CSS Basic User Interface Module Level 4

W3C Working Draft, 16 March 2021

This version:
https://www.w3.org/TR/2021/WD-css-ui-4-20210316/

Latest published version:
https://www.w3.org/TR/css-ui-4/

Editor's Draft:
https://drafts.csswg.org/css-ui-4/

Previous Versions:
https://www.w3.org/TR/2020/WD-css-ui-4-20200124/
https://www.w3.org/TR/2020/WD-css-ui-4-20200102/
https://www.w3.org/TR/2017/WD-css-ui-4-20171222/
https://www.w3.org/TR/2015/WD-css-ui-4-20150922/

Test Suite:
http://test.csswg.org/suites/css-ui-4_dev/nightly-unstable/

Issue Tracking:
CSSWG Issues Repository
Inline In Spec

Editor:
Florian Rivoal (On behalf of Bloomberg)

Suggest an Edit for this Spec:
GitHub Editor

Abstract

This specification describes user interface related properties and values to style HTML and XML (including XHTML). It includes and extends user interface related features from the properties and values of previous CSS levels. It uses various properties and values to style basic user interface elements in a document.

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 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 https://www.w3.org/TR/.

This document was published by the CSS Working Group as a **Working Draft**. 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.

Please send feedback by filing issues in GitHub (preferred), including the spec code "css-ui" in the title, like this: "[css-ui] ...*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 15 September 2020 W3C Process Document.

This document was produced by a group operating under the 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 specification will include and extend *CSS Basic User Interface Module Level 3*. [CSS-UI-3]

The following features are at-risk, and may be dropped during the CR period:

- Applicability of 'user-select' to '::before' and '::after'

"At-risk" is a W3C Process term-of-art, and does not necessarily imply that the feature is in danger of being dropped or delayed. It means that the WG believes the feature may have difficulty being interoperably implemented in a timely manner, and marking it as such allows the WG to drop the feature if necessary when transitioning to the Proposed Rec stage, without having to publish a new Candidate Rec without the feature first.

Table of Contents

§ 1. Introduction

This module describes CSS properties which enable authors to style user interface related properties and values.

Section 2.1 of CSS1 [CSS1] and Chapter 18 of CSS2 [CSS21] introduced several user interface related properties and values. User Interface for CSS3 (16 February 2000) introduced several new user interface related features.

[CSS-UI-3] was later introduced to incorporate, extend, and supersede these. This specification continues this work, and in turn replaces [CSS-UI-3].

§ 1.1. Purpose

The purpose of this specification is to achieve the following objectives:

- Extend the user interface features in [CSS21] and [CSS-UI-3]
- Provide additional CSS mechanisms to augment or replace other dynamic presentation related features in HTML.
- Introduce directional navigation properties to assist in the construction of user interfaces which make use of a directional navigation model.

§ 2. Module Interactions

This document defines new features not present in earlier specifications. In addition, it replaces and supersedes [CSS-UI-3], which itself replaced and superseded the following:

- Section 18.1, section 18.4, and Information on the stacking of outlines defined in Appendix E of Cascading Style Sheets, level 2, revision 1 [CSS21]
- User Interface for CSS3 (16 February 2000) [CSS-UI-3]

> Note: The 'box-sizing' property was previously defined in this section of the specification, but has been moved to CSS Sizing 3 §3.3 Box Edges for Sizing: the box-sizing property.

> Note: The 'text-overflow' property was previously defined in this section of the specification, but has been moved to CSS Overflow 3 §4.1 Overflow Ellipsis: the text-overflow property.

§ 2.1. Value Definitions

This specification follows the CSS property definition conventions from [CSS21] using the value definition syntax from [CSS-VALUES-3]. Value types not defined in this specification are defined in

CSS Values & Units [CSS-VALUES-3]. Combination with other CSS modules may expand the definitions of these value types.

In addition to the property-specific values listed in their definitions, all properties defined in this specification also accept the CSS-wide keywords as their property value. For readability they have not been repeated explicitly.

§ 3. Outline properties

At times, style sheet authors may want to create outlines around visual objects such as buttons, active form fields, image maps, etc., to make them stand out. Outlines differ from borders in the following ways:

1. Outlines do not take up space.
2. Outlines may be non-rectangular.
3. UAs often render outlines on elements in the ‘:focus’ state.

The outline properties control the style of these dynamic outlines.

The stacking of the rendering of these outlines is explicitly left up to implementations to provide a better user experience per platform. This supersedes the stacking of outlines as defined in Appendix E of CSS 2.1 [CSS21].

> **Keyboard users, in particular people with disabilities who may not be able to interact with the page in any other fashion, depend on the outline being visible on elements in the ‘:focus’ state, thus authors must not make the outline invisible on such elements without making sure an alternative highlighting mechanism is provided.**

The rendering of applying transforms to outlines is left explicitly undefined in CSS3-UI.

§ 3.1. Outlines Shorthand: the ‘outline’ property

Name:	***‘outline’***
Value:	[<'outline-color'> || <'outline-style'> || <'outline-width'>]
Initial:	see individual properties

Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	see individual properties
Animation type:	see individual properties
Canonical order:	per grammar

§ 3.2. Outline Thickness: the 'outline-width' property

Name:	***'outline-width'***
Value:	<line-width>
Initial:	medium
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	absolute length; '0' if the outline style is 'none'.
Canonical order:	per grammar
Animation type:	by computed value

§ 3.3. Outline Patterns: the 'outline-style' property

Name:	***'outline-style'***
Value:	auto | <outline-line-style>
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	specified keyword
Canonical order:	per grammar
Animation type:	by computed value

§ 3.4. Outline Colors: the 'outline-color' property

Name:	***'outline-color'***
Value:	<color> | invert
Initial:	invert
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	The computed value for 'invert' is 'invert'. For <color> values, see [[!CSS-COLOR-4#resolving-color-values]] in [CSS-COLOR-4].
Canonical order:	per grammar

Animation type:	by computed value

The outline created with the outline properties is drawn "over" a box, i.e., the outline is always on top and doesn't influence the position or size of the box, or of any other boxes. Therefore, displaying or suppressing outlines does not cause reflow.

Outlines may be non-rectangular. For example, if the element is broken across several lines, the outline should be an outline or minimum set of outlines that encloses all the element's boxes.

Each part of the outline should be fully connected rather than open on some sides (as borders on inline elements are when lines are broken).

The parts of the outline are not required to be rectangular. To the extent that the outline follows the border edge, it should follow the 'border-radius' curve.

The position of the outline may be affected by descendant boxes.

User agents should use an algorithm for determining the outline that encloses a region appropriate for conveying the concept of focus to the user.

> Note: This specification does not define the exact position or shape of the outline, but it is typically drawn immediately outside the border box.

The 'outline-width' property accepts the same values as 'border-width' (CSS Backgrounds 3 §3.3 Line Thickness: the border-width properties).

'<outline-line-style>' accepts the same values as <line-style> (CSS Backgrounds 3 §3.2 Line Patterns: the border-style properties) with the same meaning, except that 'hidden' is not a legal outline style. In addition, the 'outline-style' property accepts the value 'auto'. The 'auto' value permits the user agent to render a custom outline style, typically a style which is either a user interface default for the platform, or perhaps a style that is richer than can be described in detail in CSS, e.g. a rounded edge outline with semi-translucent outer pixels that appears to glow. As such, this specification does not define how the 'outline-color' and 'outline-width' are incorporated or used (if at all) when rendering 'auto' style outlines. User agents may treat 'auto' as 'solid'.

The 'outline-color' property accepts all colors, as well as the keyword ***'invert'***. 'invert' is expected to perform a color inversion on the pixels on the screen. This is a common trick to ensure the focus border is visible, regardless of color background.

Conformant UAs may ignore the 'invert' value on platforms that do not support color inversion of the pixels on the screen.

If the UA does not support the 'invert' value, then it must reject that value at parse-time, and the initial value of the 'outline-color' property is the 'currentColor' keyword.

The 'outline' property is a shorthand property and sets all three of 'outline-style', 'outline-width', and 'outline-color'.

> Note: The outline is the same on all sides. In contrast to borders, there are no 'outline-top', 'outline-left', etc. properties.

This specification does not define how multiple overlapping outlines are drawn or how outlines are drawn for boxes that are partially obscured behind other elements.

> EXAMPLE 1
> Here's an example of drawing a thick outline around a BUTTON element:
>
> ```
> button { outline: thick solid }
> ```

Graphical user interfaces may use outlines around elements to tell the user which element on the page has the focus. These outlines are shown in addition to any borders, and switching outlines on and off should not cause the document to reflow. The focus is the subject of user interaction in a document (e.g. for entering text or selecting a button).

> EXAMPLE 2
> For example, to draw a thick black line around an element when it has the focus, and a thick red line when it is active, the following rules can be used:
>
> ```
> :focus  { outline: thick solid black }
> :active { outline: thick solid red }
> ```

> Note: Since the outline does not affect formatting (i.e., no space is left for it in the box model), it may well overlap other elements on the page.

§ 3.5. Offsetting the Outline: the 'outline-offset' property

By default, the outline is drawn starting just outside the border edge. However, it is possible to offset the outline and draw it beyond the border edge.

