


Technologies ▾

References & Guides ▾

Feedback ▾

Sign in 

 Search

Advanced box effects

Previous

Overview: Styling boxes

Next

This article acts as a box of tricks, providing an introduction to some of the advanced features available for styling boxes like box shadows, blend modes and filters.

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

Objective: To get an idea about how to use advanced box effects, and know about some of the nascent styling tools that are appearing in the CSS language.

Box shadows

Back in our [Styling text](#) module, we looked at the `text-shadow` property, which allows you to apply one or more drop shadows to the text of an element. Well, an equivalent property exists for boxes — `box-shadow` allows you to apply one or more drop shadows to an actual element box. Like text shadows, box shadows are supported pretty well across browsers, but only in

IE9+. Your users of older IE versions might just have to cope with no shadows, so just test your designs to make sure your content is legible without them.

You can find the examples in this section at [box-shadow.html](#) (see the [source code](#) too).

A simple box shadow [↗](#)

Let's look at a simple example to get things started. First, some HTML:

```
1 <article class="simple">
2   <p><strong>Warning</strong>: The thermostat on the cosmic transcender has r
3 </article>
```

Now the CSS:

```
1 p {
2   margin: 0;
3 }
4
5 article {
6   max-width: 500px;
7   padding: 10px;
8   background-color: red;
9   background-image: linear-gradient(to bottom, rgba(0,0,0,0), rgba(0,0,0,0.25)
10 }
11
12 .simple {
13   box-shadow: 5px 5px 5px rgba(0,0,0,0.7);
14 }
```

This gives us the following result:

Warning: The thermostat on the cosmic transcender has reached a critical level.

You'll see that we've got four items in the `box-shadow` property value:

1. The first length value is the **horizontal offset** — the distance to the right the shadow is offset from the original box (or left, if the value is negative).
2. The second length value is the **vertical offset** — the distance downwards that the shadow is offset from the original box (or upwards, if the value is negative).
3. The third length value is the **blur radius** — the amount of blurring applied to the shadow.
4. The color value is the **base color** of the shadow.

You can use any length and color units that would make sense to do so to define these values.

Multiple box shadows [↗](#)

You can also specify multiple box shadows in a single `box-shadow` declaration, by separating them with commas:

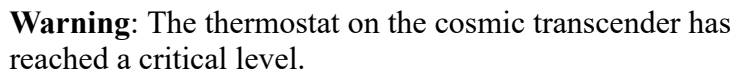
```
1  p {
2    margin: 0;
3  }
4
5  article {
6    max-width: 500px;
7    padding: 10px;
8
9    background-color: red;
10   background-image: linear-gradient(to bottom, rgba(0,0,0,0), rgba(0,0,0,0.25
11 }
12
13 .multiple {
14   box-shadow: 1px 1px 1px black,
15              2px 2px 1px black,
16              3px 3px 1px red,
```

```

16 |         4px 4px 1px red,
17 |         5px 5px 1px black,
18 |         6px 6px 1px black;
19 |     }

```

Now we get this result:



Warning: The thermostat on the cosmic transcender has reached a critical level.

We've done something fun here by creating a raised box with multiple coloured layers, but you could use it in any way you want, for example to create a more realistic look with shadows based on multiple light sources.

Other box shadow features [↗](#)

Unlike text-shadow, box-shadow has an `inset` keyword available — putting this at the start of a shadow declaration causes it to become an inner shadow, rather than an outer shadow. Let's have a look and see what we mean.

First, we'll go with some different HTML for this example:

```

1 | <button>Press me!</button>

1 | button {
2 |     width: 150px;
3 |     font-size: 1.1rem;
4 |     line-height: 2;
5 |     border-radius: 10px;
6 |     border: none;
7 |     background-image: linear-gradient(to bottom right, #777, #ddd);
8 |     box-shadow: 1px 1px 1px black,
9 |                inset 2px 3px 5px rgba(0,0,0,0.3),

```

```
10         inset -2px -3px 5px rgba(255,255,255,0.5);
11     }
12
13     button:focus, button:hover {
14         background-image: linear-gradient(to bottom right, #888, #eee);
15     }
16
17     button:active {
18         box-shadow: inset 2px 2px 1px black,
19                     inset 2px 3px 5px rgba(0,0,0,0.3),
20                     inset -2px -3px 5px rgba(255,255,255,0.5);
21     }
```

This gives us the following result:



Press me!

