CSS Transitions Level 2

Editor's Draft, 29 January 2025

▼ More details about this document

This version:
https://drafts.csswg.org/css-transitions-2/

Latest published version:
https://www.w3.org/TR/css-transitions-2/

Previous Versions:
https://www.w3.org/TR/2023/WD-css-transitions-2-20230905/

Feedback:
CSSWG Issues Repository
Bugzilla bugs for all levels
Inline In Spec

Editors:
L. David Baron (Google)
Brian Birtles (Invited Expert)

Suggest an Edit for this Spec:
GitHub Editor

Abstract

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

CSS is a language for describing the rendering of structured documents (such as HTML and XML) on screen, on paper, etc.

Status of this document

This is a public copy of the editors' draft. It is provided for discussion only and may change at any moment. Its publication here does not imply endorsement of its contents by W3C. Don't cite this document other than as work in progress.

Please send feedback by filing issues in GitHub (preferred), including the spec code "css-transitions" in the title, like this: "[css-transitions] *…summary of comment…*". All issues and comments are archived. Alternately, feedback can be sent to the (archived) public mailing list www-style@w3.org.

This document is governed by the 03 November 2023 W3C Process Document.

Table of Contents

§ 1. Delta specification

This is a delta specification, meaning that it currently contains only the differences from CSS Transitions Level 1 [CSS-TRANSITIONS-1]. Once the Level 1 specification is closer to complete, it will be merged with the additions here into a complete level 2 specification.

§ 2. Transitions

§ 2.1. The ‘transition-property’ Property

ISSUE 1 Insert text from *CSS Transitions* § 2.1 The transition-property Property and backport the following paragraph. ¶

Although the ‘transition-property’ may specify shorthand properties and the ‘all’ keyword, individual transitions are generated for each longhand sub-property that is transitionable. The ***expanded transition property name*** of a transition is the name of the longhand sub-property for which the transition was generated (e.g. ‘border-left-width’).

ISSUE 2 Update the defining instance of transitionable once it is ported from Level 1. ¶

In CSS Transitions Level 2, when comparing the before-change style and after-change style for a given property, the property values are ***transitionable*** if:

1. They have an animation type that is neither not animatable nor discrete, or
2. The ‘transition-behavior’ is ‘allow-discrete’ and they have an animation type that is discrete.

NOTE: When values with a discrete animation type are transitioned, they flip at 50% progress.

§ 2.2. The ‘transition-duration’ Property

The ‘transition-duration’ property specifies the iteration duration of the transition’s associated animation effect.

§ 2.3. The ‘transition-timing-function’ Property

The ‘transition-timing-function’ property specifies the timing function of the transition’s associated animation effect (see *Web Animations* § 4.6.11 Easing (effect timing transformations)).

NOTE: Unlike CSS animations, the timing function for CSS transitions applies to the animation effect as opposed to the individual keyframes since this allows it to be reflected in the transformed progress as used when calculating the reversing shortening factor.

§ 2.4. The ‘transition-delay’ Property

The ‘transition-delay’ property specifies the start delay of the transition’s associated animation effect.

§ 2.5. The ‘transition-behavior’ Property

The ‘transition-behavior’ property specifies whether transitions will be started or not for discrete properties.

Name:	***‘transition-behavior’***
Value:	<transition-behavior-value>#
Initial:	normal
Applies to:	all elements
Inherited:	no

Percentages:	N/A
Computed value:	as specified
Canonical order:	per grammar
Animation type:	not animatable

The syntax for specifying ‘transition-behavior’ is as follows:

‘<transition-behavior-value>’ = normal | allow-discrete

When ‘normal’ is specified, transitions will not be started for discrete properties, only for interpolable properties. When ‘allow-discrete’ is specified, transitions will be started for discrete properties as well as interpolable properties.

§ 2.6. The ‘transition’ Shorthand Property

The syntax for specifying an item in the ‘transition’ shorthand is as follows:

‘<single-transition>’ = [none | <single-transition-property>] || <time> || <easing-function> || <time> || <transition-behavior-value>

§ 3. Starting of transitions

The ***owning element*** of a transition refers to the element or pseudo-element to which the ‘transition-property’ property was applied that generated the animation. A transition may be disassociated from its owning element such that it has no owning element.

The set of running transitions includes only those transitions that have an owning element.

Whenever an implementation is required to cancel a transition, it must disassociate the transition from its owning element and run the procedure to cancel an animation on the transition.