Name:	***'outline-offset'***
Value:	<length>
Initial:	0
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	absolute length
Canonical order:	per grammar
Animation type:	by computed value

If the computed value of 'outline-offset' is anything other than 0, then the outline is outset from the border edge by that amount.

EXAMPLE 3

For example, to leave 2 pixels of space between a focus outline and the element that has the focus or is active, the following rule can be used:

```
:focus,:active  { outline-offset: 2px }
```

Negative values must cause the outline to shrink into the border box. Both the height and the width of the outside of the shape drawn by the outline should not become smaller than twice the computed value of the 'outline-width' property to make sure that an outline can be rendered even with large

negative values. User agents should apply this constraint independently in each dimension. If the outline is drawn as multiple disconnected shapes, this constraint applies to each shape separately.

§ 4. Resizing

CSS2.1 provides a mechanism for controlling the appearance of a scrolling mechanism (e.g. scrollbars) on block container elements. This specification adds a mechanism to that for controlling user resizability of elements as well as the ability to specify text overflow behavior.

§ 4.1. Resizing Boxes: the 'resize' property

The 'resize' property allows the author to specify whether or not an element is resizable by the user, and if so, along which axis/axes.

Name:	***'resize'***
Value:	none | both | horizontal | vertical | block | inline
Initial:	none
Applies to:	elements with 'overflow' other than visible, and optionally replaced elements such as images, videos, and iframes
Inherited:	no
Percentages:	N/A
Computed value:	specified keyword
Canonical order:	per grammar
Animation type:	discrete

none
The UA does not present a resizing mechanism on the element, and the user is given no direct manipulation mechanism to resize the element.

both
: The UA presents a bidirectional resizing mechanism to allow the user to adjust both the height and the width of the element.

horizontal
: The UA presents a unidirectional horizontal resizing mechanism to allow the user to adjust only the width of the element.

vertical
: The UA presents a unidirectional vertical resizing mechanism to allow the user to adjust only the height of the element.

block
: The UA presents a unidirectional block-axis resizing mechanism to allow the user to adjust only the block size of the element.

inline
: The UA presents a unidirectional inline-axis resizing mechanism to allow the user to adjust only the inline size of the element.

Currently it is possible to control the appearance of the scrolling mechanism (if any) on an element using the 'overflow' property (e.g. `overflow: scroll` vs. `overflow: hidden` etc.). The purpose of the 'resize' property is to allow control over the appearance and function of the resizing mechanism (e.g. a resize box or widget) on the element.

> Note: The resizing mechanism is NOT the same as the scrolling mechanism, nor is it related to any UA mechanism for zooming. The scrolling mechanism allows the user to determine which portion of the contents of an element is shown. The resizing mechanism allows the user to determine the size of the element.

The 'resize' property applies to elements whose computed 'overflow' value is something other than 'visible'. UAs may also apply it, regardless of the value of the 'overflow' property, to:

- Replaced elements representing images or videos, such as `<img>`, `<video>`, `<picture>`, `<svg>`, `<object>`, or `<canvas>`.
- The `<iframe>` element.

The effect of the 'resize' property on generated content is undefined. Implementations should not apply the 'resize' property to generated content.

> Note: the 'resize' property may apply to generated content in the future if there is implementation of Interface CSSPseudoElement.

When an element is resized by the user, the user agent sets the 'width' and 'height' properties to px unit length values of the size indicated by the user, in the element's style attribute DOM, replacing existing property declaration(s), if any, without '!important', if any.

If an element is resized in only one dimension, only the corresponding property is set, not both.

The precise direction of resizing (i.e. altering the top-left of the element or altering the bottom-right) may depend on a number of CSS layout factors including whether the element is absolutely positioned, whether it is positioned using the 'right' and 'bottom' properties, whether the language of the element is right-to-left etc. The UA should consider the direction of resizing (as determined by CSS layout), as well as platform conventions and constraints when deciding how to convey the resizing mechanism to the user.

The user agent must allow the user to resize the element with no other constraints than what is imposed by 'min-width', 'max-width', 'min-height', and 'max-height'.

> Note: There may be situations in which user attempts to resize an element appear to be overridden or ignored, e.g. because of '!important' cascading declarations that supersede that element's style attribute 'width' and 'height' properties in the DOM.

Changes to the computed value of an element's 'resize' property do not reset changes to the style attribute made due to user resizing of that element.

> EXAMPLE 4
> For example, to make iframes scrollable *and* resizable, the following rule can be used:

```
iframe,object[type^="text/"],
object[type$="+xml"],object[type="application/xml"]
{
  overflow:auto;
  resize:both;
}
```

§ 5. Pointing Devices and Keyboards

§ 5.1. Pointer interaction

§ 5.1.1. Styling the Cursor: the 'cursor' property

Name:	***'cursor'***
Value:	[[<url> [<x> <y>]?,]* [auto | default | none | context-menu | help | pointer | progress | wait | cell | crosshair | text | vertical-text | alias | copy | move | no-drop | not-allowed | grab | grabbing | e-resize | n-resize | ne-resize | nw-resize | s-resize | se-resize | sw-resize | w-resize | ew-resize | ns-resize | nesw-resize | nwse-resize | col-resize | row-resize | all-scroll | zoom-in | zoom-out]]
Initial:	auto
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Computed value:	as specified, except with any relative URLs converted to absolute
Canonical order:	per grammar
Animation type:	discrete

This property specifies the type of cursor to be displayed for the pointing device when the cursor's hotspot is within the element's border edge.

> Note: As per CSS Backgrounds 3 §4.1 Curve Radii: the border-radius properties, the border edge is affected by 'border-radius'.

In the case of overlapping elements, which element determines the type of cursor is based on hit testing: the element determining the cursor is the one that would receive a click initiated from this

position.

> Note: The specifics of hit testing are out of scope of this specification. Hit testing will hopefully be defined in a future revision of CSS or HTML.

User agents may ignore the cursor property over native user agent controls such as scrollbars, resizers, or other native UI widgets e.g. those that may be used inside some user agent specific implementations of form elements. User agents may also ignore the ‘cursor’ property and display a cursor of their choice to indicate various states of the UA’s user interface, such as a busy cursor when the page is not responding, or a text cursor when the user is performing text selection.

> Note: [HTML] defines special handling of image maps for the ‘cursor’ property.

Values have the following meanings:

image cursors

<url>
: The user agent retrieves the cursor from the resource designated by the URI. If the user agent cannot handle the first cursor of a list of cursors, it must attempt to handle the second, etc. If the user agent cannot handle any user-defined cursor, it must use the cursor keyword at the end of the list. Conforming User Agents may, instead of <url>, support <image> which is a superset.

  The UA must support the following image file formats:

  - PNG, as defined in [PNG]
  - SVG, as defined in [SVG11], in secure static mode [SVG2], if it has a natural size.
  - any other non-animated image file format that they support for <image> in other properties, such as the ‘background-image’ property

  In addition, the UA should support the following image file formats:

  - SVG, as defined in [SVG11], in secure animated mode [SVG2], if it has a natural size.
  - any other animated image file format that they support for <image> in other properties, such as the ‘background-image’ property

  The UA may also support additional file formats, including SVG, as defined in [SVG11], in secure static mode or secure animated mode [SVG2], even if it does not have a natural size.

> Note: The CSS Working Group initially intended support for all SVG, naturally sized or not. Support for non-naturally sized SVG was downgraded from mandatory to optional due to lack of implementations.

> Note: At the time of writing this specification (spring 2015), the only file formats supported for cursors in common desktop browsers are the .ico and .cur file formats, as designed by Microsoft. For compatibility with legacy content, UAs are encouraged to support these, even though the lack of an open specification makes it impossible to have a normative requirement about these formats. Some information on these formats can be found on Wikipedia.

The default object size for cursor images is a UA-defined size that should be based on the size of a typical cursor on the UA's operating system.

The concrete object size is determined using the default sizing algorithm. If an operating system is **incapable** of rendering a cursor above a given size, cursors larger than that size must be shrunk to within the OS-supported size bounds, while maintaining the cursor image's natural aspect ratio, if any.

The optional <x> and <y> coordinates identify the exact position within the image which is the pointer position (i.e., the hotspot).

<x>
<y>

Each is a <number>. The x-coordinate and y-coordinate of the position in the cursor's coordinate system (left/top relative) which represents the precise position that is being pointed to.

> Note: This specification does not define how the coordinate systems of the various types of <image> are established, and defers these definitions to [CSS4-IMAGES].

If the values are unspecified, then the natural hotspot defined inside the image resource itself is used. If both the values are unspecific and the referenced cursor has no defined hotspot, the effect is as if a value of "0 0" was specified.

If the coordinates of the hotspot, as specified either inside the image resource or by <x> and <y> values, fall outside of the cursor image, they must be clamped (independently) to fit.

general purpose cursors

'auto'

The UA determines the cursor to display based on the current context. Specifically, ‘auto’ behaves as ‘text’ over selectable text or editable elements, and ‘default’ otherwise.

