CSS3

HANDBOOK

mattlambert.ca

Pseudo Selectors

You are likely familiar with existing pseudo class selectors like: :hover, :active, :link and :visited. CSS3 has expanded massively on this category and we have a number of new selectors we can use to better control elements on a page. Let's jump right into them as there are a number to go over.

Browser Support

Currently the minimum browser support includes: IE9, Firefox 24, Chrome 29, Safari 5.1, Opera 17.

:first-child

The **:first-child** pseudo class selector will grab the **first** child of it's type in a parent.

CSS

```
li:first-child {
   color: red;
}
```

HTML

```
<ul>
   <li>red item</li>
   <li>item</li>
   <li>item</li>
</ul>
```

The CSS Explained

We've setup an unordered list and applied the **:first-child** pseudo class selector. This will result in the first item, in the list, being red.

:last-child

The **:last-child** pseudo class selector will grab the **last** child of it's type in a parent.

CSS

```
li:last-child {
   color: red;
}
```

HTML

```
<ul>
   <li>item</li>
   <li>item</li>
   <li>red item</li>
</ul>
```

The CSS Explained

We've setup an unordered list and applied the **:last-child** pseudo class selector. This will result in the last item, in the list, being red.

:only-child

The **:only-child** pseudo class selector will grab the **only** item of it's type in a parent. This class is less specific than :first-child or :last-child.

CSS

```
li:only-child {
   color: red;
}
```

HTML

```
<ul>
   <li>red item</li>
</ul>
```

The CSS Explained

We've setup an unordered list and applied the **:only-child** pseudo class selector. This will result in the only item, in the list, being red.

:first-of-type

The **:first-of-type** pseudo class selector grabs the first item within any parent.

CSS

```
li:first-of-type {
   color: red;
}
```

HTML

```
<ul>
    <li>red item</li>
    <li>item</li>
    <li>item</li>
</ul>
<ul>
    <li>red item</li>
    <li>item</li>
    <li>item</li>
</ul>
```

The CSS Explained

We now have two unordered list and we've applied the **:first-of-type** pseudo class to all **<li>** tags. This will result in the first item, in each list, being red.

:last-of-type

The **:last-of-type** class selector is the opposite of the **:first-of-type** selector. It will select the last item within any parent.

CSS

```
li:last-of-type {
   color: red;
}
```

HTML

```
<ul>
    <li>item</li>
    <li>item</li>
    <li>red item</li>
</ul>
<ul>
    <li>red item</li>
    <li>item</li>
    <li>red item</li>
</ul>
```

The CSS Explained

We now have two unordered list and we've applied the **:last-of-type** pseudo class to all **<li>** tags. This will result in the last item, in each list, being red.

:only-of-type

The **:only-of-type** class selector will only grab an item if it's the **only one** of it's kind within the current parent.

CSS

```
.parent ul:only-of-type {
   color: red;
}
```

HTML

```
<div class="parent">
  <ul>
    <li>red item</li>
    <li>red item</li>
    <li>red item</li>
  </ul>
  <p>lorem ipsum blah blah blah...</p>
  <p>lorem ipsum blah blah blah...</p>
</div>
```

The CSS Explained

I've applied the **:only-of-type** pseudo class selector to the **<ul>** tag within the **.parent **. Since it is the only **<ul>** tag, each list item will be red. If we were to change the declaration to **.parent p:only-of-type** it would not work. Reason being, there is more than one **<p>** tag within the parent.

:nth-child

The **:nth-child** class selector uses algebra to grab elements based on the formula you enter. For example **:nth-child(2n+2)** will grab every second item in an unordered list. That is the 2nd, 4th, 6th, 8th, etc... items in the list.

:nth-child also accepts the keywords **odd and even**. This can be quite handy for doing something like zebra-striping the rows in a table.

CSS

```
li:nth-child(2n+2) {
    color: red;
}
```

HTML

```
<ul>
  <li>item</li>
  <li>red item</li>
  <li>item</li>
  <li>red item</li>
  <li>item</li>
  <li>red item</li>
</ul>
```

The CSS Explained

We've applied the **:nth-child** selector with a value of **2n+2** which will select every second **<li>** in our list and make it red. The same could be achieved by setting the value to **li:nth-child(even)**.

:nth-last-child

The **:nth-last-child** pseudo class works exactly like **:nth-child**. The difference is this pseudo class counts from the bottom of the list up.