Although not explicitly defined by level 1 of this specification, in addition to canceling transitions on elements that are no longer connected, implementations must also cancel any running transitions on elements that are no longer being rendered and remove transitions on them from the set of completed transitions.

Note that calling the `cancel()` method on the `CSSTransition` object representing a running transition does *not* cause the transition to be disassociated from its owning element.

§ 3.1. Faster reversing of interrupted transitions

The reversing shortening factor and reversing-adjusted start value associated with a transition in order to produce correct reverse behavior, are associated with the transition itself and not, for example, its animation effect. As a result, transitions will use these same values when producing a reversed transition, even if the transition’s associated animation effect has been updated or replaced using the Web Animations API [WEB-ANIMATIONS].

§ 3.2. The current transition generation

Associated with each top-level browsing context is a ***current transition generation*** that is incremented on each style change event.

Each time a new transition is generated, the current value of the (already incremented) current transition generation is stored as the transition’s ***transition generation***.

§ 3.3. Defining before-change style: the ‘@starting-style’ rule

In Level 1 of this specification, transitions can only start during a style change event for elements which have a defined before-change style established by the previous style change event. That means a transition could not be started on an element that was not being rendered for the previous style change event (see: *CSS Transitions* § 3 Starting of transitions).

In some cases it makes sense to start transitions on newly inserted elements or elements that change from not being rendered to being rendered. To allow for that, this specification introduces ‘@starting-style’.

The ***‘@starting-style’*** rule is a grouping rule. The style rules inside it are used to establish styles to transition from, if the previous style change event did not establish a before-change style for the element whose styles are being computed.

NOTE: This means that ‘@starting-style’ rules only apply to some elements during a computed style update, namely elements that were not rendered or part of the DOM during the previous style change event.

Define ***starting style*** for an element as the after-change style with ‘@starting-style’ rules applied in addition. If an element does not have a before-change style for a given style change event, the starting style is used instead of the before-change style to compare with the after-change style to start transitions (*CSS Transitions* § 3 Starting of transitions).

The rules inside ‘@starting-style’ cascade as any other grouped style rules without introducing any new ordering to the cascade, which means rules inside ‘@starting-style’ do not necessarily win over those outside.

Style rules in ‘@starting-style’ do not apply to after-change style. Thus, the presence of matching rules in ‘@starting-style’ can cause transitions to occur on elements that otherwise could not have transitions because they lack a before-change style.

Starting style inherits from the parent’s after-change style just like after-change style does.

EXAMPLE 1 ¶

The ‘background-color’ of an `h1` element can be transitioned from transparent to green when it is initially rendered:

```
h1 {
  transition: background-color 1.5s;
  background-color: green;
}
@starting-style {
  h1 {
    background-color: transparent;
  }
}
```

Conditional rules can be used with CSS Nesting:

```
h1 {
  transition: background-color 1.5s;
  background-color: green;
  @starting-style {
    background-color: transparent;
  }
}
```

EXAMPLE 2 ¶

The ‘opacity’ of an element can be transitioned when the element changes to or from ‘display: none’:

```
#target {
  transition-property: opacity, display;
  transition-duration: 0.5s;
  display: block;
  opacity: 1;
  @starting-style {
    opacity: 0;
  }
}
#target.hidden {
  display: none;
  opacity: 0;
}
```

The display is transitioning to allow for an opacity transition before flipping from ‘display:block’ to ‘display:none’.

Specifying ‘opacity: 0’ in the ‘@starting-style’ rule means the element will transition opacity from ‘0’ to ‘1’ when inserted into the tree or when the `hidden` class flips ‘display’ from ‘none’ to ‘block’ as the target element does not already have a before-change style in those cases.

Specifying ‘opacity: 0’ in the `#target.hidden` rule makes ‘opacity’ transition from ‘1’ to ‘0’ when the `hidden` class is added.

Global, name-defining at-rules such as ‘@keyframes’, ‘@font-face’, and ‘@layer’ are allowed inside ‘@starting-style’, and when present behave as if they were outside of ‘@starting-style’.

§ 3.3.1. The `CSSStartingStyleRule` interface

The `CSSStartingStyleRule` interface represents a ‘@starting-style’ rule.

```
[Exposed=Window]
interface CSSStartingStyleRule : CSSGroupingRule {
};
```

§ 4. Application of transitions

§ 4.1. Animation composite order

Animations generated from the markup defined in this specification have an animation class of ‘CSS Transition’.