Here we've set up some button styling along with focus/hover/active states. The button has a simple black box shadow set on it by default, plus a couple of inset shadows, one light and one dark, placed on opposite corners of the button to give it a nice shading effect.

When the button is pressed in, the active state causes the first box shadow to be swapped for a very dark inset shadow, giving the appearance of the button being pressed in.

Note: There is another item that can be set in the `box-shadow` value — another length value can be optionally set just before the color value, which is a **spread radius**. If set, this causes the shadow to become bigger than the original box. It is not very commonly used, but worth mentioning.

CSS Filters provide a way to apply filters to an element in the same way as you might apply a filter to a layer in a graphics package like Photoshop. There are a number of different options available, and you can read all about them in greater detail on the [filter](#) reference page. In this section, we'll explain to you the syntax, and show you how much fun the results can be.

Basically, a filter can be applied to any element, block or inline — you just use the `filter` property, and give it a value of a particular filter function. Some of the filter options available do very similar things to other CSS features, for example `drop-shadow()` works in a very similar way and gives a similar effect to `box-shadow` or `text-shadow`. The really nice thing about filters however, is that they work on the exact shapes of the content inside the box, not just the box itself as one big chunk, which can look nicer, although it may not always be what you want. Let's use a simple example to illustrate what we mean:

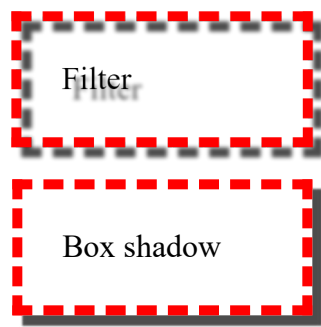
First, some simple HTML:

```
1 <p class="filter">Filter</p>
2
3 <p class="box-shadow">Box shadow</p>
```

And now some CSS to apply a drop shadow to each:

```
1 p {
2   margin: 1rem auto;
3   padding: 20px;
4   width: 100px;
5   border: 5px dashed red;
6 }
7
8 .filter {
9   -webkit-filter: drop-shadow(5px 5px 1px rgba(0,0,0,0.7));
10  filter: drop-shadow(5px 5px 1px rgba(0,0,0,0.7));
11 }
12
13 .box-shadow {
14   box-shadow: 5px 5px 1px rgba(0,0,0,0.7);
15 }
```

This gives us the following result:



As you can see, the filter drop-shadow follows the exact shape of the text and border dashes. The box shadow just follows the square of the box.

Some other things to note:

- Filters are very new — they are supported in most modern browsers, including Microsoft Edge, but they are not supported in Internet Explorer at all. If you use filters in your designs, you should therefore make sure your content is usable without the filters.
- You'll see that we've included a version of the `filter` property with `-webkit-` prefixed. This is called a Vendor Prefix, and is used sometimes by a browser before it finalizes its implementation of a new feature, to support that feature and experiment with it while not clashing with the non-prefixed version. Vendor prefixes were never intended to be used by web developers, but they do get used sometimes on production pages if experimental features are really desired. In this case, the `-webkit-` version of the property is currently required for Chrome/Safari/Opera support, while Edge and Firefox use the final, non-prefixed version.

Note: If you do decide to use prefixes in your code, make sure you include all the prefixes required as well as the non-prefixed version, so the maximum number of browsers possible can use the feature, and when browsers later drop the prefixes, they can use the non-prefixed version too. Also be warned that these experimental features might change, so your code might break. It is really best to just experiment with these features until the prefixes are removed.

You can see some other filter examples at [filters.html](#) (also see the [source code](#)).