> Note: When over selectable text or editable elements, as it does with ‘text’, the UA must use a vertical or horizontal text cursor, as appropriate for the writing mode of the element, and may take transforms and other visual effects into account as well.

‘default’

The platform-dependent default cursor. Often rendered as an arrow.

‘none’

No cursor is rendered for the element.

links and status cursors

‘context-menu’

A context menu is available for the object under the cursor. Often rendered as an arrow with a small menu-like graphic next to it.

‘help’

Help is available for the object under the cursor. Often rendered as a question mark or a balloon.

‘pointer’

The cursor is a pointer that indicates a link. Often rendered as the backside of a hand with the index finger extended.

Unless otherwise specified, UAs must apply ‘cursor: pointer’ to hyperlinks for all supported document formats via the UA stylesheet, using a normal (i.e. not ‘!important’) declaration.

Authors should use pointer on links and may use on other interactive elements.

‘progress’

A progress indicator. The program is performing some processing, but is different from ‘wait’ in that the user may still interact with the program. Often rendered as a spinning beach ball, or an arrow with a watch or hourglass.

‘wait’

Indicates that the program is busy and the user should wait. Often rendered as a watch or hourglass.

selection cursors

‘cell’

Indicates that a cell or set of cells may be selected. Often rendered as a thick plus-sign with a dot in the middle.

'crosshair'

A simple crosshair (e.g., short line segments resembling a "+" sign). Often used to indicate a two dimensional bitmap selection mode.

'text'

Indicates text that may be selected. Often rendered as an I-beam. User Agents must automatically display a vertical I-beam/cursor over elements with a horizontal writing mode, and a horizontal I-beam/cursor (e.g. same as the 'vertical-text' keyword) over elements in a vertical writing mode. Additionally, User Agents may take transforms (see [CSS-TRANSFORMS-1]) or other visual effects such as text on a path (See SVG 2 §11.8 Text on a path), when choosing between the horizontal or vertical text cursor, and may display any angle of I-beam/cursor for text that is rendered at any particular angle.

'vertical-text'

Indicates vertical-text that may be selected. Often rendered as a horizontal I-beam.

drag and drop cursors

'alias'

Indicates an alias of/shortcut to something is to be created. Often rendered as an arrow with a small curved arrow next to it.

'copy'

Indicates something is to be copied. Often rendered as an arrow with a small plus sign next to it.

'move'

Indicates something is to be moved.

'no-drop'

Indicates that the dragged item cannot be dropped at the current cursor location. Often rendered as a hand or pointer with a small circle with a line through it.

'not-allowed'

Indicates that the requested action will not be carried out. Often rendered as a circle with a line through it.

¶ ***'grab'***

Indicates that something can be grabbed (dragged to be moved). Often rendered as the backside of an open hand.

'grabbing'

Indicates that something is being grabbed (dragged to be moved). Often rendered as the backside of a hand with fingers closed mostly out of view.

resizing and scrolling cursors

'e-resize', ***'n-resize'***, ***'ne-resize'***, ***'nw-resize'***, ***'s-resize'***, ***'se-resize'***, ***'sw-resize'***, ***'w-resize'***

Indicates that some edge is to be moved. For example, the ‘se-resize’ cursor is used when the movement starts from the south-east corner of the box.

‘ew-resize’, ***‘ns-resize’***, ***‘nesw-resize’***, ***‘nwse-resize’***
Indicates a bidirectional resize cursor.

‘col-resize’
Indicates that the item/column can be resized horizontally. Often rendered as arrows pointing left and right with a vertical bar separating them.

‘row-resize’
Indicates that the item/row can be resized vertically. Often rendered as arrows pointing up and down with a horizontal bar separating them.

‘all-scroll’
Indicates that the something can be scrolled in any direction. Often rendered as arrows pointing up, down, left, and right with a dot in the middle.

¶ **zooming cursors**

‘zoom-in’, ***‘zoom-out’***
Indicates that something can be zoomed (magnified) in or out, and often rendered as a magnifying glass with a "+" or "-" in the center of the glass, for ‘zoom-in’ and ‘zoom-out’ respectively.

EXAMPLE 5
Here is an example of using several cursor values.

```
:link,:visited {
    cursor: url(example.svg#linkcursor),
      url(hyper.cur),
      url(hyper.png) 2 3,
      pointer
}
```

This example sets the cursor on all hyperlinks (whether visited or not) to an external SVG `<cursor>`. User agents that don’t support SVG cursors would simply skip to the next value and attempt to use the "hyper.cur" cursor. If that cursor format was also not supported, the UA could attempt to use the "hyper.png" cursor with the explicit hotspot. Finally if the UA does not support any of those image cursor formats, the UA would skip to the last value and render the ‘pointer’ cursor.

§ *5.1.1.1. Cursor of the canvas*

The document canvas is the infinite surface over which the document is rendered [CSS21]. Since no element corresponds to the canvas, in order to allow styling of the cursor when not over any element, the canvas cursor re-uses the root element's cursor. However, if no boxes are generated for the root element (for example, if the root element has 'display: none'), then the canvas cursor is the platform-dependent default cursor.

> Note: An element might be invisible but still generate boxes. For example, if the element has 'visibility: hidden' but not 'display: none', boxes are generated for it and its cursor is used for the canvas.

§ 5.2. Insertion caret

§ 5.2.1. Coloring the Insertion Caret: the 'caret-color' property

Name:	***'caret-color'***
Value:	auto | <color>
Initial:	auto
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Computed value:	The computed value for 'auto' is 'auto'. For <color> values, see [[!CSS-COLOR-4#resolving-color-values]] in [CSS-COLOR-4].
Canonical order:	per grammar
Animation type:	by computed value

auto

User agents should use 'currentColor'. User agents may automatically adjust the color of caret to ensure good visibility and contrast with the surrounding content, possibly based on the

currentColor, background, shadows, etc.

> Note: When ‘caret-shape’ is ‘block’, ensuring good visibility and contrast is best achieved with a UA determined color other than ‘currentColor’.

<color>
: The insertion caret is colored with the specified color.

The caret is a visible indicator of the insertion point in an element where text (and potentially other content) is inserted by the user. This property controls the color of that visible indicator.

> Note: UAs might have additional things that count as “carets”. For example, some UAs can show a “navigation caret”, which acts similarly to an insertion caret but can be moved around in non-editable text and is functionally a caret. On the other hand, the cursor image shown when hovering over text when the ‘cursor’ property is ‘auto’, or when hovering over an element where the ‘cursor’ property is ‘text’ or ‘vertical-text’, though it sometimes resembles a caret, is not a caret (it’s a cursor).

> EXAMPLE 6
>
> Example: a textarea with `caret-color:#00aacc;`
>
> ```
> caret-color:#00aacc
> ```

§ 5.2.2. Shape of the insertion caret: ‘caret-shape’

Name:	***‘caret-shape’***
Value:	auto | bar | block | underscore
Initial:	auto
Applies to:	elements that accept input
Inherited:	yes
Percentages:	N/A
Computed value:	specified keyword
Canonical	per grammar

order:

Animation type:	by computed value

This property allows authors to specify the desired shape of the text insertion caret.

Within the context of this definition, ***character*** is to be understood as *extended grapheme cluster*, as defined in [UAX29], and ***visible character*** means a character with a non-zero advance measure.

'auto'
: The UA determines the shape of the caret. It should match platform conventions, and may be adjusted based on context. For example, if a UA switches between insert mode and overtype mode when the user presses the *insert* key on their keyboard, it may show a 'bar' caret in insert mode, and a 'block' caret in overtype mode.

'bar'
: The UA must render the text insertion caret as a thin bar placed at the insertion point. This means it is between, before, or after characters, not over them. It should be perpendicular to the inline progression direction, although UAs may render it slanted when inserting italic or oblique text.

'block'
: The UA must render the text insertion caret as a rectangle overlapping the next visible character following the insertion point. If there is no visible character after the insertion point, the UA must render the caret after the last visible character. UAs may render it as a slanted rectangle when inserting italic or oblique text.

'underscore'
: The UA must render the text insertion caret as a thin line under (as defined in [CSS-WRITING-MODES-3] the next visible character following the insertion point. If there is no visible character after the insertion point, the UA must render the caret after the last visible character.

The width of the 'block' and 'underscore' carets should be the advance measure of the next visible character after the insertion point, or '1ch' if there is no next visible character or if this information is impractical to determine.

When determining the orientation and appearance of the caret, UAs must take into account the writing mode [CSS-WRITING-MODES-3] and must apply transformations [CSS-TRANSFORMS-1]. If the edited text is laid out on a path, for instance by using the SVG `<textPath>` element, UAs should also account for this.

The thickness of ‘bar’ and ‘underscore’ carets—as well as that of ‘auto’ carets with a similar rendering—is determined by the user agent. When practical, the user agent should choose a thickness that ensures the caret remains visible even if it has been scaled down via means such as transformations [CSS-TRANSFORMS-1].

The stacking position of the caret is left undefined within the following constraints:

- The caret must not be obscured by the background of the element.
- UAs must render ‘block’ carets so that the character it overlaps with is not obscured by the caret.