CSS Transitions have an *earlier* composite order that CSS Animations and animations without a specific animation class.

Within the set of CSS Transitions, two animations A and B are sorted in composite order (first to last) as follows:

1. If neither A nor B has an owning element, sort based on their relative position in the global animation list.
2. Otherwise, if only one of A or B has an owning element, let the animation *with* an owning element sort first.
3. Otherwise, if the owning element of A and B differs, sort A and B by tree order of their corresponding owning elements. With regard to pseudo-elements, the sort order is as follows:
    - element
    - ::marker
    - ::before

- any other pseudo-elements not mentioned specifically in this list, sorted in ascending order by the Unicode codepoints that make up each selector
- ::after
- element children

4. Otherwise, if *A* and *B* have different transition generation values, sort by their corresponding transition generation in ascending order.
5. Otherwise, sort *A* and *B* in ascending order by the Unicode codepoints that make up the expanded transition property name of each transition (i.e. without attempting case conversion and such that '-moz-column-width' sorts before 'column-width').

When determining the composite order in order to sort transition events where either or both of the events is a `transitioncancel` event, use the owning element and transition generation that were set immediately prior to cancelling the transition.

Transitions generated using the markup defined in this specification are *not* added to the global animation list when they are created. Instead, these animations are appended to the global animation list at the first moment when they transition out of the idle play state after being disassociated from their owning element. Transitions that have been disassociated from their owning element but are still idle do not have a defined composite order.

> NOTE: This behavior relies on the fact that disassociating a transition from its owning element always causes it to enter (or remain) in the idle play state.

§ 4.2. Animation cascade level

Animations with an animation class of 'CSS Transition' that have an owning element are applied to the 'Transitions declaration' level of the CSS cascade. All other animations generated by the markup defined in this specification, including animations that no longer have an owning element, are applied to the 'Animation declarations' level of the cascade. (See *Web Animations* § 5.4.5 Applying the composited result.)

§ 5. Transition Events

§ 5.1. Event dispatch

> NOTE, this is a more general description of event dispatch than that of CSS Transitions Level 1 [CSS-TRANSITIONS-1] since it must account for the possibility of animations being seeked or reversed using the Web Animations API [WEB-ANIMATIONS]. Furthermore, it is possible using the Web Animations API to substitute the transition effect with an entirely different effect with properties not normally used with transitions (e.g. an effect that repeats multiple times) and hence this section provides a generic definition that accounts for the full complexity of the Web Animations model.

The target for a transition event is the transition's owning element. If there is no owning element, no transition events are dispatched (although the animation playback events defined in Web Animations are still dispatched at the corresponding `CSSTransition` object).

To avoid firing redundant events, the set of events to dispatch is based on comparing the phase of the transition in the previous animation frame to its current state.

The ***transition phase*** of a transition is initially 'idle' and is updated on each animation frame according to the first matching condition from below:

↪ **If the transition has no associated effect,**
  The transition phase is set according to the first matching condition from below:

  ↪ **If the transition has an unresolved current time,**
    The transition phase is 'idle'.

  ↪ **If the transition has a current time < 0,**
    The transition phase is 'before'.

  ↪ **Otherwise,**
    The transition phase is 'after'.

↪ **If the transition has a pending play task or a pending pause task and its phase was previously 'idle' or 'pending',**
  The transition phase is 'pending'.

↪ **Otherwise,**
  The transition phase is the phase of its associated effect.

For calculating the `elapsedTime` of each event, the following definitions are used:

- ***interval start*** = `max(min(-start delay, active duration), 0)`
- ***interval end*** = `max(min(associated effect end - start delay, active duration), 0)`

In the above formulae, references to the start delay, active duration, current iteration, iteration start, and iteration duration of a transition should be understood to refer to the corresponding properties of the transition's associated effect.

Each time a new animation frame is established, the events to dispatch are determined by comparing the transition phase in the previous and current animation frame as follows:

Change	Events dispatched	*Elapsed time* (ms)
idle → pending or before	`transitionrun`	interval start
idle → active *	`transitionrun`	interval start
	`transitionstart`	
idle → after *	`transitionrun`	interval start

Change	Events dispatched	*Elapsed time* (ms)
	transitionstart	
	transitionend	interval end
pending or before → active	transitionstart	interval start
pending or before → after *	transitionstart	interval start
	transitionend	interval end
active → after	transitionend	interval end
active → before	transitionend	interval start
after → active	transitionstart	interval end
after → before *	transitionstart	interval end
	transitionend	interval start
not idle and *not* after → idle	transitioncancel	The active time of the animation at the moment it was canceled calculated using a fill mode of both.