Blend modes [↗](#)

CSS blend modes allow us to add blend modes to elements that specify a blending effect when two elements overlap — the final color shown for each pixel will be the result of a combination of the original pixel color, and that of the pixel in the layer underneath it. Blend modes are again very familiar to users of graphics applications like Photoshop.

There are two properties that use blend modes in CSS:

- `background-blend-mode`, which blends together multiple background images and colors set on a single element.
- `mix-blend-mode`, which blends together the element it is set on with elements it is overlapping — both background and content.

You can find a lot more examples than are available here in our [blend-modes.html](#) example page (see [source code](#)), and on the `<blend-mode>` reference page.

Note: Blend modes are also very new, and slightly less well supported than filters. There is no support as yet in Edge, and Safari only supports some of the blend mode options.

background-blend-mode [↗](#)

Again, let's look at some examples so we can understand this better. First, `background-blend-mode` — here we'll show a couple of simple `<div>`s, so you can compare the original with the blended version:

```
1 <div>
2 </div>
3 <div class="multiply">
4 </div>
```

Now some CSS — we are adding to the `<div>` one background image and a green background color:

```
1 div {
2   width: 250px;
3   height: 130px;
```

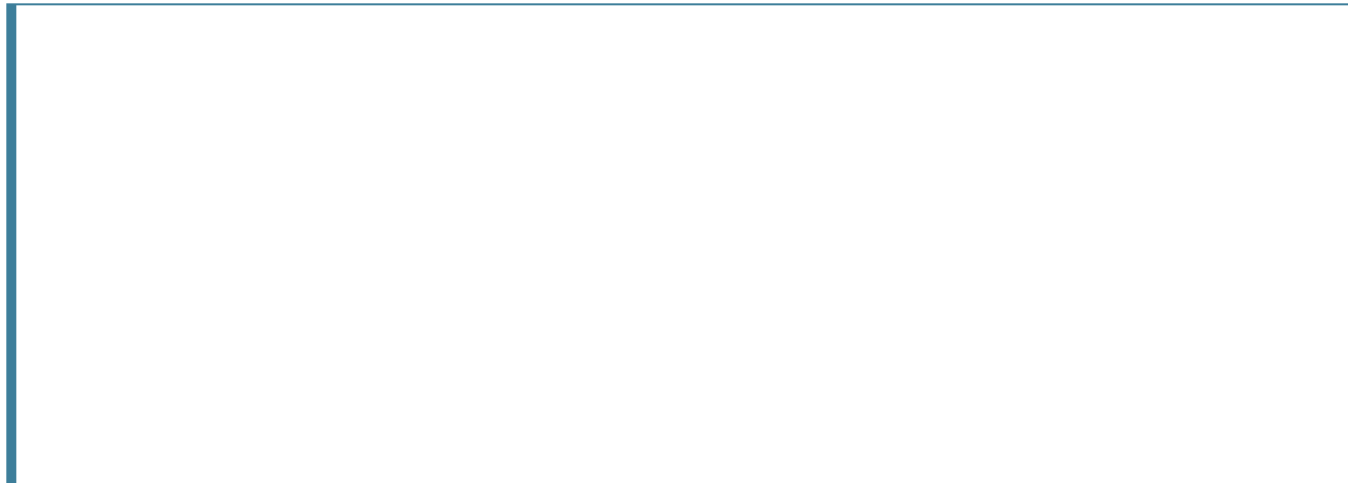


```

4   padding: 10px;
5   margin: 10px;
6   display: inline-block;
7   background: url(https://mdn.mozillademos.org/files/13090/colorful-heart.png)
8   background-color: green;
9   }
10
11  .multiply {
12    background-blend-mode: multiply;
13  }

```

The result we get is this — you can see the original on the left, and the multiply blend mode on the right:



mix-blend-mode [↗](#)

Now let's look at `mix-blend-mode`. Here we'll present the same two `<div>`s, but each one is now sat on top of a simple `<div>` with a purple background, to show how the elements will blend together:

```

1  <article>
2    No mix blend mode
3    <div>
4
5  </div>
6  <div>
7  </div>

