CSS Transitions

W3C Working Draft 19 November 2013

This version:
http://www.w3.org/TR/2013/WD-css3-transitions-20131119/
Latest version:
http://www.w3.org/TR/css3-transitions/
Editor's draft:
http://dev.w3.org/csswg/css-transitions/ (change log, older change log)
Previous version:
http://www.w3.org/TR/2013/WD-css3-transitions-20130212/
Editors:
Dean Jackson (Apple Inc)
David Hyatt (Apple Inc)
Chris Marrin (Apple Inc)
L. David Baron (Mozilla)
Issues list:
in Bugzilla
Feedback:
www-style@w3.org with subject line "`[css-transitions] … message topic …`" (archives)
Test suite:
http://test.csswg.org/suites/css-transitions-1/nightly-unstable/

Abstract

CSS Transitions allows property changes in CSS values to occur smoothly over a specified duration.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The (archived) public mailing list www-style@w3.org (see instructions) is preferred for discussion of this specification. When sending e-mail, please put the text "css3-transitions" in the subject, preferably like this: "[css3-transitions] *…summary of comment…*"

This document was produced by the CSS Working Group (part of the Style Activity).

W3C Working Draft

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is expected to be relatively close to last call. While some issues raised have yet to be addressed, new features are extremely unlikely to be considered for this level.

Table of Contents

1. Introduction

This section is not normative.

This document introduces new CSS features to enable *implicit transitions*, which describe how CSS properties can

be made to change smoothly from one value to another over a given duration.

2. Transitions

Normally when the value of a CSS property changes, the rendered result is instantly updated, with the affected elements immediately changing from the old property value to the new property value. This section describes a way to specify transitions using new CSS properties. These properties are used to animate smoothly from the old state to the new state over time.

For example, suppose that transitions of one second have been defined on the 'left' and 'background-color' properties. The following diagram illustrates the effect of updating those properties on an element, in this case moving it to the right and changing the background from red to blue. This assumes other transition parameters still have their default values.

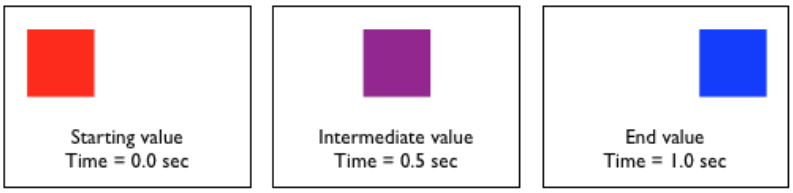

Figure 1. *Transitions of 'left' and 'background-color'*

Transitions are a presentational effect. The computed value of a property transitions over time from the old value to the new value. Therefore if a script queries the computed style of a property as it is transitioning, it will see an intermediate value that represents the current animated value of the property.

Only animatable CSS properties can be transitioned. See the table at the end of this document for a list of properties that are animatable.

The transition for a property is defined using a number of new properties. For example:

EXAMPLE 1

```
div {
  transition-property: opacity;
  transition-duration: 2s;
}
```

The above example defines a transition on the 'opacity' property that, when a new value is assigned to it, will cause a smooth change between the old value and the new value over a period of two seconds.

Each of the transition properties accepts a comma-separated list, allowing multiple transitions to be defined, each acting on a different property. In this case, the individual transitions take their parameters from the same index in all the lists. For example:

EXAMPLE 2

```
div {
  transition-property: opacity, left;
  transition-duration: 2s, 4s;
}
```

This will cause the 'opacity' property to transition over a period of two seconds and the left property to transition over a period of four seconds.

In the case where the lists of values in transition properties do not have the same length, the length of the 'transition-property' list determines the number of items in each list examined when starting transitions. The lists are matched up from the first value: excess values at the end are not used. If one of the other properties doesn't have enough comma-separated values to match the number of values of 'transition-property', the UA must calculate its used value by repeating the list of values until there are enough. This truncation or repetition does not affect the computed value. Note: This is analogous to the behavior of the 'background-*' properties, with 'background-image' analogous to 'transition-property'.

EXAMPLE 3

```
div {
  transition-property: opacity, left, top, width;
  transition-duration: 2s, 1s;
}
```

The above example defines a transition on the 'opacity' property of 2 seconds duration, a transition on the 'left' property of 1 second duration, a transition on the 'top' property of 2 seconds duration and a transition on the 'width' property of 1 second duration.

