CSS Generated Content Module Level 3

Editor’s Draft, 15 December 2020

▶ **Specification Metadata**

Abstract

This CSS3 Module describes how to insert content 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 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-content” in the title, like this: “[css-content] *…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 is a very rough draft, and is not ready for implementation.

Table of Contents

Introduction

Authors sometimes want user agents to render content that does not come from the document tree. One familiar example of this is numbered headings; the author does not want to mark the numbers up explicitly, they want the user agent to generate them automatically. Counters and markers are used to achieve these effects.

EXAMPLE 1

```
h1::before { content: counter(section) ": "; }
```

Similarly, authors may want the user agent to insert the word "Figure" before the caption of a figure, or "Chapter 7" on a line before the seventh chapter title.

EXAMPLE 2

```
chapter { counter-increment: chapter; }
chapter > title::before { content: "Chapter " counter(chapter) "\A"; }
```

Another common effect is replacing elements with images or other multimedia content. Since not all user agents support all multimedia formats, fallbacks may have to be provided.

EXAMPLE 3

```
/* Replace <logo> elements with the site’s logo, using a format
 * supported by the UA */
logo { content: url(logo.mov), url(logo.mng), url(logo.png), none; }

/* Replace <figure> elements with the referenced document, or,
 * failing that, with either the contents of the alt attribute or the
 * contents of the element itself if there is no alt attribute */
figure[alt] { content: attr(href url), attr(alt); }
figure:not([alt]) { content: attr(href url), contents; }
```

Value Definitions

This specification follows the CSS property definition conventions from [CSS2] 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.

§ 1. Inserting and replacing content with the ‘content’ property

Name:	***‘content’***
Value:	normal | none | [<content-replacement> | <content-list>] [/ [<string> | <counter>]+]?
Initial:	normal
Applies to:	all elements, tree-abiding pseudo-elements, and page margin boxes
Inherited:	no
Percentages:	n/a

Computed value:	See prose below
Canonical order:	per grammar
Animation type:	discrete

User Agents are expected to support this property on all media, including non-visual ones.

The ‘content’ property dictates what is rendered inside an element or pseudo-element.

For elements, it has only one purpose: specifying that the element renders as normal, or replacing the element with an image (and possibly some associated "alt text").

For pseudo-elements and margin boxes, it is more powerful. It controls whether the element renders at all, can replace the element with an image, or replace it with arbitrary inline content (text and images).

‘normal’

For an element or page margin box, this computes to ‘contents’.

For ‘::before’ and ‘::after’, this computes to ‘none’.

For ‘::marker’, this computes to itself (‘normal’).

‘none’

On elements, this inhibits the children of the element from being rendered as children of this element, as if the element was empty.
On pseudo-elements it inhibits the creation of the pseudo-element as if it had ‘display: none’.

In neither case does it prevent any pseudo-elements which have this element or pseudo-element as an originating element from being generated.

‘<content-replacement>’

Equal to:

```
<image>
```

Makes the element or pseudo-element a replaced element, filled with the specified <image>. Its normal contents are suppressed and do not generate boxes, as if they were ‘display: none’.

If the <image> represents an invalid image, then it must be treated as instead representing an image with zero natural width and height, filled with transparent black.

ISSUE 1 The above invalid image behavior appears to be what Chrome is doing. Is this okay? Is there a better behavior we can/should use?

Note: Replaced elements use different layout rules than normal elements. (In effect, it becomes equivalent to an HTML `<img>` element.)

Note: Replaced elements do not have '::before' or '::after' pseudo-elements; the 'content' property replaces their entire contents.

'<content-list>'

Equal to:

```
[ <string> | contents | <image> | <counter> | <quote> | <target> | <leader()>
```

Replaces the element's contents with one or more anonymous inline boxes corresponding to the specified values, in the order specified. Its normal contents are suppressed and do not generate boxes, as if they were 'display: none'.

Each value contributes an inline box to the element's contents. For <image>, this is an inline anonymous replaced element; for the others, it's an anonymous inline run of text.

If an <image> represents an invalid image, the user agent must do one of the following:

- "Skip" the <image>, generating nothing for it.
- Display some indication that the image can't be displayed in place of the <image>, such as a "broken image" icon.

This specification intentionally does not define which behavior a user agent must use, but it must use one or the other consistently.

Note: If the value of <content-list> is a single <image>, it must instead be interpreted as a <content-replacement>.

'/ [<string> | <counter>]+'

Specifies the "alt text" for the element. See § 1.2 Alternative Text for Accessibility for details. If omitted, the element has no "alt text".

ISSUE 2 Should the contents keyword be replaced with ‘content()’?

§ 1.1. Accessibility of Generated Content

Generated content should be searchable, selectable, and available to assistive technologies. The ‘content’ property applies to speech and generated content must be rendered for speech output. [CSS3-SPEECH]

ISSUE 3 Start work on an AAM for CSS.

§ 1.2. Alternative Text for Accessibility

Content intended for visual media sometimes needs alternative text for speech output or other non-visual mediums. The ‘content’ property thus accepts alternative text to be specified after a slash (‘/’) after the last <content-list>. If such alternative text is provided, it must be used for speech output instead.

This allows, for example, purely decorative text to be elided in speech output (by providing the empty string as alternative text), and allows authors to provide more readable alternatives to images, icons, or text-encoded symbols.