EXAMPLE 7

This illustrates the typical appearance of the various caret shapes. In each of the sample renderings below, the insertion point is between the letters u and m.

‘caret-shape’	**Sample rendering**	**Your browser (focus each cell to see the caret)**
‘bar’	Lorem ipsum	Lorem Ipsum
‘block’	Lorem ipsum	Lorem Ipsum
‘underscore’	Lorem ispum	Lorem Ipsum

EXAMPLE 8

'underscore' or 'block' carets are commonly used in terminals and command lines, as in this example.

```
.console {
  caret-shape: underscore;
  background: black;
  color: white;
  font-family: monospace;
  padding: 1ex;
}
```

The simulated rendering below illustrates how this should look.

```
user@host:css-ui-4 $ ls -a
. .. Overview.bs Overview.html
user@host:css-ui-4 $
```

Focus the element below to see how your browser renders it.

```
user@host:css-ui-4 $ ls -a
. .. Overview.bs Overview.html
user@host:css-ui-4 $
```

§ 5.2.3. Insertion caret shorthand: 'caret'

Name:	***'caret'***
Value:	<'caret-color'> || <'caret-shape'>
Initial:	auto
Applies to:	elements that accept input
Inherited:	yes
Percentages:	N/A
Computed	see individual properties

value:	
Animation type:	see individual properties
Canonical order:	per grammar

This property is a shorthand for setting ‘caret-color’ and ‘caret-shape’ in one declaration. Omitted values are set to their initial values.

§ 5.3. Keyboard control

§ 5.3.1. Directional Focus Navigation: the ‘nav-up’, ‘nav-right’, ‘nav-down’, ‘nav-left’ properties

Name:	***‘nav-up’***, ***‘nav-right’***, ***‘nav-down’***, ***‘nav-left’***
Value:	auto | <id> [current | root | <target-name>]?
Initial:	auto
Applies to:	all enabled elements
Inherited:	no
Percentages:	N/A
Computed value:	as specified
Canonical order:	per grammar
Animation type:	discrete

auto
: The user agent automatically determines which element to navigate the focus to in response to directional navigational input.

‘<id>’

The <id> value is an ID selector [SELECT]. In response to directional navigation input corresponding to the property, the focus is navigated to the first element in tree order matching the selector.

If this refers to the currently focused element, the directional navigation input respective to the nav- property is ignored — there is no need to refocus the same element.

If no element matches the selector, the user agent automatically determines which element to navigate the focus to.

If the focus is navigated to an element that was not otherwise focusable, it becomes focusable only as the result of this directional navigation, and the ':focus' pseudo-class matches the element while it is focused as such.

> Note: there were other options under consideration for such "not otherwise focusable" elements, including focus to the next otherwise focusable element in the document tree (including descendants of such a not otherwise focusable element). Input on such other options is welcome and explicitly solicited, especially from implementation experiences and author experience using the directional navigation properties in their content.

'<target-name>'

The <target-name> parameter indicates the target frame for the focus navigation. It is a <string> and it MUST NOT start with the underscore "_" character. Error handling: if it does start with an underscore, "_parent" navigates to the parent frame, "_root" is treated as 'root', and other values navigate to a frame by that name if it exists. If the specified target frame does not exist, the parameter will be treated as the keyword 'current', which means to simply use the frame that the element is in. The keyword 'root' indicates that the user agent should target the full window.

User agents for devices with directional navigation keys respond by navigating the focus according to four respective nav-* directional navigation properties (nav-up, nav-right, nav-down, nav-left). This specification does not define which keys of a device are directional navigational keys.

> Note: Typical personal computers have keyboards with four arrow keys. One possible implementation would be to use those four arrow keys for directional navigation. For accessibility and user convenience, user agents should allow configuration of which keys on a keyboard are used for directional navigation.

EXAMPLE 9

5.3.1.1. Example: positioned buttons

Here is an example of buttons positioned in a diamond shape whose directional focus navigation is set in such a way to navigate the focus clockwise (or counter-clockwise) around the diamond shape when the user chooses to navigate directionally.

```
button { position:absolute }

button#b1 {
  top:0; left:50%;
  nav-right:#b2; nav-left:#b4;
  nav-down:#b2; nav-up:#b4;
}

button#b2 {
  top:50%; left:100%;
  nav-right:#b3; nav-left:#b1;
  nav-down:#b3; nav-up:#b1;
}

button#b3 {
  top:100%; left:50%;
  nav-right:#b4; nav-left:#b2;
  nav-down:#b4; nav-up:#b2;
}

button#b4 {
  top:50%; left:0;
  nav-right:#b1; nav-left:#b3;
  nav-down:#b1; nav-up:#b3;
}
```

Whatever markup sequence the buttons may have (which is not specified in this example) is irrelevant in this case because they are positioned, and yet, it is still important to ensure focus navigation behaviors which relate reasonably to the specified layout.

EXAMPLE 10

5.3.1.2. Example: moving focus to inside a frame

Moving the focus to an element in a specific frame requires both the element's id and the frame's name.

This example shows how to make navigating left from the button with id "foo" move the focus to the element with id "bar" within the frame named "sidebar".

```
button#foo { nav-left: #bar "sidebar"; }
```

§ 5.3.2. Obsolete: the ime-mode property

'ime-mode' is a property somewhat implemented in some browsers, that is problematic and officially obsoleted by this specification and its predecessor [CSS-UI-3].

There is documentation of non-interoperability of these implementations.

User agents should not support the 'ime-mode' property.

Authors must not use the ime-mode property.

Users may use the ime-mode property only for repair use-cases where they have to work around bad sites and legacy implementations, e.g. with a user style sheet rule like:

EXAMPLE 11
Example: user preference

```
input[type=password] {  ime-mode: auto !important;
}
```

This example CSS may be placed into a user style sheet file to force password input fields to behave in a default manner.

This specification deliberately does not attempt to document the functionality of legacy ime-mode implementations nor what they specifically support because it does not make sense to pursue or recommend any such path.

> Note: there are several [HTML5] features which authors should use to provide information to user agents that allow them to provide a better input user experience:
>
> - The global `lang` attribute
> - The `inputmode`, `pattern`, and `type` attributes of the input element

§ 6. User Interaction

§ 6.1. Controlling content selection

The 'user-select' property enables authors to specify which elements in the document can be selected by the user and how. This allows for easier interactions when not all elements are equally useful to select, avoiding accidental selections of neighboring content.

Name:	***'user-select'***
Value:	auto | text | none | contain | all
Initial:	auto
Applies to:	all elements, and optionally to the '::before' and '::after' pseudo-elements
Inherited:	no
Percentages:	n/a
Computed value:	specified keyword
Canonical order:	per grammar
Animation type:	discrete

User agents must not apply the ‘user-select’ property to the ‘::first-line’ and ‘::first-letter’ pseudo-elements.

Note: The UA may apply this property to the ‘::before’ and ‘::after’ pseudo-elements. If it does, the ‘auto’ value maps to a used value of ‘none’ on these pseudos, but other values can be specified. This preserves the legacy behavior of generated content not being selectable or copyable, which is appropriate when these pseudos are used for decorative purposes. However, this property allows them to become selectable and copyable, as the user would expect in cases where they are used to generate parts of the content, such as the issue numbers in this document.

> **To the extent possible, authors should avoid using generated content for non-decorative purposes, and should prefer including all the content in the DOM.**

This feature is at risk.

ISSUE 1 if we allow user-select to change to a value other than ‘none’, we need to figure out what this means for copyability, and DOM APIs.

When generated content in pseudo-elements becomes selectable through this mechanism, UAs should also make this content findable through their search function.

ISSUE 2 Should it also apply to ‘::marker’? To page-margin boxes? Should the used value of ‘auto’ also be ‘none’, or would ‘text’ be more appropriate?

The used value is the same as the computed value, except:

1. on editable elements where the used value is always ‘contain’ regardless of the computed value
2. when the computed value is ‘auto’, in which case the used value one of the other values as defined below

For the purpose of this specification, an ***editable element*** is either an editing host or a mutable form control with textual content, such as `<textarea>`.

ISSUE 3 Should there be constraints on what happens to the used value on elements that are editable descendants of editing hosts? The semantics are not obvious. Maybe ‘none’ should map to ‘text’, or maybe all values should map to ‘text’.

‘auto’

The used value of ‘auto’ is determined as follows:

- On the ‘::before’ and ‘::after’ pseudo-elements, the used value is ‘none’
- If the element is an editable element, the used value is ‘contain’
- Otherwise, if the used value of ‘user-select’ on the parent of this element is ‘all’, the used value is ‘all’
- Otherwise, if the used value of ‘user-select’ on the parent of this element is ‘none’, the used value is ‘none’
- Otherwise, the used value is ‘text’

> Note: This unusual combination of a non-inherited property with an initial value of ‘auto’ whose used value depends on the parent element makes it possible to create what is effectively selective inheritance. This was initially proposed by Microsoft in IE to introduce a behavior similar to inheritance except that the ‘contain’ value does not inherit.

‘text’

The element imposes no constraint on the selection.

‘none’

The UA must not allow selections to be started in this element.
A selection started outside of this element must not end in this element. If the user attempts to create such a selection, the UA must instead end the selection range at the element boundary.