```

```
8   </article>
9
10  <article>
11    Multiply mix
12    <div class="multiply-mix">
13
14    </div>
15    <div>
16    </div>
17  </article>
```

Here's the CSS we'll style this with:

```
1  article {
2    width: 280px;
3    height: 180px;
4    margin: 10px;
5    position: relative;
6    display: inline-block;
7  }
8
9  div {
10   width: 250px;
11   height: 130px;
12   padding: 10px;
13   margin: 10px;
14 }
15
16 article div:first-child {
17   position: absolute;
18   top: 10px;
19   left: 0;
20   background: url(https://mdn.mozillademos.org/files/13090/colorful-heart.png);
21   background-color: green;
22 }
23
24 article div:last-child {
25   background-color: purple;
26   position: absolute;
27   bottom: -10px;
```

```

28     right: 0;
29     z-index: -1;
30 }
31
32 .multiply-mix {
33     mix-blend-mode: multiply;
34 }

```

This gives us the following results:

No mix blend mode

Multiply mix

You can see here that the multiply mix blend has blended together not only the two background images, but also the color from the `<div>` below it too.

Note: Don't worry if you don't understand some of the layout properties above, like `position`, `top`, `bottom`, `z-index`, etc. We will cover these in great detail in our CSS Layout module.

-webkit-background-clip: text [🔗](#)

Another nascent feature we thought we'd mention briefly before moving on (currently supported in Chrome, Safari, and Opera, and being implemented in Firefox) is the `text` value for `background-clip`. When used along with the proprietary `-webkit-text-fill-color:`

`transparent`; feature, this allows you to clip background images to the shape of the element's text, making for some nice effects. This is not an official standard, but has been implemented across multiple browsers, as it is popular, and used fairly widely by developers. When used in this context, both of the properties would require a `-webkit-` vendor prefix, even for Non-Webkit/Chrome-based browsers:

```
1  .text-clip {  
2    -webkit-background-clip: text;  
3    -webkit-text-fill-color: transparent;  
4  }
```

So why have other browsers implemented a `-webkit-` prefix? Mainly for browser compatibility — so many web developers have started implementing websites with `-webkit-` prefixes that it started to look like the other browsers were broken, whereas in actual fact they were following the standards. So they were forced to implement a few such features. This highlights the danger of using non-standard and/or prefixed CSS features in your work — not only do they cause browser compatibility issues, but they are also subject to change, so your code could break at any time. It is much better to stick to the standards.

If you do want to use such features in your production work, make sure to test across browsers thoroughly and check that, where these features don't work, the site is still usable.

Note: For a full `-webkit-background-clip: text` code example, see [background-clip-text.html](#) (see also the [source code](#)).

Active learning: experiment with some effects [↗](#)

Now it's your turn. For this active learning, we want you to experiment with some of the effects you've read about above, using the provided code below.

If you make a mistake, you can always reset the example using the *Reset* button.

HTML Input

```
<div class="style-me">  
</div>
```

CSS Input

```
.style-me {  
  width: 280px;  
  height: 130px;  
  padding: 10px;  
  margin: 10px;  
  display: inline-block;  
  background-color: red;  
  background: url(https://mdn.mozillademos.org/files/13090/colorful-heart.png) no-repeat center 20px,  
              linear-gradient(to bottom right, #f33, #a33);  
}
```

Output

Reset

Summary [↗](#)

We hope this article was fun — playing with shiny toys generally is, and it is always interesting to see what kinds of tools are just becoming available in cutting edge browsers. You've reached the end of the [Styling boxes](#) articles, so next up you'll test your box styling skills with our assessments.

[Previous](#)[Overview: Styling boxes](#)[Next](#)

In this module [↗](#)

- Box model recap
 - Backgrounds
 - Borders
 - Styling tables
 - Advanced box effects
 - Creating a fancy letterheaded paper
 - A cool looking box
-