EXAMPLE 4
Here the content property is an image, so the alt value is required to provide alternative text.

```
.new::before {
 content: url(./img/star.png) / "New!";
  /* or a localized attribute from the DOM: attr("data-alt") */
}
```

EXAMPLE 5

If the pseudo-element is purely decorative and its function is covered elsewhere, setting alt to the empty string can avoid reading out the decorative element. Here the ARIA attribute will be spoken as "collapsed". Without the empty string alt value, the content would also be spoken as "Black right-pointing pointer".

```
.expandable::before {
 content: "\25BA" / "";
/* a.k.a. ► */
 /* aria-expanded="false" already in DOM,
   so this pseudo-element is decorative */
}
```

§ 2. <content-list> Values and Functions

The <content-list> value is used in ‘content’ to fill an element with one or more anonymous inline boxes, including images, strings, the values of counters, and the text value of elements. In this section we enumerate the possibilities.

§ 2.1. String

‘<string>’

Represents an anonymous inline box filled with the specified text.

Note: White space in the string is handled the same as in literal text, and controlled by the properties in [CSS-TEXT-3] and elsewhere. In particular, white space character can collapse, even across multiple strings, such as in ‘content: "First " " Second";’, which by default will render similar to `"First Second"` (with a single visible space between the two words).

§ 2.2. <image>

‘<image>’

Represents an anonymous inline replaced element filled with the specified <image>.

If the <image> represents an invalid image, this value instead represents nothing. (No inline content is added to the element, as if this value were "skipped".)

ISSUE 4 CSS2.1 explicitly allowed the UA to substitute a broken image icon if the image was invalid. However, no browser appears to do this. Is this removal okay?

§ 2.3. Element Content

'contents'

The element's descendants. Since this can only be used once per element (you can't duplicate the children if, e.g., one is a plugin or form control), it is handled as follows:

If set on the element:

Always honoured. Note that this is the default, since the initial value of 'content' is 'normal' and 'normal' computes to 'contents' on an element.

If set on one of the element's other pseudo-elements:

Check to see that it is not set on a "previous" pseudo-element, in the following order, depth first:

1. the element itself
2. ::before
3. ::after

ISSUE 5 Should this behave as an empty string on pseudo-elements?

If it is already used, then it evaluates to nothing (like 'none'). Only pseudo-elements that are actually generated are checked.

EXAMPLE 6
In the following case:

```
foo { content: normal; }  /* this is the initial value */
foo::after { content: contents; }
```

...the element's ‘content’ property would compute to ‘contents’ and the after pseudo element would have no contents (equivalent to ‘none’) and thus would not appear.

```
foo { content: none; }
foo::after { content: contents; }
```

But in this example, the ::after pseudo-element will contain the contents of the foo element.

ISSUE 6 Use cases for suppressing the content on the element and using it in a pseudo-element would be welcome.

Note: While it is useless to include ‘contents’ twice in a single ‘content’ property, that is not a parse error. The second occurrence simply has no effect, as it has already been used. It is also not a parse error to use it on a ‘::marker’ pseudo-element, it is only during the rendering stage that it gets treated like ‘none’.

ISSUE 7 Do we need the statement about marker pseudo-elements here? Or is this legacy from the old version of the spec?

§ 2.4. Quotes

HTML has long had the `q` element, used to delimit quotations. The ‘quotes’ property, in conjunction with the various ‘*-quote’ values of the ‘content’ property, can be used to properly style such quotations.

§ 2.4.1. Specifying quotes with the ‘quotes’ property

Name:	***'quotes'***
Value:	auto | none | [<string> <string>]+
Initial:	auto
Applies to:	all elements
Inherited:	yes
Percentages:	n/a
Computed value:	the keyword 'none', the keyword 'auto', or a list, each item a pair of string values
Canonical order:	per grammar
Animation type:	discrete

User Agents are expected to support this property on all media, including non-visual ones.

This property specifies quotation marks for any number of embedded quotations. Values have the following meanings:

'none'
: The 'open-quote' and 'close-quote' values of the 'content' property produce no quotations marks, as if they were 'no-open-quote' and 'no-close-quote' respectively.

'auto'
: A typographically appropriate used value for 'quotes' is automatically chosen by the UA based on the content language of the element and/or its parent.

> Note: The Unicode Common Locale Data Repository [CLDR] maintains information on typographically appropriate quotation marks. UAs can use other sources of information as well, particularly as typographic preferences can vary; however it is encouraged to submit any improvements to Unicode so that the entire software ecosystem can benefit.

[<string> <string>]+
: Values for the 'open-quote' and 'close-quote' values of the 'content' property are taken from this list of pairs of quotation marks (opening and closing). The first (leftmost) pair represents the

outermost level of quotation, the second pair the first level of embedding, etc. The user agent must apply the appropriate pair of quotation marks according to the level of embedding.

§ 2.4.2. The *-quote values of the content property

```
<quote> = open-quote | close-quote | no-open-quote | no-close-quote
```

‘open-quote’
‘close-quote’
: These values are replaced by the appropriate string from the ‘quotes’ property, and increments (decrements) the level of nesting for quotes. See § 2.4.1 Specifying quotes with the quotes property for more information.

‘no-open-quote’
‘no-close-quote’
: Inserts nothing (as in ‘none’), but increments (decrements) the level of nesting for quotes. See § 2.4.1 Specifying quotes with the quotes property for more information.

Quotation marks are inserted in appropriate places in a document with the ‘open-quote’ and ‘close-quote’ values of the ‘content’ property. Each occurrence of ‘open-quote’ or ‘close-quote’ is replaced by one of the strings from the value of ‘quotes’, based on the depth of nesting.