> Note: As of the time of writing, experimental implementations do not all behave like this. Firefox does. Chrome and Safari almost do: for a selection started after the element and trying to go backwards into the element they behave as specified here, but for a selection started before the element and trying to go into the element they behave as if the element has ‘all’ and select it entirely. IE does not restrict selections started outside of the element from going into it at all. Another difference is that in Chrome and Safari, if the user attempts to start a selection inside a ‘user-select: none’, and to end the selection out of it, a selection will be created from the boundary of the element to the user-designated end-point. Firefox and Internet explorer behave as prescribed in this specification and do not create a selection at all.

However, if this element has descendants on which the used value of ‘user-select’ is not ‘none’, selections that start and end within these descendants are allowed.

The UA must allow selections to extend across this element, and must exclude this element from such a selection. An exception is made for UAs which do not support multiple ranges per selection, and they may include this element. If the element has descendants on which the used value of ‘user-select’ is not ‘none’, these descendants must be included in a selection extending across the element. This specification makes no normative requirement about the behavior of

the clipboard. however, UAs are encouraged to keep the visual selection consistent with what would get copied to the clipboard when copying. Copying text that does not appear to be selected, or vice versa, is highly confusing to users.

Attempting to start a selection in an element where 'user-select' is 'none', such as by clicking in it or starting a drag in it, must not cause a pre-existing selection to become unselected or to be affected in any way.

As 'user-select' is a UI convenience mechanism, not a copy protection mechanism, the UA may provide an alternative way for the user to explicitly select the text even when 'user-select' is 'none'.

> Note: 'none' is not a copy protection mechanism, and using it as such is ineffective: User Agents are allowed to provide ways to bypass it, it will have no effect on legacy User Agents that do not support it, and the user can disable it through the user style sheet or equivalent mechanisms on UAs that do anyway. Instead, 'none' is meant to make it easier for the user to select the content they want, by letting the author disable selection on UI elements that are not useful to select. Tools such as CSS validators, linters or in-browser developer tools are encouraged to use heuristics to detect and warn against incorrect or abusive usage that would hamper usability or violate common user expectations.

> Authors should be careful about not constraining the user unnecessarily. For example, setting 'user-select: none' on the root element, and relaxing that restriction on a handful of elements the author judges useful to select can be frustrating to users:
>
> - Clicking in empty areas to undo the current selection will no longer be effective
> - The author may have overlooked some areas which should be selectable
> - Areas may be added later to the page without remembering to make them selectable
> - The user may want to select pieces of text other than the main content, for instance to look them up in a dictionary or in some translation tool, or to look for warnings and error messages in a search engine…
>
> Instead, a good practice is for authors is to selectively apply 'user-select: none' to elements which seem likely to be accidentally selected when doing so would interfere with the more likely intended action. Accidentally leaving parts of the page that are unlikely to be interacted with selectable will likely cause much less frustration to users than the opposite.

'contain'

UAs must not allow a selection which is started in this element to be extended outside of this element.
A selection started outside of this element must not end in this element. If the user attempts to create such a selection, the UA must instead end the selection range at the element boundary.

The UA must allow selections to extend across this element, and such selections must include the content of the element.

> Note: At the time of writing, experimental implementations behave differently from each other about selections started outside and selections going into the element. The behavior can be observed even on browsers that do not explicitly support 'contain' by trying to select into a `<textarea>` or a contenteditable element.

> Note: This was initially introduced as an experimental feature in Internet Explorer, under the name `user-select: element`.

'all'

The content of the element must be selected atomically: If a selection would contain part of the element, then the selection must contain the entire element including all its descendants. If the element is selected and the used value of 'user-select' on its parent is 'all', then the parent must be included in the selection, recursively.

If this element has descendants on which the used value of 'user-select' is not 'all' and if a selection is entirely contained in these descendants, then the selection is not extended to include this whole element.

> Note: Selections can include more than just text and extend over images, tables, videos, etc. The behavior when copying and pasting a such selections is out of scope for this specification.

The following additions are made to the UA stylesheet for HTML:

```
button, meter, progress, select { user-select: none; }
```

> ISSUE 4 the list above is incomplete and needs to include at least the various button-like variants of `<input>`.

For compatibility with legacy content, UAs that support 'user-select' must also support ***'-webkit-user-select'*** as an alias.

Note: The details of the aliasing mechanism is intentionally left up to the UA. Making '-webkit-user-select' a shorthand property of 'user-select' is known to be an effective approach, and new implementers should consider it. However, UAs supporting '-webkit-user-select' as an alias of 'user-select' through other means exist, without adverse consequences to compatibility, so this specification allows flexibility.

§ 7. Styling Widgets

§ 7.1. Widget Accent Colors: the 'accent-color' property

Name:	***'accent-color'***
Value:	auto | <color>
Initial:	'auto'
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Computed value:	the keyword 'auto' or a computed color
Canonical order:	per grammar
Animation type:	by computed value type

User interface controls on any given platform are typically designed as a cohesive set, under a single, cohesive visual style. On many platforms (though not all), the visual style includes the use of an ***accent color***, a typically bright color that contrasts markedly with the more utilitarian background and foreground colors. Not every control uses the accent color in every state, but it is nonetheless a core part of the controls' color scheme.

The 'accent-color' CSS property allows the author to specify the accent color for user-interface controls generated by the element. Values have the following meanings:

'auto'

Represents a UA-chosen color, which should match the accent color of the platform, if any.

'<color>'

Specifies the color to be used as the accent color.

The UA should use the specified 'accent-color' to draw whichever parts of the element's control(s) would have otherwise been styled with an accent color. The UA must maintain contrast for legibility of the control, and in order to do so may adjust the luminance or brightness of the color or make color substitutions in other parts of the control (e.g. switching an overlaid glyph from using 'color' to using 'background-color'). It may also generate variations of the color for gradients etc. to match the control to platform conventions for the use of the accent color.

EXAMPLE 12

A radio button is typically composed of a circle, potentially overlaid by a "dot" representing the checked state. Certain visual styles use an accent color for the checked state: some of these use it for the background of the circle, others for its foreground.

In both cases, the 'accent-color' property styles the affected parts of the control. For visual styles that do not use an accent color for radio buttons, 'accent-color' will have no effect.

The following are all examples of radio buttons conformantly rendered with a blue 'accent-color':

Platform	Sample	Notes
Firefox 79 / Win		No use of accent color.
Chrome 81 / Win		No use of accent color.
Safari / Snow Leopard		Accent color used to generate checked-state "background" gradient.
Safari 13 / Mac / Light		Accent color as checked-state "foreground".
Safari 13 / Mac / Dark		Accent color as checked-state "foreground".
Chrome 83 / Win / Light		Accent color as checked-state "foreground".
Chrome 83 / Win / Dark		Accent color as checked-state "foreground", contrast-adjusted for dark mode backgrounds.

§ 7.2. Switching appearance: the 'appearance' property

Name:	***'appearance'***
Value:	none ¦ auto | textfield | menulist-button | <compat-auto>

Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	n/a
Computed value:	specified keyword
Canonical order:	per grammar
Animation type:	discrete

While the way most elements in a document look can be fully controlled by CSS, widgets are typically rendered by UAs using native UI controls of the host operating system, which can neither be replicated nor styled using CSS.

The term ***widget*** in this specification denotes elements that can have ***native appearance***, meaning that they are rendered like analogous native widgets or controls of the host operating system or platform, or with a look and feel not otherwise expressible in CSS. It is up to the host language (e.g., HTML [HTML]) to define which elements can have native appearance.

This specification introduces the 'appearance' property to provide some control over this behavior. In particular, using 'appearance: none' allows authors to suppress the native appearance of widgets, so that CSS can be used to restyle them.

▶ **Note on the history of this feature**

'none'
: The element is rendered following the usual rules of CSS. Replaced elements other than widgets are not affected by this and remain replaced elements. Widgets must not have their native appearance. See § 7.2.1 Effects of appearance on Decorative Aspects of Elements and § 7.2.2 Effects of appearance on Semantic Aspects of Elements for details.

'auto'
: Widgets may have their native appearance.

  Elements other than widgets must be rendered as if 'none' had been specified.

'textfield'

For `<input>` elements where the `type` attribute is in the Search state, the element is rendered as a "normal" text entry widget, similar to an `<input>` element where the `type` attribute is in the Text state.

For all other elements, this value has the same effect as 'auto'.

'menulist-button'

For drop-down box `<select>` elements, the element is rendered as a drop-down box, including a "drop-down button", but not necessarily using a native control of the host operating system. For such elements, CSS properties such as 'color', 'background-color', and 'border' (that can be disregarded for 'auto') should not be disregarded.

For all other elements, this value has the same effect as 'auto'.

'<compat-auto>'

These values exist for compatibility of content developed for earlier non-standard versions of this property. They all have the same effect as 'auto'.

`<compat-auto>` = ***searchfield*** | ***textarea*** | ***push-button*** | ***slider-horizontal*** | ***checkbox*** | ***radio*** | ***square-button*** | ***menulist*** | ***listbox*** | ***meter*** | ***progress-bar*** | ***button***

ISSUE 5 When 'auto' is widely supported, recommend using it instead.