CSS

```
li:nth-last-child(2n+2) {
    color: red;
}
```

HTML

```
<ul>
  <li>red item</li>
  <li>item</li>
  <li>red item</li>
  <li>item</li>
  <li>red item</li>
  <li>item</li>
</ul>
```

The CSS Explained

In this example we've used the **:nth-last-child** selector with the same value. The result will be slightly different though because we will start counting from the bottom of the list. The result will be all the **<li>** tags will flip their colors.

:nth-of-type

The **:nth-of-type** pseudo class is used when you are trying to grab an element that exists at the same level as other elements. For example if you had a number of child elements that were paragraphs, divs, and images, but you only wanted to grab the even paragraphs, you would declare: **p:nth-of-type(even)**.

CSS

```
p:nth-of-type(even) {
    color: red;
}
```

HTML

```
<div>
  <p>para 1</p>
  <p>red para 2</p>
  <img src="#" />
  <p>para 3</p>
  <img src="#" />
  <p>red para 4</p>
</div>
```

The CSS Explained

In this case I'm using the **:nth-of-type** selector, with a value of **even** on all **<p>** tags in the parent ** **. This will result in the second and fourth paragraphs being red.

:nth-last-of-type

The **:nth-last-of-type** pseudo class works the same as the **:nth-of-type** class. The difference is it counts from the bottom of the list up.

CSS

```
p:nth-last-of-type(even) {
    color: red;
}
```

HTML

```
<div>
  <p>red para 1</p>
  <p>para 2</p>
  <img src="#" />
  <p>red para 3</p>
  <img src="#" />
  <p>para 4</p>
</div>
```

The CSS Explained

In this case I'm using the **:nth-last-of-type** selector, with a value of **even** on all **<p>** tags in the parent ** **. Since we are counting from the bottom up, the result will be that the third and first paragraphs will be colored red.

As you can see there has been a number of pseudo selectors added in CSS3. If you are feeling overwhelmed, I'd recommend first focusing on :first-child, :last-child, and simple uses of :nth-child with the odd or even keywords. They are the most commonly used and will likely be the most useful to you when getting started.

Transforms

Possibly the most confusing new feature in CSS3 is Transforms. The **transforms** property allows you to alter the two-dimensional or three-dimensional properties of an element. Both 2D and 3D come with their own set of properties which I'll cover in this chapter.

Browser Support

Currently the minimum browser support includes: IE10, Firefox 24, Chrome 29, Safari 5.1, Opera 17.

2D Transforms

There are a number of basic transforms you can do on 2D objects including: rotate, scale, translate, skew, and perspective. Let's run through a simple example of each type.

2D Rotate

Let's start with 2D transforms first. I'll show how to easily rotate an element using the rotate value.

CSS

```
.rotate {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: rotate(25deg);
    -moz-transform: rotate(25deg);
    transform: rotate(25deg);
}
```

HTML

```
<div class="rotate"></div>
```

The CSS Explained

- I've thrown in a **height**, **width** and **background** color for demo purposes only.
- The **transform** property has a value o**f rotate** which is set to **25 degrees**.

2D Scale

Using the **scale** property you can easily increase or decrease the size of an element.

CSS

```
.scale {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: scale(1.5);
    -moz-transform: scale(1.5);
    transform: scale(1.5);
}
```

HTML

```
<div class="scale"></div>
```

The CSS Explained

I've changed up the **transform** property value to **scale** and increased the size of my box to **1.5 times** it's original size. That will scale it up to 150px.

Complex Scaling

The previous example scales the X and Y axis of the element equally. It is possible to scale each axis differently with the **scaleX and scaleY** properties. However, keep in mind you cannot scale the X and Y values as separate **transform** properties on one element. If you would like to scale both on one property you need to use a comma delimited list of values. See below for an example of each type.

CSS

```
.scaleX {
    -webkit-transform: scaleX(2);
    -moz-transform: scaleX(2);
    transform: scaleX(2);
}

.scaleY {
    -webkit-transform: scaleY(3);
    -moz-transform: scaleY(3);
    transform: scaleY(3);
}

.scale-both {
    -webkit-transform: scale(2, 3);
    -moz-transform: scale(2, 3);
    transform: scale(2, 3);
}
```

2D Translate

The **translate** property is similar to using relative positioning to push an element in different directions without effecting the normal flow of the document. Using the **translateX** value will will change the position on the horizontal axis, while using the **translateY** value will change the vertical position of an element. As with the **scale** property, you have to set the X and Y values separately unless you use a comma delimited list of values. The values for pushing an element should be in pixel or percentage values.