‘open-quote’ refers to the first of a pair of quotes, ‘close-quote’ refers to the second. Which pair of quotes is used depends on the nesting level of quotes: the number of occurrences of ‘open-quote’ in all generated text before the current occurrence, minus the number of occurrences of ‘close-quote’. If the depth is 0, the first pair is used, if the depth is 1, the second pair is used, etc. If the depth is greater than the number of pairs, the last pair is repeated.

Note that this quoting depth is independent of the nesting of the source document or the formatting structure.

Note: Quote nesting, like counter inheritance, operates on the “flattened element tree” in the context of the [DOM].

Some typographic styles require open quotation marks to be repeated before every paragraph of a quote spanning several paragraphs, but only the last paragraph ends with a closing quotation mark. In CSS, this can be achieved by inserting "phantom" closing quotes. The keyword ‘no-close-quote’ decrements the quoting level, but does not insert a quotation mark.

EXAMPLE 7

The following style sheet puts opening quotation marks on every paragraph in a `blockquote`, and inserts a single closing quote at the end:

```
blockquote p:before { content: open-quote }
blockquote p:after { content: no-close-quote }
blockquote p:last-child::after { content: close-quote }
```

For symmetry, there is also a ‘no-open-quote’ keyword, which inserts nothing, but increments the quotation depth by one.

Note: If a quotation is in a different language than the surrounding text, it is customary to quote the text with the quote marks of the language of the surrounding text, not the language of the quotation itself.

EXAMPLE 8

For example, French inside English:

> The device of the order of the garter is “Honi soit qui mal y pense.”

English inside French:

> Il disait: « Il faut mettre l’action en ‹ fast forward ›. »

A style sheet like the following will set the ‘quotes’ property so that ‘open-quote’ and ‘close-quote’ will work correctly on all elements. These rules are for documents that contain only English, French, or both. One rule is needed for every additional language. Note the use of the child combinator (">") to set quotes on elements based on the language of the surrounding text:

```
:lang(fr) > * { quotes: "\00AB\2005" "\2005\00BB" "\2039\2005" "\2005\203A" }
:lang(en) > * { quotes: "\201C" "\201D" "\2018" "\2019" }
```

The quotation marks are shown here in a form that most people will be able to type. If you can type them directly, they will look like this:

```
:lang(fr) > * { quotes: "« " " »" "‹ " " ›" }
:lang(en) > * { quotes: "“" "”" "‘" "’" }
```

EXAMPLE 9

For example, applying the following style sheet:

```
/* Specify pairs of quotes for two levels in two languages */
:lang(en) > q { quotes: '"' '"' "'" "'" }
:lang(no) > q { quotes: "«" "»" "’" "’" }

/* Insert quotes before and after Q element content */
q::before { content: open-quote }
q::after  { content: close-quote }
```

to the following HTML fragment:

```
<html lang="en">
  <head>
   <title>Quotes</title>
  </head>
  <body>
   <p><q>Quote me!</q></p>
  </body>
</html>
```

would allow a user agent to produce:

```
"Quote me!"
```

while this HTML fragment:

```
<html lang="no">
  <head>
   <title>Quotes</title>
  </head>
  <body>
   <p><q>Trøndere gråter når <q>Vinsjan på kaia</q> blir deklamert.</q></p>
  </body>
</html>
```

would produce:

```
«Trøndere gråter når ’Vinsjan på kaia’ blir deklamert.»
```

§ 2.5. Leaders

A leader, sometimes known as a tab leader or a dot leader, is a repeating pattern used to visually connect content across horizontal spaces. They are most commonly used in tables of contents, between titles and page numbers. The ‘leader()’ function, as a value for the content property, is used to create leaders in CSS. This function takes a string (the leader string), which describes the repeating pattern for the leader.

§ 2.5.1. The ‘leader()’ function

‘leader(<leader-type>)’
Inserts a leader. See the section on leaders for more information.

```
leader() = leader( <leader-type> )
<leader-type> = dotted | solid | space | <string>
```

Three keywords are shorthand values for common strings:

‘dotted’
Equivalent to ‘leader(".")’

‘solid’
Equivalent to ‘leader("_")’

‘space’
Equivalent to ‘leader(" ")’

‘<string>’
Issue: Define this.

EXAMPLE 10

```
ol.toc a::after {
  content: leader('.') target-counter(attr(href), page);
}

<h1>Table of Contents</h1>
<ol class="toc">
<li><a href="#chapter1">Loomings</a></li>
<li><a href="#chapter2">The Carpet-Bag</a></li>
<li><a href="#chapter3">The Spouter-Inn</a></li>
</ol>
```

This might result in:

```
Table of Contents

1. Loomings.....................1
2. The Carpet-Bag...............9
3. The Spouter-Inn.............13
```

ISSUE 8 Do leaders depend on the assumption that the content after the leader is right-aligned (end-aligned)?

§ 2.5.2. Rendering leaders

Consider a line which contains the content before the leader (the “before content”), the leader, and the content after the leader (the “after content”). Leaders obey the following rules:

1. The leader string must appear in full at least once.
2. The leader should be as long as possible
3. Visible characters in leaders should vertically align with each other when possible.
4. Line break characters in the leader string must be ignored.
5. White space in the leader string follows normal CSS rules.
6. A leader only appears between the start content and the end content.
7. A leader only appears on a single line, even if the before content and after content are on different

lines.