While authors can use transitions to create dynamically changing content, dynamically changing content can lead to seizures in some users. For information on how to avoid content that can lead to seizures, see Guideline 2.3: Seizures: Do not design content in a way that is known to cause seizures ([WCAG20]).

2.1. The 'transition-property' Property

The 'transition-property' property specifies the name of the CSS property to which the transition is applied.

Name:	***transition-property***
Value:	none | <single-transition-property> [',' <single-transition-property>]*
Initial:	all
Applies to:	all elements, :before and :after pseudo elements
Inherited:	no
Animatable:	no
Percentages:	N/A
Media:	visual
Computed value:	Same as specified value.
Canonical order:	per grammar

<single-transition-property> = all | <IDENT>

A value of 'none' means that no property will transition. Otherwise, a list of properties to be transitioned, or the keyword 'all' which indicates that all properties are to be transitioned, is given.

If one of the identifiers listed is not a recognized property name or is not an animatable property, the implementation must still start transitions on the animatable properties in the list using the duration, delay, and timing function at their respective indices in the lists for 'transition-duration', 'transition-delay', and 'transition-timing-function'. In other words, unrecognized or non-animatable properties must be kept in the list to preserve the matching of indices.

The keywords 'none', 'inherit', and 'initial' are not permitted as items within a list of more that one identifier; any list that uses them is syntactically invalid. In other words, the <IDENT> production in <single-transition-property> matches any identifier other than these three keywords.

For the keyword 'all', or if one of the identifiers listed is a shorthand property, implementations must start transitions for any of its longhand sub-properties that are animatable (or, for 'all', all animatable properties), using the duration, delay, and timing function at the index corresponding to the shorthand.

If a property is specified multiple times in the value of 'transition-property' (either on its own, via a shorthand that contains it, or via the 'all' value), then the transition that starts uses the duration, delay, and timing function at the index corresponding to the *last* item in the value of 'transition-property' that calls for animating that property.

Note: The 'all' value and 'all' shorthand property work in similar ways, so the 'all' value is just like a shorthand that covers all properties.

2.2. The 'transition-duration' Property

The 'transition-duration' property defines the length of time that a transition takes.

Name:	***transition-duration***
Value:	<time> [, <time>]*
Initial:	0s
Applies to:	all elements, :before and :after pseudo elements
Inherited:	no
Animatable:	no
Percentages:	N/A
Media:	interactive
Computed value:	Same as specified value.
Canonical order:	per grammar

This property specifies how long the transition from the old value to the new value should take. By default the value is '0s', meaning that the transition is immediate (i.e. there will be no animation). A negative value for 'transition-duration' renders the declaration invalid.

2.3. The 'transition-timing-function' Property

The 'transition-timing-function' property describes how the intermediate values used during a transition will be calculated. It allows for a transition to change speed over its duration. These effects are commonly called *easing* functions. In either case, a mathematical function that provides a smooth curve is used.

Timing functions are either defined as a stepping function or a cubic Bézier curve. The timing function takes as its input the current elapsed percentage of the transition duration and outputs the percentage of the way the transition is from its start value to its end value. How this output is used is defined by the interpolation rules for the value type.

A stepping function is defined by a number that divides the domain of operation into equally sized intervals. Each subsequent interval is a equal step closer to the goal state. The function also specifies whether the change in output percentage happens at the start or end of the interval (in other words, if 0% on the input percentage is the point of initial change).

