidently use techniques such as Grid and Flexbox, create fallbacks for older browsers, and make use of any new techniques that might come along in the future.

See Also

- [Using Feature Queries in CSS](#)
- [Backwards Compatibility of Flexbox](#)
- [CSS Grid Layout and Progressive Enhancement](#)
- [Using CSS Grid: Supporting Browsers Without Grid](#)
- [A tutorial which uses the IE10 and 11 version of Grid](#)
- [Should I try to use the IE10 implementation of Grid Layout?](#)
- [Cascading Web Design with Feature Queries](#)
- [Using Feature Queries \(Video\)](#)

[Previous](#)[Overview: CSS layout](#)[Next](#)

In this module

- [Introduction to CSS layout](#)
 - [Normal Flow](#)
 - [Flexbox](#)
 - [Grid](#)
 - [Floats](#)
 - [Positioning](#)
 - [Multiple-column Layout](#)
 - [Legacy Layout Methods](#)
 - [Supporting older browsers](#)
 - [Fundamental Layout Comprehension Assessment](#)
-