✔ MDN

✔ MDN

* Where multiple events are listed for a state change, all events are dispatched in the order listed and in immediate succession.

Since the elapsed time defined in the table and procedure above is expressed in milliseconds, it must be divided by 1,000 to produce a value in seconds before being assigned to the `elapsedTime` member of the `TransitionEvent`.

The above state transition chart ensures that, with the exception of transitions that are paused or have an infinite running time, the following invariants hold:

- For every `transitionrun` event there will be a exactly one `transitionend` *or* `transitioncancel` and never both.
- For every `transitionstart` event there will be a exactly one `transitionend` *or* `transitioncancel` and never both.
- Every `transitionend` event is preceded by a corresponding `transitionstart` event.

The typical sequences of events, then, are as follows:

- Regular playback: `transitionrun`, `transitionstart`, `transitionend`.
- Interrupted playback: `transitionrun`, `transitionstart`, `transitioncancel`.
- Interrupted playback during delay or pending phase: `transitionrun`, `transitioncancel`.
- Reversed playback after completion: `transitionrun`, `transitionstart`, `transitionend`, `transitionstart`, `transitionend`.

§ 6. DOM Interfaces

§ 6.1. The CSSTransition interface

```
[Exposed=Window]
interface CSSTransition : Animation {
  readonly attribute CSSOMString transitionProperty;
};
```

`transitionProperty`, of type CSSOMString, readonly
The expanded transition property name of this transition.

§ 6.2. Requirements on pending style changes

Various operations may affect the computed values of properties on elements. User agents may, as an optimization, defer recomputing these values until it becomes necessary. However, all operations included in programming interface defined in this specification, as well as those operations defined in Web Animations [WEB-ANIMATIONS] that may return objects defined by this specification, must produce a result consistent with having fully processed any such pending changes to computed values.

As an example, in the following code fragment, when the specified value of `elem`'s 'opacity' property is updated, a user agent may defer recalculating the computed value of the 'opacity' property.

The first time this occurs, calling `getComputedStyle(elt)` and subsequently accessing the `opacity` property of the result will cause the user agent to recompute the value of opacity.

After the 'opacity' property is updated a second time, the `getAnimations()` method is called on `elem`. This method is specified by Web Animations and can return `CSSTransition` objects as defined in this specification. Hence, as result of the requirements in this section, the user agent must apply any pending style changes thus generating a new `CSSTransition` for the 'opacity' property before returning its result.

¶

```
EXAMPLE 3
elem.style.transition = 'opacity 100s';
elem.style.opacity = '0';
window.getComputedStyle(elem).opacity; // 0
elem.style.opacity = '1';
elem.getAnimations()[0].transitionProperty // 'opacity'
```

§ 7. Privacy Considerations

No new privacy considerations have been reported on this specification.

§ 8. Security Considerations

No new security considerations have been reported on this specification.

§ 9. Changes

§ 9.1. Changes since First Public Working Draft (5 September 2023)

None yet

§ 9.2. Changes since Level 1, in First Public Working Draft

- Transitions can now occur on discretely-animatable properties. The newly-introduced ‘transition-behavior’ property allows opting in to this behavior.
- The ‘@starting-style’ rule (and `CSSStartingStyleRule` interface) is introduced to allow transitions on elements whose style changes from not being rendered to being rendered.
- Interactions with the Web Animations API are more clearly specified:
  - The `CSSTransition` interface is defined.
  - The composite order of the generated animations is defined.
  - The animation class of the generated animations is defined.
  - The cascade level of animation objects that were created from transitions and then reused through the Web Animations API is defined, along with the concept of owning element needed to define it.
  - The rules for event dispatch have been specified to explain what happens when the animations are seeked or reversed using the Web Animations API.
  - Interaction of the Web Animations API with the rules for faster reversing of interrupted transitions is specified.
  - § 6.2 Requirements on pending style changes are specified.
- The handling of transitions on elements that are no longer being rendered is more clearly defined (the transitions are canceled and removed from the set of completed transitions).