8. A leader can't be the only thing on a line.

§ 2.5.3. Procedure for rendering leaders

1. Lay out the *before content*, until reaching the line where the *before content* ends.

   ```
   BBBBBBBBBB
   BBB
   ```

2. The leader string consists of one or more glyphs, and is thus an inline box. A leader is a row of these boxes, drawn from the end edge to the start edge, where only those boxes not overlaid by the before or after content. On this line, draw the leader string, starting from the end edge, repeating as many times as possible until reaching the start edge.

   ```
   BBBBBBBBBB
   ..........
   ```

3. Draw the before and after content on top of the leader. If any part of the *before content* or *after content* overlaps a glyph in a leader string box, that glyph is not displayed.

   ```
   BBBBBBBBBB
   BBB....AAA
   ```

4. If one full copy of the leader string is not visible:

   ```
   BBBBBBB
   BBBBBBA
   ```

   Insert a line break after the *before content*, draw the leader on the next line, and draw the *after content* on top, and hide any leader strings that are not fully displayed.

   ```
   BBBBBBB
   BBBBBB
   ......A
   ```

ISSUE 9 what to do if *after content* is wider than the line box?

ISSUE 10 Leaders don’t quite work in table layouts. How can we fix this?

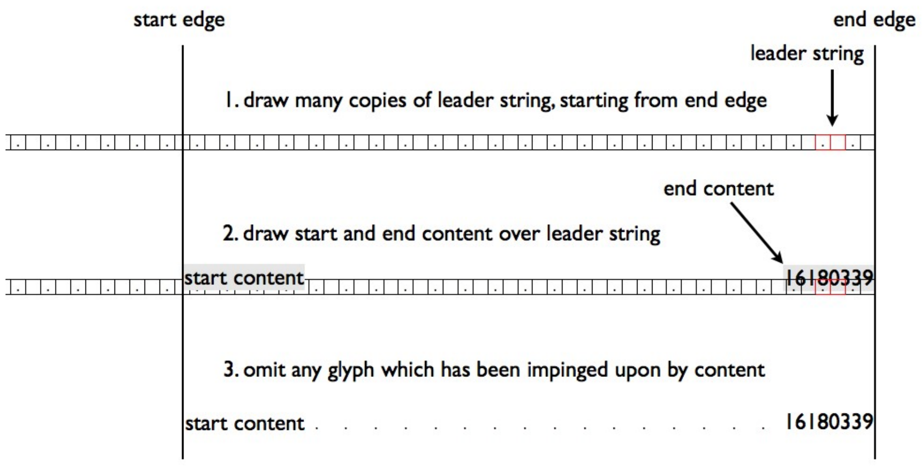

start edge
end edge
leader string
1. draw many copies of leader string, starting from end edge
end content
2. draw start and end content over leader string
start content
16180339
3. omit any glyph which has been impinged upon by content
start content
16180339

Figure 1 *Procedure for drawing leaders*

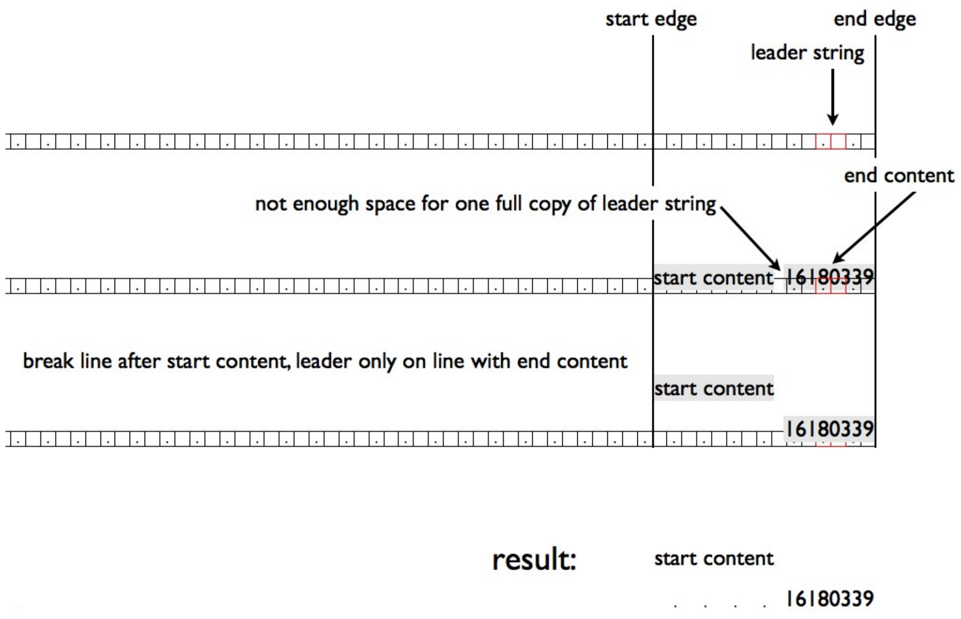

start edge
end edge
leader string
end content
not enough space for one full copy of leader string
start content
16180339
break line after start content, leader only on line with end content
start content
16180339
result:
start content
16180339

Figure 2 *Procedure for drawing leaders when the content doesn’t fit on a single line*

§ 2.6. Cross references and the target-* functions

Many documents contain internal references:

EXAMPLE 11

- See chapter 7
- in section 4.1
- on page 23

Three new values for the content property are used to automatically create these types of cross-references: ‘target-counter()’, ‘target-counters()’, and ‘target-text()’. Each of these displays information obtained from the target end of a link.

```
<target> = <target-counter()> | <target-counters()> | <target-text()>
```

See sections below for details on each of these.