ISSUE 6 If any of these values are not needed for web compat, they should be dropped from this list; conversely, any value needed for compat not included in this list should be added.

ISSUE 7 It is possible that for compat reasons some of these values need to be promoted to being full blown values with effects on arbitrary elements, or that some of these values need to have some side effects on some widgets.

ISSUE 8 The values `slider-vertical` and `sliderthumb-vertical` are supported in some user agents to change the orientation of `<input type=range>` elements. These values are not specified because changing the orientation of this control is better done with a separate mechanism. See issue whatwg/html#4177.

EXAMPLE 13

An author wanting to customize the look and feel of check boxes in HTML could use the following:

```
input[type=checkbox] {
  all: unset; /* this shorthand includes resetting appearance to none */
  width: 1em;
  height: 1em;
  display: inline-block;
  background: red;
}
input[type=checkbox]:checked {
  border-radius: 50%;
  background: green;
}
input[type=checkbox]:focus-visible {
  outline: auto;
}
```

`<input type="checkbox">` would be rendered as , while `<input type="checkbox" checked>` would be rendered as , and activating (for example by clicking) the element would toggle the state as usual.

On widgets where the computed value is ‘auto’, ‘textfield’, ‘menulist-button’, or one of the <compat-auto> values, UAs may disregard some CSS properties to ensure that the intended appearance is preserved, or because these properties may not be meaningful for the chosen appearance. However, the following properties must not be disregarded:

- ‘appearance’
- ‘display’ (the inner display type may be ignored)
- ‘visibility’
- ‘position’
- ‘top’
- ‘right’
- ‘bottom’
- ‘left’

- ‘float’
- ‘clear’
- ‘margin’ and related long-hand properties
- ‘unicode-bidi’
- ‘direction’
- ‘cursor’
- ‘z-index’

ISSUE 9 Are there more properties that should be included in this list? Should we remove some? Should we whitelist the properties that are ok to ignore instead of blacklisting those that are not? Either way, UAs do ignore some properties when rendering widgets, so this specification needs to say something about this.

For compatibility with legacy content, UAs must also support ***‘-webkit-appearance’*** as a legacy name alias of ‘appearance’.

§ 7.2.1. Effects of ‘appearance’ on Decorative Aspects of Elements

All decorative visual aspects of a widget which are not caused by a CSS style rule must be suppressed when the appearance is changed using ‘appearance’, even if the UA’s internal representation for this element was composed of multiple elements or pseudo-elements combined together.

EXAMPLE 14
For example, the [HTML] `<select>` element is often rendered with an arrow on the right side indicating that the list will be expanded if the element is clicked. If the computed value of ‘appearance’ on a `<select>` element is ‘none’ this must disappear.

UAs should include in their user agent stylesheet style rules to give widgets a recognizable shape when ‘appearance’ is ‘none’.

Note: Authors may therefore need to override these UA style rules to get the styling they intended.

Authors may find it convenient to use 'all: unset', to get fully unstyled widgets, which they can then style as they want without interference from the user agent's stylesheet. However, this also suppresses the focus indicators provided by the same UA stylesheet. In order to avoid damaging accessibility, authors who do so should restore a focus indicator, for instance by using ':focus-visible { outline: auto; }'.

§ 7.2.2. Effects of 'appearance' on Semantic Aspects of Elements

UAs must preserve aspects of the widget which are necessary to operate the widget with its original semantics. The UA may however give them a different look and feel as long as it remains possible to operate the widget. Aspects of a widget which are merely needed to observe the state the widget is in and are not needed to operate it should be suppressed as well when the same information can be conveyed using ordinary CSS. Document languages should specify for each widget that they define what needs to be preserved.

EXAMPLE 15

For example, the slider of an `<input type=range>` is preserved (or replaced by an equivalent mechanism) even if 'appearance' is 'none', as it would otherwise not be possible to change the value of the range with the mouse or touchscreen.

On the other hand, the check mark in an `<input type=checkbox checked>` must be suppressed, as it merely indicates the state the element is in, which can be styled in different ways using the ':checked' selector.

The behavior and semantics of elements remain defined by the document language, and are not influenced by this property.

EXAMPLE 16

For example, regardless of the computed value of 'appearance':

- Elements which can be in different states keep this ability, and the relevant pseudo-classes match as usual.
- If a `<select>` element is activated, the UI to choose one of the associated `<option>` elements is shown (although it may look different).
- Event handlers attached to the element trigger as usual.

Conversely, changing the appearance of an element must not cause it to acquire the semantics, pseudo-classes, event handlers, etc. that are typical of the element whose appearance it acquires.

> EXAMPLE 17
> For example, neither ‘:enabled’ nor ‘:disabled’ match on a `<div>` styled with ‘appearance: button’. The ability for an element to be focused is also not changed by the ‘appearance’ property.

§ 7.3. Control Specific Rules

> ISSUE 10 Maybe some or all of this section should be moved to the [HTML] spec. at some point.

> ISSUE 11 On some elements, setting some properties inhibits ‘appearance: auto’. For instance, even if ‘appearance’ is ‘auto’, buttons lose their native appearance when a ‘border’ is set. This is not in contradiction with the definition of ‘auto’, since ‘appearance: auto’ only means UAs *may* use a native appearance, but for greater interoperability, it would be good to document which properties on which elements have this effect.

> ISSUE 12 Documenting what UAs need to put in their UA stylesheet would be good.

§ 7.3.1. Single-Line Text Input Controls

When ‘appearance’ is ‘auto’, single-line text input controls such as [HTML] `<input type=text>` must be rendered so that:

- The content is clipped in the inline direction to the content edge
- The content is clipped in the block direction to the padding edge
- The content is vertically centered
- The content does not wrap
- The ‘text-overflow’ property applies regardless of the value of the ‘overflow’ property

✧ ✧ ✧

§ Appendix A. Acknowledgments

This appendix is *informative*.

This specification builds upon [CSS-UI-3], which was edited and written for the most part by Tantek Çelik from 1999 to the present, first while representing Microsoft, then as an Invited Expert, and most recently while representing Mozilla.

Thanks to feedback and contributions from Rossen Atanassov, Tab Atkins, L. David Baron, Bert Bos, Matthew Brealey, Rick Byers, Ada Chan, James Craig, Michael Cooper, Axel Dahmen, Michael Day, Micah Dubinko, Elika E., Steve Falkenburg, Andrew Fedoniouk, Al Gilman, Ian Hickson, Bjoern Hoehrmann, Alan Hogan, David Hyatt, Richard Ishida, Sho Kuwamoto, Yves Lafon, Stuart Langridge, Susan Lesch, Peter Linss, Kang-Hao Lu, Masayuki Nakano, Mats Palmgren, Brad Pettit, Chris Rebert, François Remy, Andrey Rybka, Simon Sapin, Alexander Savenkov, Sebastian Schnitzenbaumer, Lea Verou, Etan Wexler, David Woolley, Frank Yan, Boris Zbarsky, and Domel.

§ Appendix B. Changes

This appendix is *informative*.

§ Changes from the 24 January 2020 Working Draft

In addition to a number of editorial adjustments, the following normative changes have been made:

- Changed appearance: button to only apply to buttons.
- Outline-width may be ignored if outline-style is auto.
- Ensured caret visibility despite transforms.
- Moved ‘resize: block’ and ‘resize: inline’ from css-logical-1 to this specification.
- Introduced the ‘accent-color’ property.

§ Changes from the 2 January 2020 Working Draft

In addition to a number of editorial adjustments, the following normative changes have been made:

- Defined user-select to be discretely animatable, rather than not.

- Restricted the applicability of ‘appearance: button’ so that it can no longer give a buttony appearance to widgets that aren’t buttons to start with.
- Add a clause to preserve visibility of carets even when scaled down via transforms.

§ Changes from the 22 December 2017 Working Draft

In addition to a number of editorial adjustments, the following normative changes have been made:

- Added details about form control specific rendering rules
- Clarify interaction between text-overflow and anonymous block containers
- Require the ‘pointer’ cursor for hyperlinks in all document formats
- Moved the ‘box-sizing’ and ‘text-overflow’ properties to [CSS-SIZING-3] and [CSS-OVERFLOW-4] respectively.
- Changed the requirement that the ‘text’ cursor matches the writing mode from MAY to MUST
- The definition of the appearance property has been significantly reworked to make the design compatible with existing web content
- The logic that mapped ‘user-select: auto’ to various values at computed value time has been changed to used value time
- Clarify Author requirements on the ‘cursor: pointer’ value

§ Changes from the 22 Sep 2015 First Public Working Draft

In addition to a number of editorial adjustments, the following normative changes have been made:

- The caret-animation property have been removed. It may be reintroduced later given sufficient evidence for use cases.
- The ‘resize’ property now also applies to replaced elements representing images or videos, and to iframes.
- The string values and dual values of the text-overflow property were moved to this specification from level 3.
- Move directional focus navigation properties from level 3 to level 4
- Content from the now stable [CSS-UI-3] has been merged in.