§ 10. Issues commonly raised as issues with previous levels

These issues were commonly reported issues with the previous level of the specification.

ISSUE 3 ¶

More powerful timing function syntax is a common request from developers. See, for example: 2013 message or 2015 thread.

ISSUE 4 ¶

Developers frequently have to trigger style flushes in order to force transitions to start. It would be good to have an API that would avoid this requirement. See, for example, 2011 proposal.

§ 11. Issues deferred from previous levels of the spec

These issues were in previous levels of the specification, but may not turn out to be important in this level either.

ISSUE 5 We may ultimately want to support a keypath syntax for the ‘transition-property’ property. A keypath syntax would enable different transitions to be specified for components of a property. For example the blur of a shadow could have a different transition than the color of a shadow. ¶

§ Conformance

§ Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

EXAMPLE 4 ¶

This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

Note, this is an informative note.

Advisements are normative sections styled to evoke special attention and are set apart from other normative text with `<strong class="advisement">`, like this:

UAs MUST provide an accessible alternative.

▼ TESTS

Tests relating to the content of this specification may be documented in "Tests" blocks like this one. Any such block is non-normative.

§ Conformance classes

Conformance to this specification is defined for three conformance classes:

style sheet
: A CSS style sheet.

renderer
: A UA that interprets the semantics of a style sheet and renders documents that use them.

authoring tool
: A UA that writes a style sheet.

A style sheet is conformant to this specification if all of its statements that use syntax defined in this module are valid according to the generic CSS grammar and the individual grammars of each feature defined in this module.

A renderer is conformant to this specification if, in addition to interpreting the style sheet as defined by the appropriate specifications, it supports all the features defined by this specification by parsing them correctly and rendering the document accordingly. However, the inability of a UA to correctly render a document due to limitations of the device does not make the UA non-conformant. (For example, a UA is not required to render color on a monochrome monitor.)

An authoring tool is conformant to this specification if it writes style sheets that are syntactically correct according to the generic CSS grammar and the individual grammars of each feature in this module, and meet all other conformance requirements of style sheets as described in this module.

§ Partial implementations

So that authors can exploit the forward-compatible parsing rules to assign fallback values, CSS renderers **must** treat as invalid (and ignore as appropriate) any at-rules, properties, property values, keywords, and other syntactic constructs for which they have no usable level of support. In particular, user agents **must not** selectively ignore unsupported component values and honor supported values in a single multi-value property declaration: if any value is considered invalid (as unsupported values must be), CSS requires that the entire declaration be ignored.

§ Implementations of Unstable and Proprietary Features

To avoid clashes with future stable CSS features, the CSSWG recommends following best practices for the implementation of unstable features and proprietary extensions to CSS.

§ Non-experimental implementations

Once a specification reaches the Candidate Recommendation stage, non-experimental implementations are possible, and implementors should release an unprefixed implementation of any CR-level feature they can demonstrate to be correctly implemented according to spec.

To establish and maintain the interoperability of CSS across implementations, the CSS Working Group requests that non-experimental CSS renderers submit an implementation report (and, if necessary, the testcases used for that implementation report) to the W3C before releasing an unprefixed implementation of any CSS features. Testcases submitted to W3C are subject to review and correction by the CSS Working Group.

Further information on submitting testcases and implementation reports can be found from on the CSS Working Group's website at http://www.w3.org/Style/CSS/Test/. Questions should be directed to the public-css-testsuite@w3.org mailing list.

§ Index

§ Terms defined by this specification

§ Terms defined by reference

[CSS-ANIMATIONS-1] defines the following terms:
- @keyframes

[CSS-BACKGROUNDS-3] defines the following terms:
- background-color
- border-left-width

[CSS-CASCADE-5] defines the following terms:
- @layer
- computed value

[CSS-COLOR-4] defines the following terms:
- opacity

[CSS-DISPLAY-4] defines the following terms:
- block
- display
- none

[CSS-EASING-2] defines the following terms:
- <easing-function>
- timing function

[CSS-FONTS-5] defines the following terms:
- @font-face

[CSS-TRANSITIONS-1] defines the following terms:
- <single-transition-property>
- TransitionEvent
- after-change style
- all
- before-change style
- cancel
- completed transition
- elapsedTime
- none
- reversing shortening factor
- reversing-adjusted start value
- running transition
- style change event
- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function
- transitioncancel
- transitionend
- transitionrun
- transitionstart