§ 2.6.1. The ‘target-counter()’ function

```
target-counter() = target-counter( [ <string> | <url> ] , <custom-ident> , <counte
```

The ‘target-counter()’ function retrieves the value of the innermost counter with a given name. The required arguments are the url of the target and the name of the counter. An optional counter-style argument can be used to format the result.

These functions only take a fragment URL which points to a location in the current document. If there’s no fragment, if the ID referenced isn’t there, or if the URL points to an outside document, the user agent must treat that as an error.

ISSUE 11 what should error handling be?

ISSUE 12 restrict syntactically to local references for now.

EXAMPLE 12

HTML:

```
…which will be discussed on page <a href="#chapter4_sec2"></a>.
```

CSS:

```
a::after { content: target-counter(attr(href url), page) }
```

Result:

```
…which will be discussed on page 137.
```

EXAMPLE 13

Page numbers in tables of contents can be generated automatically:

HTML:

```
<nav>
  <ol>
   <li class="frontmatter"><a href="#pref_01">Preface</a></li>
   <li class="frontmatter"><a href="#intr_01">Introduction</a></li>
   <li class="bodymatter"><a href="#chap_01">Chapter One</a></li>
  </ol>
</nav>
```

CSS:

```
.frontmatter a::after { content: leader('.') target-counter(attr(href url), pa
.bodymatter a::after { content: leader('.') target-counter(attr(href url), page
```

Result:

```
Preface.............vii
Introduction.........xi
Chapter One...........1
```

§ 2.6.2. The ‘target-counters()’ function

This functions fetches the value of all counters of a given name from the end of a link, and formats them by inserting a given string between the value of each nested counter.

```
target-counters() = target-counters( [ <string> | <url> ] , <custom-ident> , <stri
```

EXAMPLE 14

```
I have not found a compelling example for target-counters() yet.
```

ISSUE 13 found a compelling example, in CSS specs. Do something.

§ 2.6.3. The ‘target-text()’ function

The ‘target-text()’ function retrieves the text value of the element referred to by the URL. An optional second argument specifies what content is retrieved, using the same values as the ‘string-set’ property above.

```
target-text() = target-text( [ <string> | <url> ] , [ content | before | after | f
```

ISSUE 14 A simpler syntax has been proposed by fantasai: http://lists.w3.org/Archives/Public/www-style/2012Feb/0745.html

EXAMPLE 15

```
…which will be discussed <a href="#chapter_h1_1">later</a>.
```

```
a::after { content: ", in the chapter entitled " target-text(attr(href url)) }
```

Result: …which will be discussed later, in the chapter entitled Loomings.

§ 2.7. Named strings

This section introduces ***named strings***, which are the textual equivalent of counters and which have a distinct namespace from counters. Named strings follow the same nesting rules as counters. The ‘string-set’ property accepts values similar to the ‘content’ property, including the extraction of the current value of counters.

Named strings are a convenient way to pull metadata out of the document for insertion into headers and footers. In HTML, for example, META elements contained in the document HEAD can set the value of named strings. In conjunction with attribute selectors, this can be a powerful mechanism:

EXAMPLE 16

```
meta[author] { string-set: author attr(author); }
head > title { string-set: title contents; }
@page:left {
  @top {
   text-align: left;
   vertical-align: middle;
   content: string(title);
  }
}
@page:right {
  @top {
   text-align: right;
   vertical-align: middle;
   content: string(author);
  }
}
```

§ 2.7.1. The string-set property

Name:	***‘string-set’***
Value:	none | [<custom-ident> <string>+]#
Initial:	none
Applies to:	all elements, but not pseudo-elements
Inherited:	no

Percentages:	N/A
Computed value:	the keyword ‘none’ or a list, each item an identifier paired with a list of string values
Canonical order:	per grammar
Animation type:	discrete

User Agents are expected to support this property on all media, including non-visual ones.

The ‘string-set’ property copies the text content of an element into a ‘named string’, which functions as a variable. The text content of this named string can be retrieved using the ‘string()’ function. Since these variables may change on a given page, an optional second value for the ‘string()’ function allows authors to choose which value on a page is used.

‘none’

The element does not set any named strings.

`[ <custom-ident> <string>+ ]#`

The element establishes one or more named strings, corresponding to each comma-separated entry in the list.

For each entry, the <custom-ident> gives the name of the named string. It’s followed by one or more <string> values, which are concatenated together to form the value of the named string.

If an element has style containment, the ‘string-set’ property must have no effects on descendants of that element.

EXAMPLE 17

The following example captures the contents of H1 elements, which represent chapter names in this hypothetical document.

```
H1 { string-set: chapter contents; }
```

When an H1 element is encountered, the ‘chapter’ string is set to the element’s textual contents, and the previous value of ‘chapter’, if any, is overwritten.

§ 2.7.2. The ‘string()’ function

```
string() = string( <custom-ident> , [ first | start | last | first-except ]? )
```

The ‘string()’ function is used to copy the value of a named string to the document, via the ‘content’ property. This function requires one argument, the name of the named string. Since the value of a named string may change several times on a page (as multiple elements defining the string can appear) an optional second argument indicates which value of the named string should be used.

The second argument of the ‘string()’ function is one of the following keywords:

‘first’
The value of the first assignment on the page is used. If there is no assignment on the page, the entry value is used. If no second argument is provided, this is the default value.