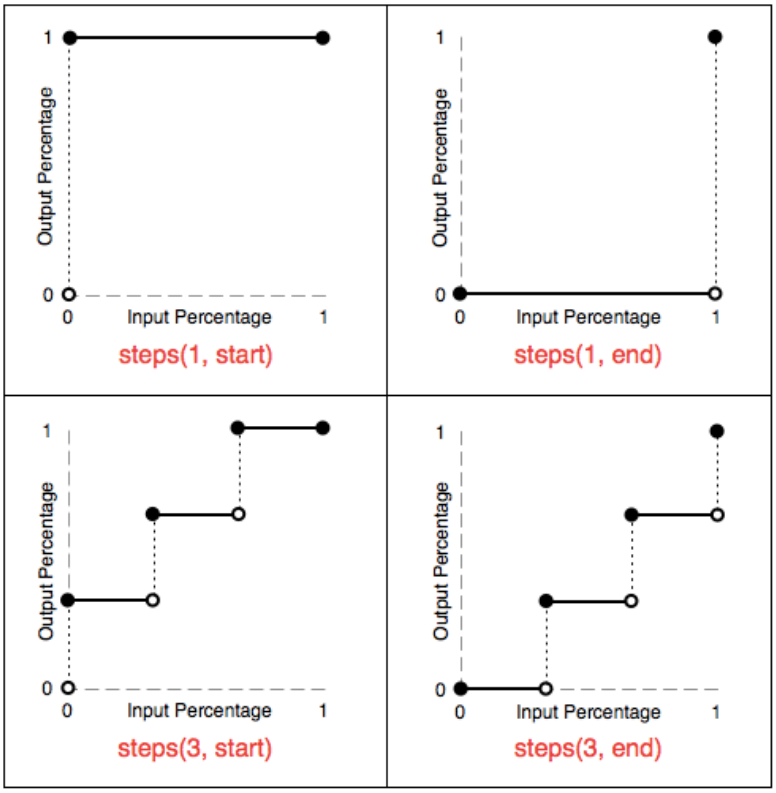

Figure 2. *Step timing functions*

A cubic Bézier curve is defined by four control points, P_0 through P_3 (see Figure 1). P_0 and P_3 are always set to (0,0) and (1,1). The 'transition-timing-function' property is used to specify the values for points P_1 and P_2. These can be set to preset values using the keywords listed below, or can be set to specific values using the 'cubic-bezier' function. In the 'cubic-bezier' function, P_1 and P_2 are each specified by both an X and Y value.

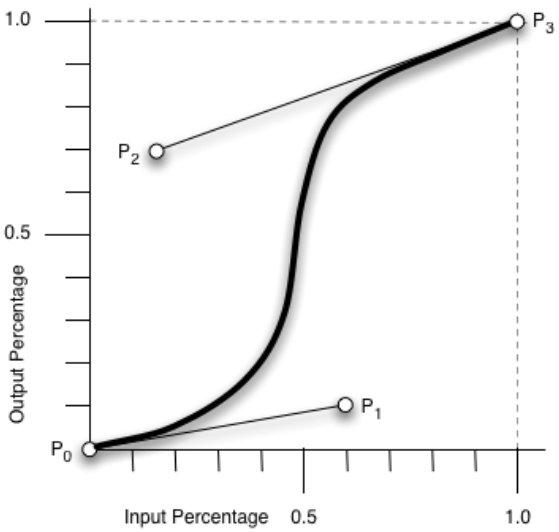

Figure 3. *Bézier Timing Function Control Points*

Name:	***transition-timing-function***
Value:	<single-transition-timing-function> [',' <single-transition-timing-function>]*
Initial:	ease
Applies to:	all elements, :before and :after pseudo elements
Inherited:	no
Animatable:	no
Percentages:	N/A
Media:	interactive
Computed value:	Same as specified value.
Canonical order:	per grammar

<single-transition-timing-function> = ease | linear | ease-in | ease-out | ease-in-out | step-start | step-end | steps(<integer>[, [start | end]]?) | cubic-bezier(<number>, <number>, <number>, <number>)

The timing functions have the following definitions.

ease
The ease function is equivalent to cubic-bezier(0.25, 0.1, 0.25, 1).

linear
The linear function is equivalent to cubic-bezier(0, 0, 1, 1).

ease-in
The ease-in function is equivalent to cubic-bezier(0.42, 0, 1, 1).

ease-out
The ease-out function is equivalent to cubic-bezier(0, 0, 0.58, 1).

ease-in-out
The ease-in-out function is equivalent to cubic-bezier(0.42, 0, 0.58, 1)

step-start
The step-start function is equivalent to steps(1, start).

step-end
The step-end function is equivalent to steps(1, end).

steps(<integer>[, [start | end]]?)
Specifies a stepping function, described above, taking two parameters. The first parameter specifies the number of intervals in the function. It must be a positive integer (greater than 0). The second parameter, which is optional, is either the value 'start' or 'end', and specifies the point at which the change of values occur within the interval. If the second parameter is omitted, it is given the value 'end'.

cubic-bezier(<number>, <number>, <number>, <number>)
Specifies a cubic-bezier curve. The four values specify points P_1 and P_2 of the curve as (x1, y1, x2, y2). Both x values must be in the range [0, 1] or the definition is invalid. The y values can exceed this range.

2.4. The ‘transition-delay’ Property

The ‘transition-delay’ property defines when the transition will start. It allows a transition to begin execution some some period of time from when it is applied. A ‘transition-delay’ value of ‘0s’ means the transition will execute as soon as the property is changed. Otherwise, the value specifies an offset from the moment the property is changed, and the transition will delay execution by that offset.