[CSS-VALUES-4] defines the following terms:
- #
- <time>
- |
- ||

[CSSOM-1] defines the following terms:
- CSSGroupingRule
- CSSOMString
- getComputedStyle(elt)

[DOM] defines the following terms:
- connected
- event target
- tree order

[HTML] defines the following terms:
- being rendered

[WEB-ANIMATIONS] defines the following terms:
- Animation
- active duration
- active time
- animation
- animation class
- animation effect
- animation frame
- animation playback events
- animation type
- associated effect
- associated effect end
- cancel an animation
- cancel()
- current iteration
- current time
- discrete
- fill mode
- getAnimations()
- global animation list
- idle
- iteration duration
- iteration start
- not animatable
- pending pause task
- pending play task
- start delay
- transformed progress
- unresolved

[WEBIDL] defines the following terms:
- Exposed

§ References

§ Normative References

[CSS-ANIMATIONS-1]
David Baron; et al. *CSS Animations Level 1*. URL: https://drafts.csswg.org/css-animations/

[CSS-BACKGROUNDS-3]
Elika Etemad; Brad Kemper. *CSS Backgrounds and Borders Module Level 3*. URL: https://drafts.csswg.org/css-backgrounds/

[CSS-CASCADE-5]
Elika Etemad; Miriam Suzanne; Tab Atkins Jr.. *CSS Cascading and Inheritance Level 5*. URL: https://drafts.csswg.org/css-cascade-5/

[CSS-EASING-2]
CSS Easing Functions Level 2. URL: https://drafts.csswg.org/css-easing/

[CSS-FONTS-5]
Chris Lilley. *CSS Fonts Module Level 5*. URL: https://drafts.csswg.org/css-fonts-5/

[CSS-TRANSITIONS-1]
David Baron; et al. *CSS Transitions*. URL: https://drafts.csswg.org/css-transitions/

[CSS-VALUES-4]
Tab Atkins Jr.; Elika Etemad. *CSS Values and Units Module Level 4*. URL: https://drafts.csswg.org/css-values-4/

[CSSOM-1]
Daniel Glazman; Emilio Cobos Álvarez. *CSS Object Model (CSSOM)*. URL: https://drafts.csswg.org/cssom/

[DOM]
Anne van Kesteren. *DOM Standard*. Living Standard. URL: https://dom.spec.whatwg.org/

[HTML]
Anne van Kesteren; et al. *HTML Standard*. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://datatracker.ietf.org/doc/html/rfc2119

[WEB-ANIMATIONS]
Brian Birtles; et al. *Web Animations*. URL: https://drafts.csswg.org/web-animations-1/

[WEBIDL]
Edgar Chen; Timothy Gu. *Web IDL Standard*. Living Standard. URL: https://webidl.spec.whatwg.org/

§ Informative References

[CSS-COLOR-4]
Chris Lilley; Tab Atkins Jr.; Lea Verou. *CSS Color Module Level 4*. URL: https://drafts.csswg.org/css-color-4/

[CSS-DISPLAY-4]
Elika Etemad; Tab Atkins Jr.. *CSS Display Module Level 4*. URL: https://drafts.csswg.org/css-display/

§ Property Index

Name	Value	Initial	Applies to	Inh.	%ages	Animation type	Canonical order	Computed value
'transition-behavior'	<transition-behavior-value>#	normal	all elements	no	N/A	not animatable	per grammar	as specified

§ IDL Index

```
[Exposed=Window]
interface CSSStartingStyleRule : CSSGroupingRule {
};

[Exposed=Window]
interface CSSTransition : Animation {
  readonly attribute CSSOMString transitionProperty;
};
```

§ Issues Index

ISSUE 1 Insert text from *CSS Transitions* § 2.1 The transition-property Property and backport the following paragraph. ↵

ISSUE 2 Update the defining instance of transitionable once it is ported from Level 1. ↵

ISSUE 3
More powerful timing function syntax is a common request from developers. See, for example: 2013 message or 2015 thread.
↵

ISSUE 4
Developers frequently have to trigger style flushes in order to force transitions to start. It would be good to have an API that would avoid this requirement. See, for example, 2011 proposal.
↵

ISSUE 5 We may ultimately want to support a keypath syntax for the 'transition-property' property. A keypath syntax would enable different transitions to be specified for components of a property. For example the blur of a shadow could have a different transition than the color of a shadow. ↵