‘start’
If the element is the first element on the page, the value of the first assignment is used. Otherwise the entry value is used. The entry value may be empty if the named string hasn’t yet appeared.

‘last’
The exit value of the named string is used.

‘first-except’
This is identical to ‘first’, except that the empty string is used on the page where the value is assigned.

ISSUE 15 we may need to kill the entire content string. Is this necessary?

The content values of named strings are assigned at the point when the content box of the element is first created (or would have been created if the element’s display value is none). The ***entry value*** for a page is the assignment in effect at the end of the previous page. The ***exit value*** for a page is the assignment in effect at the end of the current page.

EXAMPLE 18

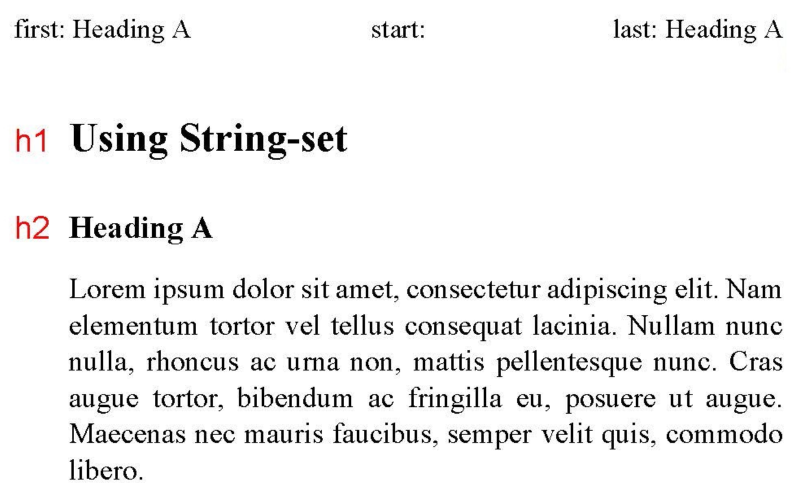
CSS:

```
@page {
  size: 15cm 10cm;
  margin: 1.5cm;

  @top-left {
  content: "first: " string(heading, first);
  }
  @top-center {
  content: "start: " string(heading, start);
  }
   @top-right {
   content: "last: " string(heading, last);
  }
  }

h2 { string-set: heading content() }
```

The following figures show the first, start, and last assignments of the “heading” string on various pages.

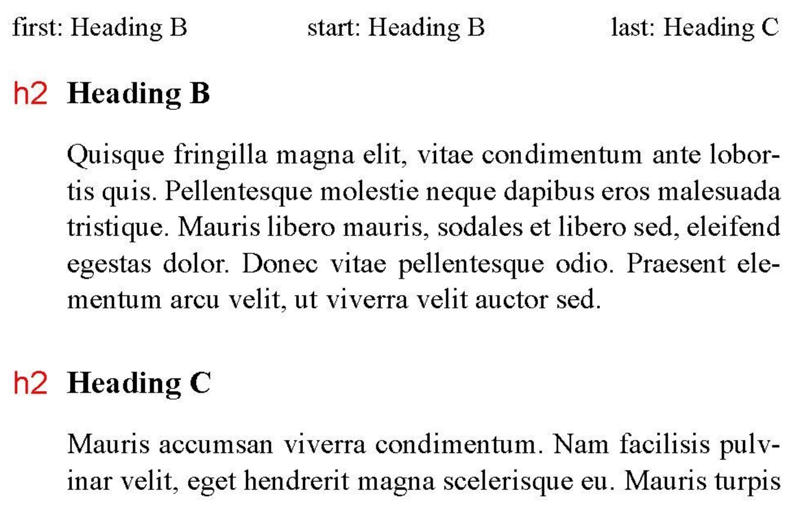

first: Heading A start: last: Heading A

h1 Using String-set

h2 Heading A

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam elementum tortor vel tellus consequat lacinia. Nullam nunc nulla, rhoncus ac urna non, mattis pellentesque nunc. Cras augue tortor, bibendum ac fringilla eu, posuere ut augue. Maecenas nec mauris faucibus, semper velit quis, commodo libero.

Figure 3 *The ‘start’ value is empty, as the string had not yet been set at the start of the page.*



first: Heading B start: Heading B last: Heading C

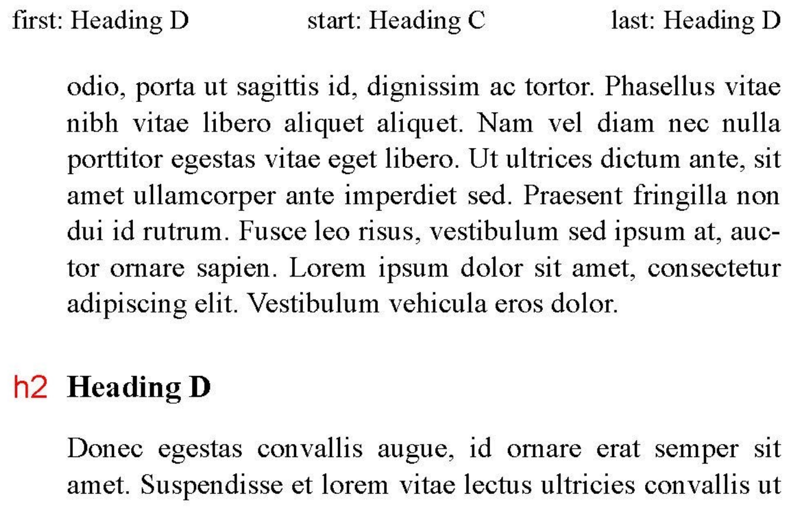
h2 **Heading B**

Quisque fringilla magna elit, vitae condimentum ante lobortis quis. Pellentesque molestie neque dapibus eros malesuada tristique. Mauris libero mauris, sodales et libero sed, eleifend egestas dolor. Donec vitae pellentesque odio. Praesent elementum arcu velit, ut viverra velit auctor sed.