If the value for ‘transition-delay’ is a negative time offset then the transition will execute the moment the property is changed, but will appear to have begun execution at the specified offset. That is, the transition will appear to begin part-way through its play cycle. In the case where a transition has implied starting values and a negative ‘transition-delay’, the starting values are taken from the moment the property is changed.

Name:	***transition-delay***
Value:	<time> [, <time>]*
Initial:	0s
Applies to:	all elements, :before and :after pseudo elements
Inherited:	no
Animatable:	no
Percentages:	N/A
Media:	interactive
Computed value:	Same as specified value.
Canonical order:	per grammar

2.5. The ‘transition’ Shorthand Property

The ‘transition’ shorthand property combines the four properties described above into a single property.

Name:	***transition***
Value:	<single-transition> [‘,’ <single-transition>]*
Initial:	see individual properties
Applies to:	all elements, :before and :after pseudo elements
Inherited:	no
Animatable:	no
Percentages:	N/A
Media:	interactive
Computed value:	see individual properties
Canonical order:	per grammar

<single-transition> = [none | <single-transition-property>] || <time> || <single-transition-timing-function> || <time>

Note that order is important within the items in this property: the first value that can be parsed as a time is assigned to the transition-duration, and the second value that can be parsed as a time is assigned to transition-delay.

If there is more than one <single-transition> in the shorthand, and any of the transitions has 'none' as the <single-transition-property>, then the declaration is invalid.

3. Starting of transitions

When the computed value of an animatable property changes, implementations must decide what transitions to start based on the values of the 'transition-property', 'transition-duration', 'transition-timing-function', and 'transition-delay' properties at the time the animatable property would first have its new computed value. This means that when one of these 'transition-*' properties changes at the same time as a property whose change might transition, it is the *new* values of the 'transition-*' properties that control the transition.

EXAMPLE 4

This provides a way for authors to specify different values of the 'transition-*' properties for the "forward" and "reverse" transitions (but see below for special reversing behavior when an *incomplete* transition is interrupted). Authors can specify the value of 'transition-duration', 'transition-timing-function', or 'transition-delay' in the same rule where they specify the value that triggers the transition, or can change these properties at the same time as they change the property that triggers the transition. Since it's the new values of these 'transition-*' properties that affect the transition, these values will be used for the transitions *to* the associated transitioning values. For example:

```
li {
  transition: background-color linear 1s;
  background: blue;
}
li:hover {
  background-color: green;
  transition-duration: 2s; /* applies to the transition *to* the :hover state */
}
```

When a list item with these style rules enters the :hover state, the computed 'transition-duration' at the time that 'background-color' would have its new value ('green') is '2s', so the transition from 'blue' to 'green' takes 2 seconds. However, when the list item leaves the :hover state, the transition from 'green' to 'blue' takes 1 second.

Various things can cause the computed style of an element to change, or for an element to start or stop having computed style. (For the purposes of this specification, an element has computed style when it is in the document tree, and does not have computed style when it is not in the document tree.) These include insertion and removal of elements from the document tree (which both changes whether those elements have computed styles and can change the styles of other elements through selector matching), changes to the document tree that cause changes to which selectors match elements, changes to style sheets or style attributes, and other things. This specification does not define when computed styles are updated. However, when an implementation updates the computed style for an element to reflect one of these changes, it must update the computed style for all elements to reflect all of these changes at the same time (or at least it must be undetectable that it was done at a different time). This processing of a set of simultaneous style changes is called a ***style change event***. (Implementations typically have a style change event to correspond with their desired screen refresh rate, and when up-to-date computed style is needed for a script API that depends on it.)

Since this specification does not define when a style change event occurs, and thus what changes to computed values are considered simultaneous, authors should be aware that changing any of the transition properties a small amount of time after making a change that might transition can result in behavior that varies between implementations, since the changes might be considered simultaneous in some implementations but not others.

When a style change event occurs, implementations must start transitions based on the computed styles that changed in that event. If an element does not have a computed style either before or after the style change event, then transitions are not started for that element in that style change event. Otherwise, define the ***before-change style*** as the computed style for the element as of the previous style change event, except with any styles derived from declarative animations such as CSS Transitions, CSS Animations ([CSS3-ANIMATIONS]), and SMIL Animations ([SMIL-ANIMATION], [SVG11]) updated to the current time. Likewise, define the ***after-change style*** as the computed style for the element based on the information known at the start of that style change event, in other words, excluding any changes resulting from CSS Transitions that start during that style change event.