§ Appendix C. Considerations for Security and Privacy

This appendix is *informative*.

The W3C TAG is developing a Self-Review Questionnaire: Security and Privacy for editors of specifications to informatively answer.

Per the Questions to Consider

Does this specification deal with personally-identifiable information?

No.

Does this specification deal with high-value data?

No.

Does this specification introduce new state for an origin that persists across browsing sessions?

No.

Does this specification expose persistent, cross-origin state to the web?

No.

Does this specification expose any other data to an origin that it doesn't currently have access to?

No.

Does this specification enable new script execution/loading mechanisms?

Yes to loading, but not to execution.

The 'cursor' property accepts <image> values which may include URLs to be loaded. These may be SVG documents which may contain scripts, but this specification requires that scripts must not be run.

Does this specification allow an origin access to a user's location?

No.

Does this specification allow an origin access to sensors on a user's device?

Yes.

The directional focus navigation properties indirectly allow access to the device's keyboard navigation input mechanism such as arrow keys.

Does this specification allow an origin access to aspects of a user's local computing environment?

No.

Does this specification allow an origin access to other devices?

No.

Does this specification allow an origin some measure of control over a user agent's native UI?

Yes.

The ‘cursor’ and ‘caret’ property and sub-properties enable the page to change the display of the cursor and text insertion caret of the user agent’s native UI. In addition the ‘outline-style’ property’s ‘auto’ value (and thus ‘outline’ shorthand) enable the page to potentially display a native focused element outline presentation around any element.

The ‘appearance’ property also allows author to turn off native styling and replace it with css based styling.

Does this specification expose temporary identifiers to the web?

No.

Does this specification distinguish between behavior in first-party and third-party contexts?

No.

How should this specification work in the context of a user agent’s "incognito" mode?

No differently.

Does this specification persist data to a user’s local device?

No.

Does this specification have a "Security Considerations" and "Privacy Considerations" section?

Yes.

Does this specification allow downgrading default security characteristics?

No.

§ Appendix D. Default style sheet additions for HTML

This appendix is *informative*.

Potential additions to the base style sheet to express HTML form controls, and a few dynamic presentation attributes:

```
:enabled:focus {
  outline: 2px inset;
}

button,
input[type=button],
input[type=reset],
input[type=submit],
input[type=checkbox],
```

```
input[type=radio],
textarea,
input,
input[type=text],
input[type=password],
input[type=image] {
  display: inline-block;
}

input[type=button],
input[type=reset],
input[type=submit],
input[type=checkbox],
input[type=radio],
input,
input[type=text],
input[type=password],
input[type=image] {
  white-space: nowrap;
}

button {
  /* white space handling of BUTTON tags in particular */
  white-space:normal;
}

input[type=reset]:lang(en) {
  /* default content of HTML input type=reset button, per language */
  content: "Reset";
}

input[type=submit]:lang(en) {
  /* default content of HTML input type=submit button, per language */
  content: "Submit";
}

/* UAs should use language-specific Reset/Submit rules for others. */

input[type=button],
input[type=reset][value],
input[type=submit][value] {
  /* text content/labels of HTML "input" buttons */
  content: attr(value);
```

```
}

textarea {
  /* white space handling of TEXTAREA tags in particular */
  white-space:pre-wrap;
  resize: both;
}

input[type=hidden] {
  /* appearance of the HTML hidden text field in particular */
  display: none !important;
}

input[type=image] {
  content: attr(src,url);
  border: none;
}

select[size] {
  /* HTML4/XHTML1 <select> w/ size more than 1 - appearance of list */
  display: inline-block;
  height: attr(size,em);
}

select,select[size=1] {
  /* HTML4/XHTML1 <select> without size, or size=1 - popup-menu */
  display: inline-block;
  height: 1em;
  overflow: hidden;
}

select[size]:active {
  /* active HTML <select> w/ size more than 1 - appearance of active list */
  display: inline-block;
}

optgroup, option {
  display: block;
  white-space: nowrap;
}

optgroup[label],option[label] {
  content: attr(label);
```

```
}

option[selected]::before {
  display: inline;
  content: check;
}

/* Though FRAME resizing is not directly addressed by this specification,
   the following rules may provide an approximation of reasonable behavior. */

/*

frame { resize:both }
frame[noresize] { resize:none }

*/
```

§ 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 18
>
> 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.

§ 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 https://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

searchfield, in §7.2

se-resize, in §5.1.1

slider-horizontal, in §7.2

square-button, in §7.2

s-resize, in §5.1.1

sw-resize, in §5.1.1

<target-name>, in §5.3.1

text

- value for cursor, in §5.1.1
- value for user-select, in §6.1

textarea, in §7.2

textfield, in §7.2

underscore, in §5.2.2

user-select, in §6.1

vertical-text, in §5.1.1

visible character, in §5.2.2

wait, in §5.1.1

-webkit-appearance, in §7.2

-webkit-user-select, in §6.1

widget, in §7.2

w-resize, in §5.1.1

zoom-in, in §5.1.1

zoom-out, in §5.1.1

§ Terms defined by reference

[css-backgrounds-3] defines the following terms:

- <line-style>
- <line-width>
- background-color
- background-image
- border
- border-radius
- border-width
- none

[css-box-4] defines the following terms:

- border edge
- content edge
- margin
- padding edge

[css-cascade-5] defines the following terms:

- all
- computed value
- legacy name alias
- sub-property
- used value
- user-agent origin

[css-color-3] defines the following terms:

- <color>

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

- color
- currentcolor

[css-display-3] defines the following terms:

- display
- inner display type

[css-images-3] defines the following terms:

- <image>
- concrete object size
- default object size
- default sizing algorithm
- natural aspect ratio
- natural size

[css-overflow-3] defines the following terms:

- overflow
- visible

[CSS-OVERFLOW-4] defines the following terms:

- text-overflow

[css-page-3] defines the following terms:

- page-margin boxes

[css-position-3] defines the following terms:

- left
- position

[css-pseudo-4] defines the following terms:

- ::after
- ::before
- ::first-letter
- ::first-line
- ::marker

[CSS-SIZING-3] defines the following terms:

- box-sizing
- height
- width

[CSS-UI-3] defines the following terms:

- auto

[CSS-VALUES-3] defines the following terms:

- <length>
- <number>
- <string>
- <url>

[css-values-4] defines the following terms:

- *
- ,
- ?
- css-wide keywords
- |
- ||

[CSS-WRITING-MODES-3] defines the following terms:

- direction
- unicode-bidi

[css-writing-modes-4] defines the following terms:

- block size
- block-axis
- inline size
- inline-axis
- under
- writing mode

[CSS21] defines the following terms:

- bottom
- clear
- float
- max-height
- max-width
- min-height
- min-width
- right
- top
- visibility
- z-index

[HTML] defines the following terms:

- canvas
- div
- drop-down box
- iframe
- img
- input
- mutable
- object
- option
- picture
- select
- textarea
- type
- video

[selectors-4] defines the following terms:

- :checked
- :disabled
- :enabled
- :focus

[SVG11] defines the following terms:

- cursor

[SVG2] defines the following terms:

- svg
- textpath

§ References

§ Normative References

[CSS-BACKGROUNDS-3]
Bert Bos; Elika Etemad; Brad Kemper. CSS Backgrounds and Borders Module Level 3. 22 December 2020. CR. URL: https://www.w3.org/TR/css-backgrounds-3/

[CSS-BOX-4]
Elika Etemad. CSS Box Model Module Level 4. 21 April 2020. WD. URL: https://www.w3.org/TR/css-box-4/

[CSS-CASCADE-5]
Elika Etemad; Miriam Suzanne; Tab Atkins Jr.. CSS Cascading and Inheritance Level 5. 19 January 2021. WD. URL: https://www.w3.org/TR/css-cascade-5/

[CSS-COLOR-3]
Tantek Çelik; Chris Lilley; David Baron. CSS Color Module Level 3. 19 June 2018. REC. URL: https://www.w3.org/TR/css-color-3/

[CSS-COLOR-4]
Tab Atkins Jr.; Chris Lilley. CSS Color Module Level 4. 12 November 2020. WD. URL: https://www.w3.org/TR/css-color-4/

[CSS-DISPLAY-3]

Tab Atkins Jr.; Elika Etemad. CSS Display Module Level 3. 18 December 2020. CR. URL: https://www.w3.org/TR/css-display-3/

[CSS-IMAGES-3]
Tab Atkins Jr.; Elika Etemad; Lea Verou. CSS Images Module Level 3. 17 December 2020. CR. URL: https://www.w3.org/TR/css-images-3/

[CSS-OVERFLOW-3]
David Baron; Elika Etemad; Florian Rivoal. CSS Overflow Module Level 3. 3 June 2020. WD. URL: https://www.w3.org/TR/css-overflow-3/

[CSS-OVERFLOW-4]
David Baron; Florian Rivoal. CSS Overflow Module Level 4. 13 June 2017. WD. URL: https://www.w3.org/TR/css-overflow-4/

[CSS-PAGE-3]
Elika Etemad; Simon Sapin. CSS Paged Media Module Level 3. 18 October 2018. WD. URL: https://www.w3.org/TR/css-page-3/