CSS

```
.translateX {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: translateX(10px);
    -moz-transform: translateX(10px);
    transform: translateX(10px);
}

.translateY {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: translateY(20px);
    -moz-transform: translateY(20px);
    transform: translateY(20px);
}

/* cont... next page */
```

```
.translate-both {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: translate(10px, 20px);
    -moz-transform: translate(10px, 20px);
    transform: translate(10px, 20px);
}
```

HTML

```
<div class="translateX"></div>
<div class="translateY"></div>
<div class="translate-both"></div>
```

The CSS Explained

- I've setup 3 different classes. One for the X axis, one for the Y axis and one for both.
- **.translateX** will push the box down **10px**.
- **.translateY** will push the box to the left **20px**.
- **.translate-both** will push the box down **10px** and left **20px**.

> To move the box up or to the right, use a negative pixel value.

2D Skew

The **skew** property will distort an element on it's horizontal axis, vertical axis or both. The **skewX** property will distort the element on it's horizontal axis, while the **skewY** property will distort the element on the vertical axis. Like with the scale and translate property, you can distort both axis by using a comma delimited value for both. When setting skew values, you must use degrees. Pixel and percentage values are not valid with the **skew** property.

CSS

```
.skewX {
    -webkit-transform: skewX(15deg);
    -moz-transform: skewX(15deg);
    transform: skewX(15deg);
}

.skewY {
    -webkit-transform: skewY(-30deg);
    -moz-transform: skewY(-30deg);
    transform: skewY(-30deg);
}

.skew-both {
    -webkit-transform: skew(15deg, -30deg);
    -moz-transform: skew(15deg, -30deg);
    transform: skew(15deg, -30deg);
}
```

HTML

```
<div class="skewX"></div>
<div class="skewY"></div>
<div class="skew-both"></div>
```

The CSS Explained

- I've setup 3 different classes. One for the X axis, one for the Y axis and one for both.
- **.skewX** will skew the box's X axis by **15 degrees**.
- **.skewY** will skew the box'x Y axis by **-30 degrees**.
- **.skew-both** will skew the box's X axis by **15 degrees** and the Y axis by **-30 degrees.**

Transform Origin

By default, the transform origin is the dead center of the element. To change the origin point, you can use the **transform-origin** property. The property will accept a list of keywords (**top**, **left**, **bottom**, **right**), pixel, or percentage values. If only one value is set, it will be applied to both the X and Y axis. However, you can specify each value individually in one property declaration.

CSS

```
.origin1 {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: scale(0.5);
    -moz-transform: scale(0.5);
    transform: scale(0.5);

    -webkit-transform-origin: 0;
    -moz-transform-origin: 0;
    transform-origin: 0;
}

.origin2 {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: scale(0.5);
    -moz-transform: scale(0.5);
    transform: scale(0.5);

    -webkit-transform-origin: 100% 100%;
    -moz-transform-origin: 100% 100%;
    transform-origin: 100% 100%;
}
```

```
/* cont... from previous page */

.origin3 {
   width: 100px;
   height: 100px;
   background: #bdc3c7;

   -webkit-transform: scale(0.5);
   -moz-transform: scale(0.5);
   transform: scale(0.5);

   -webkit-transform-origin: top right;
   -moz-transform-origin: top right;
   transform-origin: top right;
}
```

HTML

```
<div class="origin1"></div>
<div class="origin2"></div>
<div class="origin3"></div>
```

The CSS Explained

- I've setup three classes to define three different origin points.
- **.origin1** places the origin point in the top left corner. Both values are 0.
- **.origin2** places the origin point in the bottom left corner.
- **.origin3** places the origin point in the top right corner.

3D Transforms

3D Perspective

To get three-dimensional transforms working on an element, we need to include a perspective on which to transform them. The perspective of an element is set by including the **perspective** value in the **transform** property declaration.

CSS

```
.perspective {
   width: 100px;
   height: 100px;
   background: #bdc3c7;

   -webkit-transform: perspective(100px) rotateX(45deg);
}
```

HTML

```
<div class="perspective"></div>
```

The CSS Explained

- The **perspective** value is set to **100px**. The larger the number the farther away the perspective appears.
- I've rotated the **X horizontal axis by 45 degrees** to give the illusion of a shadow.

3D Rotate

Building off what we learned in 2D rotations, using 3D will allow us to rotate an element around any axis. To accomplish this, we have three new properties: **rotateX, rotateY, and rotateZ**.