ISSUE 1 This wording needs to handle already-running transitions better! Need to cancel a transition that hasn't moved yet when we're resetting to its start value! Define cancelling as not firing transition events. And point to other occurrence of cancelling in reversing section.

Note that this definition of the after-change style means that a single change can start a transition on the same property on both an ancestor element and its descendant element. This can happen when a property change is inherited from one element with 'transition-*' properties that say to animate the changing property to another element with 'transition-*' properties that also say to animate the changing property.

When this happens, both transitions will run, and the transition on the descendant will override the transition on the ancestor because of the normal CSS cascading and inheritance rules ([CSS3CASCADE]).

If the transition on the descendant completes before the transition on the ancestor, the descendant will then resume inheriting the (still transitioning) value from its parent. This effect is likely not a desirable effect, but it is essentially doing what the author asked for.

For each element with a before-change style and an after-change style, and each property (other than shorthands) for which the before-change style is different from the after-change style, implementations must start transitions based on the relevant item (see the definition of 'transition-property') in the computed value of 'transition-property'. Corresponding to this item there is a matching transition duration, a matching transition delay, and a matching transition timing function in the computed values of 'transition-duration', 'transition-delay', and 'transition-timing-function' (see the rules on matching lists). Define the ***combined duration*** of the transition as the sum of max(matching transition duration, '0s') and the matching transition-delay. When the combined duration is greater than '0s', then a transition starts based on the values of the matching transition duration, the matching transition delay, and the matching transition-timing-function; in other cases transitions do not occur. The ***start time*** of this transition is defined as the time of the style change event plus the matching transition delay. The ***end time*** of this transition is defined as the start time plus the matching transition duration. The ***start value*** of this transition is defined as the value of the transitioning property in the before-change style, and the ***end value*** of this transition is defined as the value of the transitioning property in the after-change style. Except in the cases described in the section on reversing of transitions, the ***reversing-adjusted start value*** is the same as the start value, and the ***reversing shortening factor*** is 1.

Once the transition of a property has started, it must continue running based on the original timing function, duration, and delay, even if the 'transition-timing-function', 'transition-duration', or 'transition-delay' property changes before the transition is complete. However, if the 'transition-property' property changes such that the transition would not have started, the transition must stop (and the property must immediately change to its final value).

Implementations must not start a transition when the computed value of a property changes as a result of

declarative animation (as opposed to scripted animation).

Implementations also must not start a transition when the computed value changes because it is inherited (directly or indirectly) from another element that is transitioning the same property.

3.1. Automatically reversing interrupted transitions

Many common transitions effects involve transitions between two states, such as the transition that occurs when the mouse pointer moves over a user interface element, and then later moves out of that element. With these effects, it is common for a running transition to be interrupted before it completes, and the property reset to the starting value of that transition. An example is a hover effect on an element, where a transition starts when the pointer enters the element, and then the pointer exits the element before the effect has completed. If the outgoing and incoming transitions are executed using their specified durations and timing functions, the resulting effect can be distractingly asymmetric because the second transition takes the full specified time to move a shortened distance. Instead, the expected behavior is that the second transition is shorter.

To meet this expectation, when a transition is started for a property on an element (henceforth, the ***new transition***) that has a currently-running transition whose reversing-adjusted start value is the same as the end value of the new transition (henceforth, the ***old transition***), implementations must cancel the old transition ISSUE 2 link to definition above and adjust the new transition as follows (prior to following the rules for computing the combined duration, start time, and end time):

1. The reversing-adjusted start value of the new transition is instead the the end value of the old transition.
   Note: This represents the logical start state of the transition, and allows some calculations to ignore that the transition started before that state was reached, which in turn allows repeated reversals of the same transition to work correctly.
2. The reversing shortening factor of the new transition is the absolute value, clamped to the range [0, 1], of the sum of:
   1. the output of the timing function of the old transition at the time of the style change event, times the reversing shortening factor of the old transition
   2. 1 minus the reversing shortening factor of the old transition.

   Note: This represents the portion of the space between the reversing-adjusted start value and the end value that the old transition has traversed (in amounts of the value, not time), except with the absolute value and clamping to handle timing functions that have y1 or y2 outside the range [0, 1].
3. The matching transition-duration for the new transition is multiplied by the reversing shortening factor.
4. If the matching transition-delay for the new transition is negative, it is also multiplied by the reversing shortening factor.

Note that these rules do not fully address the problem for transition patterns that involve more than two states.