[CSS-POSITION-3]
Elika Etemad; et al. CSS Positioned Layout Module Level 3. 19 May 2020. WD. URL: https://www.w3.org/TR/css-position-3/

[CSS-PSEUDO-4]
Daniel Glazman; Elika Etemad; Alan Stearns. CSS Pseudo-Elements Module Level 4. 31 December 2020. WD. URL: https://www.w3.org/TR/css-pseudo-4/

[CSS-SIZING-3]
Tab Atkins Jr.; Elika Etemad. CSS Box Sizing Module Level 3. 18 December 2020. WD. URL: https://www.w3.org/TR/css-sizing-3/

[CSS-TRANSFORMS-1]
Simon Fraser; et al. CSS Transforms Module Level 1. 14 February 2019. CR. URL: https://www.w3.org/TR/css-transforms-1/

[CSS-UI-3]
Tantek Çelik; Florian Rivoal. CSS Basic User Interface Module Level 3 (CSS3 UI). 21 June 2018. REC. URL: https://www.w3.org/TR/css-ui-3/

[CSS-VALUES-3]
Tab Atkins Jr.; Elika Etemad. CSS Values and Units Module Level 3. 6 June 2019. CR. URL: https://www.w3.org/TR/css-values-3/

[CSS-VALUES-4]
Tab Atkins Jr.; Elika Etemad. CSS Values and Units Module Level 4. 11 November 2020. WD. URL: https://www.w3.org/TR/css-values-4/

[CSS-WRITING-MODES-3]

Elika Etemad; Koji Ishii. CSS Writing Modes Level 3. 10 December 2019. REC. URL: https://www.w3.org/TR/css-writing-modes-3/

[CSS-WRITING-MODES-4]
Elika Etemad; Koji Ishii. CSS Writing Modes Level 4. 30 July 2019. CR. URL: https://www.w3.org/TR/css-writing-modes-4/

[CSS21]
Bert Bos; et al. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. 7 June 2011. REC. URL: https://www.w3.org/TR/CSS21/

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

[PNG]
Tom Lane. Portable Network Graphics (PNG) Specification (Second Edition). 10 November 2003. REC. URL: https://www.w3.org/TR/PNG/

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

[SELECTORS-4]
Elika Etemad; Tab Atkins Jr.. Selectors Level 4. 21 November 2018. WD. URL: https://www.w3.org/TR/selectors-4/

[SVG11]
Erik Dahlström; et al. Scalable Vector Graphics (SVG) 1.1 (Second Edition). 16 August 2011. REC. URL: https://www.w3.org/TR/SVG11/

[SVG2]
Amelia Bellamy-Royds; et al. Scalable Vector Graphics (SVG) 2. 4 October 2018. CR. URL: https://www.w3.org/TR/SVG2/

[UAX29]
Mark Davis; Christopher Chapman. Unicode Text Segmentation. 19 February 2020. Unicode Standard Annex #29. URL: https://www.unicode.org/reports/tr29/tr29-37.html

§ Informative References

[CSS1]
Håkon Wium Lie; Bert Bos. Cascading Style Sheets, level 1. 13 September 2018. REC. URL: https://www.w3.org/TR/CSS1/

[CSS4-IMAGES]

Tab Atkins Jr.; Elika Etemad; Lea Verou. CSS Image Values and Replaced Content Module Level 4. 13 April 2017. WD. URL: https://www.w3.org/TR/css-images-4/

[HTML5]
Ian Hickson; et al. HTML5. 27 March 2018. REC. URL: https://www.w3.org/TR/html5/

[SELECT]
Tantek Çelik; et al. Selectors Level 3. 6 November 2018. REC. URL: https://www.w3.org/TR/selectors-3/

§ Property Index

Name	Value	Initial	Applies to	Inh.	%ages	Anim-ation type	Canonical order	Computed value
'accent-color'	auto | <color>	auto	all elements	yes	N/A	by computed value type	per grammar	the keyword auto or a computed color
'appearance'	none | auto | textfield | menulist-button | <compat-auto>	none	all elements	no	n/a	discrete	per grammar	specified keyword
'caret'	<'caret-color'> || <'caret-shape'>	auto	elements that accept input	yes	N/A	see individual properties	per grammar	see individual properties
'caret-color'	auto | <color>	auto	all elements	yes	N/A	by computed value	per grammar	The computed value for auto is auto. For <color> values, see [[!CSS-COLOR-4#resolving-color-values]] in [CSS-COLOR-4].
'caret-shape'	auto | bar | block | underscore	auto	elements that accept input	yes	N/A	by computed value	per grammar	specified keyword

Name	Value	Initial	Applies to	Inh.	%ages	Anim-ation type	Canonical order	Computed value
'cursor'	[[<url> [<x> <y>]?,]* [auto | default | none | context-menu | help | pointer | progress | wait | cell | crosshair | text | vertical-text | alias | copy | move | no-drop | not-allowed | grab | grabbing | e-resize | n-resize | ne-resize | nw-resize | s-resize | se-resize | sw-resize | w-resize | ew-resize | ns-resize | nesw-resize | nwse-resize | col-resize | row-resize | all-scroll | zoom-in | zoom-out]]	auto	all elements	yes	N/A	discrete	per grammar	as specified, except with any relative URLs converted to absolute
'nav-down'	auto | <id> [current | root | <target-name>]?	auto	all enabled elements	no	N/A	discrete	per grammar	as specified
'nav-left'	auto | <id> [current | root | <target-name>]?	auto	all enabled elements	no	N/A	discrete	per grammar	as specified
'nav-right'	auto | <id> [current | root | <target-name>]?	auto	all enabled elements	no	N/A	discrete	per grammar	as specified
'nav-up'	auto | <id> [current | root | <target-name>]?	auto	all enabled elements	no	N/A	discrete	per grammar	as specified
'outline'	[<'outline-color'> || <'outline-style'> || <'outline-width'>]	see individual properties	all elements	no	N/A	see individual properties	per grammar	see individual properties

Name	Value	Initial	Applies to	Inh.	%ages	Anim-ation type	Canonical order	Computed value
'outline-color'	<color> | invert	invert	all elements	no	N/A	by computed value	per grammar	The computed value for invert is invert. For <color> values, see [[!CSS-COLOR-4#resolving-color-values]] in [CSS-COLOR-4].
'outline-offset'	<length>	0	all elements	no	N/A	by computed value	per grammar	absolute length
'outline-style'	auto | <outline-line-style>	none	all elements	no	N/A	by computed value	per grammar	specified keyword
'outline-width'	<line-width>	medium	all elements	no	N/A	by computed value	per grammar	absolute length; 0 if the outline style is none.
'resize'	none | both | horizontal | vertical | block | inline	none	elements with overflow other than visible, and optionally replaced elements such as images, videos, and iframes	no	N/A	discrete	per grammar	specified keyword

Name	Value	Initial	Applies to	Inh.	%ages	Anim-ation type	Canonical order	Computed value
'user-select'	auto | text | none | contain | all	auto	all elements, and optionally to the ::before and ::after pseudo-elements	no	n/a	discrete	per grammar	specified keyword

§ Issues Index

ISSUE 1 if we allow user-select to change to a value other than 'none', we need to figure out what this means for copyability, and DOM APIs. ↵

ISSUE 2 Should it also apply to '::marker'? To page-margin boxes? Should the used value of 'auto' also be 'none', or would 'text' be more appropriate? ↵

ISSUE 3 Should there be constraints on what happens to the used value on elements that are editable descendants of editing hosts? The semantics are not obvious. Maybe 'none' should map to 'text', or maybe all values should map to 'text'. ↵

ISSUE 4 the list above is incomplete and needs to include at least the various button-like variants of `<input>`. ↵

ISSUE 5 When 'auto' is widely supported, recommend using it instead. ↵

ISSUE 6 If any of these values are not needed for web compat, they should be dropped from this list; conversely, any value needed for compat not included in this list should be added. ↵

ISSUE 7 It is possible that for compat reasons some of these values need to be promoted to being full blown values with effects on arbitrary elements, or that some of these values need to have some side effects on some widgets. ↵

ISSUE 8 The values `slider-vertical` and `sliderthumb-vertical` are supported in some user agents to change the orientation of `<input type=range>` elements. These values are not specified because changing the orientation of this control is better done with a separate mechanism. See issue whatwg/html#4177. ↲

ISSUE 9 Are there more properties that should be included in this list? Should we remove some? Should we whitelist the properties that are ok to ignore instead of blacklisting those that are not? Either way, UAs do ignore some properties when rendering widgets, so this specification needs to say something about this. ↲

ISSUE 10 Maybe some or all of this section should be moved to the [HTML] spec. at some point. ↲

ISSUE 11 On some elements, setting some properties inhibits 'appearance: auto'. For instance, even if 'appearance' is 'auto', buttons lose their native appearance when a 'border' is set. This is not in contradiction with the definition of 'auto', since 'appearance: auto' only means UAs *may* use a native appearance, but for greater interoperability, it would be good to document which properties on which elements have this effect. ↲

ISSUE 12 Documenting what UAs need to put in their UA stylesheet would be good. ↲