Chapter 27: Animations

Transition

Parameter	Details
property	Either the CSS property to transition on, or `all`, which specifies all transition-able properties.
duration	Transition time, either in seconds or milliseconds.
timing-function	Specifies a function to define how intermediate values for properties are computed. Common values are `ease`, `linear`, and `step-end`. Check out the easing function cheat-sheet for more.
delay	Amount of time, in seconds or milliseconds, to wait before playing the animation.

@keyframes

[from | to | `<percentage>`]	You can either specify a set time with a percentage value, or two percentage values, ie `10%, 20%`, for a period of time where the keyframe's set attributes are set.
block	Any amount of CSS attributes for the keyframe.

Section 27.1: Animations with keyframes

For multi-stage CSS animations, you can create CSS `@keyframes`. Keyframes allow you to define multiple animation points, called a keyframe, to define more complex animations.

Basic Example

In this example, we'll make a basic background animation that cycles between all colors.

```
@keyframes rainbow-background {
    0%      { background-color: #ff0000; }
    8.333%  { background-color: #ff8000; }
    16.667% { background-color: #ffff00; }
    25.000% { background-color: #80ff00; }
    33.333% { background-color: #00ff00; }
    41.667% { background-color: #00ff80; }
    50.000% { background-color: #00ffff; }
    58.333% { background-color: #0080ff; }
    66.667% { background-color: #0000ff; }
    75.000% { background-color: #8000ff; }
    83.333% { background-color: #ff00ff; }
    91.667% { background-color: #ff0080; }
    100.00% { background-color: #ff0000; }
}

.RainbowBackground {
    animation: rainbow-background 5s infinite;
}
```

View Result

There's a few different things to note here. First, the actual `@keyframes` syntax.

```
@keyframes rainbow-background{
```

This sets the name of the animation to `rainbow-background`.

```
0%      { background-color: #ff0000; }
```

This is the definition for a keyframe within the animation. The first part, the `0%` in the case, defines where the keyframe is during the animation. The `0%` implies it is 0% of the total animation time from the beginning.

The animation will automatically transition between keyframes. So, by setting the next background color at `8.333%`, the animation will smoothly take 8.333% of the time to transition between those keyframes.

```
.RainbowBackground  {
    animation: rainbow-background 5s infinite;
}
```

This code attaches our animation to all elements which have the `.RainbowBackground` class.

The actual animation property takes the following arguments.

- **animation-name**: The name of our animation. In this case, `rainbow-background`
- **animation-duration**: How long the animation will take, in this case 5 seconds.
- **animation-iteration-count (Optional)**: The number of times the animation will loop. In this case, the animation will go on indefinitely. By default, the animation will play once.
- **animation-delay (Optional)**: Specifies how long to wait before the animation starts. It defaults to 0 seconds, and can take negative values. For example, `-2s` would start the animation 2 seconds into its loop.
- **animation-timing-function (Optional)**: Specifies the speed curve of the animation. It defaults to `ease`, where the animation starts slow, gets faster and ends slow.

In this particular example, both the `0%` and `100%` keyframes specify `{ background-color: #ff0000; }`. Wherever two or more keyframes share a state, one may specify them in a single statement. In this case, the two `0%` and `100%` lines could be replaced with this single line:

```
0%, 100%      { background-color: #ff0000; }
```

Cross-browser compatibility

For older WebKit-based browsers, you'll need to use the vendor prefix on both the `@keyframes` declaration and the `animation` property, like so:

```
@-webkit-keyframes {}

-webkit-animation: ...
```

Section 27.2: Animations with the transition property

Useful for simple animations, the CSS `transition` property allows number-based CSS properties to animate between states.

Example

```
.Example{
    height: 100px;
    background: #fff;
}

.Example:hover{
    height: 120px;
```

```
    background: #ff0000;
}
```

View Result

By default, hovering over an element with the `.Example` class would immediately cause the element's height to jump to `120px` and its background color to red (`#ff0000`).

By adding the `transition` property, we can cause these changes to occur over time:

```
.Example{
    ...
    transition: all 400ms ease;
}
```

View Result

The `all` value applies the transition to all compatible (numbers-based) properties. Any compatible property name (such as `height` or `top`) can be substituted for this keyword.

`400ms` specifies the amount of time the transition takes. In this case, the element's change in height will take 400 milliseconds to complete.

Finally, the value `ease` is the animation function, which determines how the animation is played. `ease` means start slow, speed up, then end slow again. Other values are `linear`, `ease-out`, and `ease-in`.

Cross-Browser Compatibility

The `transition` property is generally well-supported across all major browsers, excepting IE 9. For earlier versions of Firefox and Webkit-based browsers, use vendor prefixes like so:

```
.Example{
    transition:         all 400ms ease;
    -moz-transition:    all 400ms ease;
    -webkit-transition: all 400ms ease;
}
```

Note: The `transition` property can animate changes between any two numerical values, regardless of unit. It can also transition between units, such as `100px` to `50vh`. However, it cannot transition between a number and a default or automatic value, such as transitioning an element's height from `100px` to `auto`.

Section 27.3: Syntax Examples

Our first syntax example shows the animation shorthand property using all of the available properties/parameters:

```
  animation: 3s ease-in 1s 2 reverse both paused
      slidein;
  /* duration | timing-function | delay | iteration-count | direction | fill-mode | play- state |
name */
```

Our second example is a little more simple, and shows that some properties can be omitted:

```
  animation: 3s linear 1s slidein;
  /* duration | timing-function | delay | name */
```

Our third example shows the most minimal declaration. Note that the animation-name and animation-duration must be declared:

```
animation: 3s              slidein;
/*         duration | name */
```

It's also worth mentioning that when using the animation shorthand the order of the properties makes a difference. Obviously the browser may confuse your duration with your delay.

If brevity isn't your thing, you can also skip the shorthand property and write out each property individually:

```
animation-duration: 3s;
animation-timing-function: ease-in;
animation-delay: 1s;
animation-iteration-count: 2;
animation-direction: reverse;
animation-fill-mode: both;
animation-play-state: paused;
animation-name: slidein;
```

Section 27.4: Increasing Animation Performance Using the `will-change` Attribute

When creating animations and other GPU-heavy actions, it's important to understand the `will-change` attribute.

Both CSS keyframes and the `transition` property use GPU acceleration. Performance is increased by offloading calculations to the device's GPU. This is done by creating paint layers (parts of the page that are individually rendered) that are offloaded to the GPU to be calculated. The `will-change` property tells the browser what will animate, allowing the browser to create smaller paint areas, thus increasing performance.

The `will-change` property accepts a comma-separated list of properties to be animated. For example, if you plan on transforming an object and changing its opacity, you would specify:

```
.Example{
    ---
    will-change: transform, opacity;
}
```

Note: Use `will-change` sparingly. Setting `will-change` for every element on a page can cause performance problems, as the browser may attempt to create paint layers for every element, significantly increasing the amount of processing done by the GPU.