Note that these rules lead to the entire timing function of the new transition being used, rather than jumping into the middle of a timing function, which can create a jarring effect.

This was one of several possibilities that was considered by the working group. See the reversing demo demonstrating a number of them, leading to a working group resolution made on 2013-06-07 and edits made on 2013-11-11.

4. Application of transitions

When a property on an element is undergoing a transition (that is, when or after the transition has started and before the end time of the transition) the transition adds a style to the CSS cascade at the level defined for CSS Transitions in [CSS3CASCADE].

> Note that this means that computed values resulting from CSS transitions can inherit to descendants just like any other computed values. In the normal case, this means that a transition of an inherited property applies to descendant elements just as an author would expect.

Implementations must add this value to the cascade if and only if that property is not currently undergoing a CSS Animation ([CSS3-ANIMATIONS]) on the same element.

> Note that this behavior of transitions not applying to the cascade when an animation on the same element and property is running does not affect whether the transition has started or ended. APIs that detect whether transitions are running (such as transition events) still report that a transition is running.

If the current time is at or before the start time of the transition (that is, during the delay phase of the transition), this value is a specified style that will compute to the start value of the transition.

If the current time is after the start time of the transition (that is, during the duration phase of the transition), this value is a specified style that will compute to the result of interpolating the property using the start value of the transition as V_{start}, using the end value of the transition as V_{end}, and using (current time - start time) / (end time - start time) as the input to the timing function.

5. Transition Events

The completion of a CSS Transition generates a corresponding DOM Event. An event is fired for each property that undergoes a transition. This allows a content developer to perform actions that synchronize with the completion of a transition.

Each event provides the name of the property the transition is associated with as well as the duration of the transition.

Interface *TransitionEvent*

The `TransitionEvent` interface provides specific contextual information associated with transitions.

IDL Definition

```
[Constructor(DOMString type, optional TransitionEventInit transitionEventInitDict)]
interface TransitionEvent : Event {
  readonly attribute DOMString          propertyName;
  readonly attribute float              elapsedTime;
  readonly attribute DOMString          pseudoElement;
};

dictionary TransitionEventInit : EventInit {
  DOMString propertyName = "";
  float elapsedTime = 0.0;
  DOMString pseudoElement = "";
}
```

Attributes

***propertyName* of type `DOMString`, readonly**

The name of the CSS property associated with the transition.

***elapsedTime* of type `float`, readonly**
The amount of time the transition has been running, in seconds, when this event fired. Note that this value is not affected by the value of transition-delay.

***pseudoElement* of type `DOMString`, readonly**
The name (beginning with two colons) of the CSS pseudo-element on which the transition occured (in which case the target of the event is that pseudo-element's corresponding element), or the empty string if the transition occurred on an element (which means the target of the event is that element).

`TransitionEvent(type, transitionEventInitDict)` is an event constructor.

There is one type of transition event available.

transitionend
The `transitionend` event occurs at the completion of the transition. In the case where a transition is removed before completion, such as if the transition-property is removed, then the event will not fire.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: propertyName, elapsedTime, pseudoElement

6. Animation of property types

When interpolating between two values, V_{start} and V_{end}, interpolation is done using the output p of the timing function, which gives the portion of the value space that the interpolation has crossed. Thus the result of the interpolation is $V_{res} = (1 - p) \cdot V_{start} + p \cdot V_{end}$.

However, if this value (V_{res}) is outside the allowed range of values for the property, then it is clamped to that range. This can occur if p is outside of the range 0 to 1, which can occur if a timing function is specified with a *y1* or *y2* that is outside the range 0 to 1.

The following describes how each property type undergoes transition or animation.

- **color**: interpolated via red, green, blue and alpha components (treating each as a number, see below). The interpolation is done between premultiplied colors (that is, colors for which the red, green, and blue components specified have been multiplied by the alpha).
- **length**: interpolated as real numbers.
- **percentage**: interpolated as real numbers.
- **length, percentage, or calc**: when both values are lengths, interpolated as lengths; when both values are percentages, interpolated as percentages; otherwise, both values are converted into a 'calc()' function that is the sum of a length and a percentage (each possibly zero), and these 'calc()' functions have each half interpolated as real numbers.
- **integer**: interpolated via discrete steps (whole numbers). The interpolation happens in real number space and is converted to an integer by rounding to the nearest integer, with values halfway between a pair of integers rounded towards positive infinity.
- **font weight**: interpolated via discrete steps (multiples of 100). The interpolation happens in real number space and is converted to an integer by rounding to the nearest multiple of 100, with values halfway between multiples of 100 rounded towards positive infinity.
- **number**: interpolated as real (floating point) numbers.
- **rectangle**: interpolated via the x, y, width and height components (treating each as a number).