h2 **Heading C**

Mauris accumsan viverra condimentum. Nam facilisis pulvinar velit, eget hendrerit magna scelerisque eu. Mauris turpis

Figure 4 *Since the page starts with an h2, the 'start' value is the value of that head.*



first: Heading D start: Heading C last: Heading D

odio, porta ut sagittis id, dignissim ac tortor. Phasellus vitae nibh vitae libero aliquet aliquet. Nam vel diam nec nulla porttitor egestas vitae eget libero. Ut ultrices dictum ante, sit amet ullamcorper ante imperdiet sed. Praesent fringilla non dui id rutrum. Fusce leo risus, vestibulum sed ipsum at, auctor ornare sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum vehicula eros dolor.

h2 **Heading D**

Donec egestas convallis augue, id ornare erat semper sit amet. Suspendisse et lorem vitae lectus ultricies convallis ut

Figure 5 *Since there's not an h2 at the top of this page, the 'start' value is the exit value of the previous page.*

§ 2.7.3. The 'content()' function

```
content() = content( [ text | before | after | first-letter | marker ]? )
```

'text'
: The string value of the element. If no value is specified in 'content()', it acts as if 'text' were specified.

'before'
: The string value of the '::before' pseudo-element.

'after'
: The string value of the '::after' pseudo-element.

'first-letter'
: The first letter of the element, as defined for the '::first-letter' pseudo-element

'marker'
: The string value of the '::marker' pseudo-element.

EXAMPLE 19

HTML:

```
<h1>Loomings</h1>
```

CSS:

```
h1::before { content: 'Chapter ' counter(chapter); }
h1 { string-set: header content(before) ':' content(text); }
h1::after { content: '.'; }
```

The value of the named string “header” will be “Chapter 1: Loomings”.

EXAMPLE 20

HTML:

```
<section title="Loomings">
```

CSS:

```
section { string-set: header attr(title) }
```

The value of the “header” string will be “Loomings”.

§ 3. Automatic counters and numbering: the ‘counter-increment’ and ‘counter-reset’ properties (moved)

ISSUE 16 Now described in [CSS3LIST]

ISSUE 17 Should this move back to CSS Content?

§ 4. Bookmarks

Some document formats, most notably PDF, allow the use of ***bookmarks*** as an aid to navigation. Bookmarks provide a list of links to document elements, as well as text to label the links and a level value. A bookmark has three properties: ‘bookmark-level’, ‘bookmark-label’, and ‘bookmark-state’.

When a user activates a bookmark, the user agent must bring that reference point to the user’s attention, exactly as if navigating to that element by fragment URL. This will also trigger matching the ‘:target’ pseudo-class.

If an element has style containment, the ‘bookmark-level’, ‘bookmark-label’, and ‘bookmark-state’ properties must have no effect on descendants of the element.

§ 4.1. bookmark-level

The ‘bookmark-level’ property determines if a bookmark is created, and at what level. If this property is absent, or has value ‘none’, no bookmark should be generated, regardless of the values of ‘bookmark-label’ or ‘bookmark-state’.

Name:	***‘bookmark-level’***
Value:	none | <integer>
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	the keyword ‘none’ or the specified integer
Canonical order:	per grammar
Animation type:	by computed value type

‘<integer>’
defines the level of the bookmark, with the top level being 1 (negative and zero values are invalid).

‘none’
no bookmark is generated.

EXAMPLE 21

```
section h1 { bookmark-level: 1; }
section section h1 { bookmark-level: 2; }
section section section h1 { bookmark-level: 3; }
```

Note: Bookmarks do not need to create a strict hierarchy of levels.

ISSUE 18 Should a bookmark be created for elements with `display: none`?

§ 4.2. bookmark-label

Name:	***'bookmark-label'***
Value:	<content-list>
Initial:	content(text)
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Computed value:	specified value
Canonical order:	per grammar
Animation type:	discrete

'<content-list>'

<content-list> is defined above, in the section on the 'string-set' property. The value of <content-list> becomes the text content of the bookmark label.

EXAMPLE 22
HTML:

```
<h1>Loomings</h1>
```

CSS:

```
h1 {
bookmark-label: content(text);
bookmark-level: 1;
}
```

The bookmark label will be “Loomings”.

§ 4.3. bookmark-state

The ‘bookmark-state’ may be open or closed. The user must be able to toggle the bookmark state.

Name:	***‘bookmark-state’***
Value:	open | closed
Initial:	open
Applies to:	block-level elements
Inherited:	no
Percentages:	N/A
Computed value:	specified keyword
Canonical order:	per grammar
Animation type:	discrete

'open'

Subsequent bookmarks with 'bookmark-level' greater than the given bookmark are displayed, until reaching another bookmark of the same level or lower. If one of subsequent bookmark is closed, apply the same test to determine if its subsequent bookmarks should be displayed.