CSS

```
.rotateX {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: perspective(100px) rotateX(45deg);
    -moz-transform: perspective(100px) rotateX(45deg);
    transform: perspective(100px) rotateX(45deg);
}

.rotateY {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-transform: perspective(100px) rotateY(45deg);
    -moz-transform: perspective(100px) rotateY(45deg);
    transform: perspective(100px) rotateY(45deg);
}

/* cont... next page */
```

```
.rotateZ {
    width: 100px;
    height: 100px;
    background:#bdc3c7;

    -webkit-transform: perspective(100px) rotateZ(45deg);
    -moz-transform: perspective(100px) rotateZ(45deg);
    transform: perspective(100px) rotateZ(45deg);
}
```

HTML

```
<div class="rotateX"></div>
<div class="rotateY"></div>
<div class="rotateZ"></div>
```

The CSS Explained

- I've setup a separate class for each type of rotation.
- **.rotateX** sets the perspective to **100px** and rotates the X axis by **45 degrees**. The result is very similar to the previous perspective example.
- **.rotateY** rotates our box **45 degrees** on the Y axis. The result is the box is skewed to the left.
- **.rotateZ** rotates are box **45 degrees** on the Z axis. This will result in spinning the box by **45 degrees**. This is similar to a 2D rotate.

3D Scale

Using the **scaleZ** value, we can scale the size of the Z axis up and down. However, to see this in action, we need to apply another value like rotate. On a flat, 2D box, you will not see any change when applying the **scaleZ** value.

CSS

```
.scaleZ {
   width: 100px;
   height: 100px;
   background: #bdc3c7;


   -webkit-transform: perspective(200px) scaleZ(2)
rotateX(45deg);
   -moz-transform: perspective(200px) scaleZ(2)
rotateX(45deg);
   transform: perspective(200px) scaleZ(2)
rotateX(45deg);
}
```

HTML

```
<div class=”scaleZ”></div>
```

The CSS Explained

We’ve taken the CSS from the previous example and added the **scaleZ** value and set it to **2** which will double the size of the box. Notice the preview is similar to the previous example but the box is now twice as long.

3D Translate

Using the **translateZ** value, we can scale the size of an element up or down from the middle of the element. This is different from the 2D scale value that changes the size of the element based on the X or Y axis. This value is quite handy if you want to scale an item down and have equal space around it.

CSS

```
.translateZ {
 width: 100px;
 height: 100px;
 background: #bdc3c7;

 -webkit-transform: perspective(100px) translateZ(-50px);
 -moz-transform: perspective(100px) translateZ(-50px);
 transform: perspective(100px) translateZ(-50px);
}
```

HTML

```
<div class="translateZ"></div>
```

The CSS Explained

- I've set the perspective of the transform to **100px**.
- The **translateZ** value has been set to **-50px** which will reduce the size of the box by 50px in total. That means it will remove 25px for each side of our box since we are reducing from the middle or Z axis. Only pixel values will work here.

3D Skew

the **skew** value is the only one that cannot actually be transformed on a three-dimensional scale. Elements may only be skewed on the X and Y axis.

Transform Style

It's possible to get into a scenario where a three-dimensional transformed element is nested with another transformed element. When this happens, you need to apply the **transform-style** property to a wrapping element to preserve the 3D transform within. The **preserve-3d** value is applied to the **transform-style** property.

CSS

```
.wrapper {
   -webkit-transform-style: preserve-3d;
   -moz-transform-style: preserve-3d;
   transform-style: preserve-3d;
}


.box {
   -webkit-transform: perspective(200px) scaleZ(2)
rotateX(45deg);
   -moz-transform: perspective(200px) scaleZ(2)
rotateX(45deg);
   transform: perspective(200px) scaleZ(2)
rotateX(45deg);
}
```

HTML

```
<div class=”wrapper”>
    <div class=”box”></div>
</div>
```

Backface Visibility

It is also possible to get into a scenario when working with 3D transforms, where the element will be facing away from the screen. For example, if you have some text in your box and the rotate is set to 180 degrees, the text will be backwards. When this happens, you can use the backface-visibilty property to hide the element from the screen. The values the property takes are **hidden or visible**, the default being visible. The property should be applied to the same element that the transform is on.

CSS

```
.visibility {
    width: 100px;
    height: 100px;
    background: #bdc3c7;

    -webkit-backface-visibility: hidden;
    -moz-backface-visibility: hidden;
    backface-visibility: hidden;

    -webkit-transform: rotateY(180deg);
    -moz-transform: rotateY(180deg);
    transform: rotateY(180deg);
}
```

HTML

```
<div class="visibility"></div>
```

Well I believe that is the longest chapter in the book. I hope you we're able to follow along easily. Transforms are definitely one of the most complicated parts of CSS3. However, they are also one of the most fun to play around with. I'd encourage you to experiment with transforms on state changes to create some nice effects for your websites.