- **visibility**: if one of the values is 'visible', interpolated as a discrete step where values of the timing function between 0 and 1 map to 'visible' and other values of the timing function (which occur only at the start/end of the transition or as a result of 'cubic-bezier()' functions with Y values outside of [0, 1]) map to the closer endpoint; if neither value is 'visible' then not interpolable.
- **shadow list**: Each shadow in the list is interpolated via the color (as color) component, and x, y, blur, and (when appropriate) spread (as length) components. For each shadow, if one input shadow is 'inset' and the other is not, then the result for that shadow matches the inputs; otherwise the entire list is not interpolable. If the lists of shadows have different lengths, then the shorter list is padded at the end with shadows whose color is 'transparent', all lengths are '0', and whose 'inset' (or not) matches the longer list.
- **gradient**: interpolated via the positions and colors of each stop. They must have the same type (radial or linear) and same number of stops in order to be animated. Note: [CSS3-IMAGES] may extend this definition.
- **paint server** (SVG): interpolation is only supported between: gradient to gradient and color to color. They then work as above.
- **simple list** of other types: If the lists have the same number of items, and each pair of values can be interpolated, each item in the list is interpolated using the rules given for those types. Otherwise the values are not interpolable.
- **repeatable list** of other types: The result list has a length that is the least common multiple of the lengths of the input lists. Each item in the result is the interpolation of the value from each input list repeated to the length of the result list. If a pair of values cannot be interpolated, then the lists are not interpolable. The repeatable list concept ensures that a list that is conceptually repeated to a certain length (as 'background-origin' is repeated to the length of the 'background-image' list) or repeated infinitely will smoothly transition between any values, and so that the computed value will properly represent the result (and potentially be inherited correctly).

Future specifications may define additional types that can be animated.

See the definition of 'transition-property' for how animation of shorthand properties and the 'all' value is applied to any properties (in the shorthand) that can be animated.

7. Animatable properties

The definition of each CSS property defines when the values of that property can be interpolated by referring to the definitions of property types in the previous section. Values are animatable when both the from and the to values of the property have the type described. (When a composite type such as "length, percentage, or calc" is listed, this means that both values must fit into that composite type.) When multiple types are listed in the form "either A or B", both values must be of the same type to be interpolable.

For properties that exist at the time this specification was developed, this specification defines whether and how they are animated. However, future CSS specifications may define additional properties, additional values for existing properties, or additional animation behavior of existing values. In order to describe new animation behaviors and to have the definition of animation behavior in a more appropriate location, future CSS specifications should include an "Animatable:" line in the summary of the property's definition (in addition to the other lines described in [CSS21], section 1.4.2). This line should say "no" to indicate that a property cannot be animated or should reference an animation behavior (which may be one of the behaviors in the Animation of property types section above, or may be a new behavior) to define how the property animates. Such definitions override those given in this specification.

7.1. Properties from CSS

The following definitions define the animation behavior for properties in CSS Level 2 Revision 1 ([CSS21]) and in Level 3 of the CSS Color Module ([CSS3COLOR]).

Property Name	Type
background-color	as color
background-position	as repeatable list of simple list of length, percentage, or calc
border-bottom-color	as color
border-bottom-width	as length
border-left-color	as color
border-left-width	as length
border-right-color	as color
border-right-width	as length
border-spacing	as simple list of length
border-top-color	as color
border-top-width	as length
bottom	as length, percentage, or calc
clip	as rectangle
color	as color
font-size	as length
font-weight	as font weight
height	as length, percentage, or calc
left	as length, percentage, or calc
letter-spacing	as length
line-height	as either number or length
margin-bottom	as length
margin-left	as length
margin-right	as length
margin-top	as length
max-height	as length, percentage, or calc
max-width	as length, percentage, or calc
min-height	as length, percentage, or calc
min-width	as length, percentage, or calc
opacity	as number
outline-color	as color
outline-width	as length
padding-bottom	as length
padding-left	as length
padding-right	as length
padding-top	as length
right	as length, percentage, or calc
text-indent	as length, percentage, or calc

text-shadow	as shadow list
top	as length, percentage, or calc
vertical-align	as length
visibility	as visibility
width	as length, percentage, or calc
word-spacing	as length
z-index	as integer

7.2. Properties from SVG

All properties defined as animatable in the SVG specification, provided they are one of the property types listed above.