'closed'

Subsequent bookmarks of bookmark-level greater than the given bookmark are not displayed, until reaching another bookmark of the same level or lower.

ISSUE 19 Is the initial bookmark state, or the bookmark state updated by the UA as appropriate?

§ 5. Changes since the 2 June 2016 Working Draft

Significant changes since the 2 June 2016 Working Draft consist primarily of:

- Adding 'auto' as the initial value of 'quotes'.
- Lots of miscellaneous spec clean up: errors, cross-references, overly-loose or sloppy definitions, etc.

See also previous changes.

§ Acknowledgements

Stuart Ballard, David Baron, Bert Bos, Tantek Çelik, and James Craig provided invaluable suggestions used in this specification.

§ 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 23
>
> 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 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-cascade-5] defines the following terms:

- used value

[css-contain-2] defines the following terms:

- style containment

[css-counter-styles-3] defines the following terms:

- <counter-style>

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

- display

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

- <image>
- natural dimension

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

- invalid image

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

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

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

- content language
- white space

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

- <integer>
- <string>
- <url>

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

- #
- +
- ,
- <custom-ident>
- ?
- css-wide keywords
- |

[CSS3LIST] defines the following terms:

- <counter>
- counter-increment
- counter-reset

[HTML] defines the following terms:

- img

[selectors-4] defines the following terms:

- :target
- originating element

§ References

§ Normative References

[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-CONTAIN-2]
Tab Atkins Jr.; Florian Rivoal; Vladimir Levin. CSS Containment Module Level 2. 16 December 2020. WD. URL: https://www.w3.org/TR/css-contain-2/

[CSS-COUNTER-STYLES-3]
Tab Atkins Jr.. CSS Counter Styles Level 3. 14 December 2017. CR. URL: https://www.w3.org/TR/css-counter-styles-3/

[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-IMAGES-4]
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/

[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-TEXT-3]
Elika Etemad; Koji Ishii; Florian Rivoal. CSS Text Module Level 3. 22 December 2020. CR. URL: https://www.w3.org/TR/css-text-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/

[CSS2]
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/

[CSS3-SPEECH]
Daniel Weck. CSS Speech Module. 10 March 2020. CR. URL: https://www.w3.org/TR/css-speech-1/

[CSS3LIST]
Elika Etemad; Tab Atkins Jr.. CSS Lists and Counters Module Level 3. 17 November 2020. WD. URL: https://www.w3.org/TR/css-lists-3/

[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/

§ Informative References

[CLDR]
Unicode Common Locale Data Repository. URL: http://cldr.unicode.org/

[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/

§ Property Index

Name	Value	Initial	Applies to	Inh.	%ages	Animation type	Canonical order	Computed value
'bookmark-label'	<content-list>	content(text)	all elements	no	N/A	discrete	per grammar	specified value
'bookmark-level'	none | <integer>	none	all elements	no	N/A	by computed value type	per grammar	the keyword none or the specified integer
'bookmark-state'	open | closed	open	block-level elements	no	N/A	discrete	per grammar	specified keyword
'content'	normal | none | [<content-replacement> | <content-list>] [/ [<string> | <counter>]+]?	normal	all elements, tree-abiding pseudo-elements, and page margin boxes	no	n/a	discrete	per grammar	See prose below
'quotes'	auto | none | [<string> <string>]+	auto	all elements	yes	n/a	discrete	per grammar	the keyword none, the keyword auto, or a list, each item a pair of string values

Name	Value	Initial	Applies to	Inh.	%ages	Animation type	Canonical order	Computed value
‘string-set’	none | [<custom-ident> <string>+]#	none	all elements, but not pseudo-elements	no	N/A	discrete	per grammar	the keyword none or a list, each item an identifier paired with a list of string values

§ Issues Index

ISSUE 1 The above invalid image behavior appears to be what Chrome is doing. Is this okay? Is there a better behavior we can/should use? ↵

ISSUE 2 Should the contents keyword be replaced with ‘content()’? ↵

ISSUE 3 Start work on an AAM for CSS. ↵

ISSUE 4 CSS2.1 explicitly allowed the UA to substitute a broken image icon if the image was invalid. However, no browser appears to do this. Is this removal okay? ↵

ISSUE 5 Should this behave as an empty string on pseudo-elements? ↵

ISSUE 6 Use cases for suppressing the content on the element and using it in a pseudo-element would be welcome. ↵

ISSUE 7 Do we need the statement about marker pseudo-elements here? Or is this legacy from the old version of the spec? ↵

ISSUE 8 Do leaders depend on the assumption that the content after the leader is right-aligned (end-aligned)? ↵

ISSUE 9 what to do if *after content* is wider than the line box? ↵

ISSUE 10 Leaders don’t quite work in table layouts. How can we fix this? ↵

ISSUE 11 what should error handling be? ↵

ISSUE 12 restrict syntactically to local references for now. ↵

ISSUE 13 found a compelling example, in CSS specs. Do something. ↵

ISSUE 14 A simpler syntax has been proposed by fantasai: http://lists.w3.org/Archives/Public/www-style/2012Feb/0745.html ↵

ISSUE 15 we may need to kill the entire content string. Is this necessary? ↵

ISSUE 16 Now described in [CSS3LIST] ↵

ISSUE 17 Should this move back to CSS Content? ↵

ISSUE 18 Should a bookmark be created for elements with `display: none`? ↵

ISSUE 19 Is the initial bookmark state, or the bookmark state updated by the UA as appropriate? ↵