8. Changes since Working Draft of 12 February 2013

The following are the substantive changes made since the Working Draft dated 12 February 2013:

- Fixed missed substitution (TransitionEventInit rather than AnimationEventInit) when copying event IDL from css3-animations.
- Make naming of event constructor dictionary parameters more consistent with DOM-Level-3-Events.
- Make the behavior of simultaneous changes of 'transition-*' properties and transitionable properties even clearer.
- Computed Value line for shorthands should say "see individual properties".
- Define initial values of event properties, using initializers in TransitionEventInit.
- **Define the model for starting of transitions and their interaction with other animations more precisely**:
  - Define the before-change style and after-change style used for the style comparison, using the new concept of a style change event.
  - Define that a CSS transition for a property does not affect computed style when a CSS Animation for the same property is running, but that the transition is still running in terms of APIs.
  - Add a note pointing out that the above definitions imply that transitions can start simultaneously, from the same change, on ancestors and descendants.
  - Define that CSS transitions participate in CSS's cascading and inheritance model
- **Change the rules for automatic reversing of transitions to shorten the duration (and negative delay) based on the portion of the value space traversed instead of reversing and jumping into the middle of the timing function.**
- Move the section on reversing of transitions to be a subsection of the section on starting of transitions, since it belongs there.

For more details on these changes, see the version control change logs, which are split in two parts because of a file renaming: change log since 2013 March 28, change log before 2013 March 28.

For changes in previous working drafts, see the ChangeLog, and the above version control logs.

9. Acknowledgments

Thanks especially to the feedback from Tab Atkins, Carine Bournez, Aryeh Gregor, Vincent Hardy, Anne van Kesteren, Cameron McCormack, Alex Mogilevsky, and all the rest of the www-style community.

10. References

Normative references

Other references

[CSS21]

Bert Bos; et al. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification.* 7 June 2011. W3C Recommendation. URL: http://www.w3.org/TR/2011/REC-CSS2-20110607

[CSS3-ANIMATIONS]

Dean Jackson; et al. *CSS Animations.* 19 February 2013. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2013/WD-css3-animations-20130219/

[CSS3-IMAGES]

Elika J. Etemad; Tab Atkins Jr. *CSS Image Values and Replaced Content Module Level 3.* 17 April 2012. W3C Candidate Recommendation. (Work in progress.) URL: http://www.w3.org/TR/2012/CR-css3-images-20120417/

[CSS3CASCADE]

Håkon Wium Lie; fantasai; Tab Atkins Jr. *CSS Cascading and Inheritance Level 3.* 3 October 2013. W3C Candidate Recommendation. (Work in progress.) URL: http://www.w3.org/TR/2013/CR-css-cascade-3-20131003/

[CSS3COLOR]

Tantek Çelik; Chris Lilley; L. David Baron. *CSS Color Module Level 3.* 7 June 2011. W3C Recommendation. URL: http://www.w3.org/TR/2011/REC-css3-color-20110607

[SMIL-ANIMATION]

Patrick Schmitz; Aaron Cohen. *SMIL Animation.* 4 September 2001. W3C Recommendation. URL: http://www.w3.org/TR/2001/REC-smil-animation-20010904/

[SVG11]

Erik Dahlström; et al. *Scalable Vector Graphics (SVG) 1.1 (Second Edition).* 16 August 2011. W3C Recommendation. URL: http://www.w3.org/TR/2011/REC-SVG11-20110816/

[WCAG20]

Ben Caldwell; et al. *Web Content Accessibility Guidelines (WCAG) 2.0.* 11 December 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-WCAG20-20081211/

Property index

Property	Values	Initial	Applies to	Inh.	Percentages	Media
transition	<single-transition> [',' <single-transition>]*	see individual properties	all elements, :before and :after pseudo elements	no	N/A	interactive

transition-delay	<time> [, <time>]*	0s	all elements, :before and :after pseudo elements	no	N/A	interactive
transition-duration	<time> [, <time>]*	0s	all elements, :before and :after pseudo elements	no	N/A	interactive
transition-property	none | <single-transition-property> [',' <single-transition-property>]*	all	all elements, :before and :after pseudo elements	no	N/A	visual
transition-timing-function	<single-transition-timing-function> [',' <single-transition-timing-function>]*	ease	all elements, :before and :after pseudo elements	no	N/A	interactive

Index