CSS Counter Styles Level 3

W3C Candidate Recommendation Snapshot, 27 July 2021

This version:
https://www.w3.org/TR/2021/CR-css-counter-styles-3-20210727/

Latest published version:
https://www.w3.org/TR/css-counter-styles-3/

Editor's Draft:
https://drafts.csswg.org/css-counter-styles/

Previous Versions:
https://www.w3.org/TR/2017/CR-css-counter-styles-3-20171214/
https://www.w3.org/TR/2015/CR-css-counter-styles-3-20150611/
https://www.w3.org/TR/2014/WD-css-counter-styles-3-20140826/
https://www.w3.org/TR/2013/WD-css-counter-styles-3-20130718/
https://www.w3.org/TR/2013/WD-css-counter-styles-3-20130221/
https://www.w3.org/TR/2012/WD-css-counter-styles-3-20121009/

Implementation Report:
https://test.csswg.org/harness/results/css-counter-styles-3_dev/grouped/

Test Suite:
http://test.csswg.org/suites/css-counter-styles-3_dev/nightly-unstable

Issue Tracking:
CSSWG Issues Repository

Editor:
Tab Atkins Jr. (Google)

Suggest an Edit for this Spec:
GitHub Editor

Abstract

This module introduces the '@counter-style' rule, which allows authors to define their own custom counter styles for use with CSS list-marker and generated-content counters [CSS-LISTS-3]. It also predefines a set of common counter styles, including the ones present in CSS2 and CSS2.1.

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 **Candidate Recommendation Snapshot**. Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. A Candidate Recommendation Snapshot has received wide review and is intended to gather implementation experience. This document is intended to become a W3C Recommendation; it will remain a Candidate Recommendation at least until 27 October 2021 to gather additional feedback.

Please send feedback by filing issues in GitHub (preferred), including the spec code "css-counter-styles" in the title, like this: "[css-counter-styles] *...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.

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

- the <image> value in <symbol>

"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

IDL Index

§ 1. Introduction

CSS 1 defined a handful of useful counter styles based on the styles that HTML traditionally allowed on ordered and unordered lists. While this was expanded slightly by CSS2.1, it doesn't address the needs of worldwide typography.

This module introduces the '@counter-style' rule which allows CSS to address this in an open-ended manner, by allowing the author to define their own counter styles. These styles can then be used in the 'list-style-type' property or in the 'counter()' and 'counters()' functions. It also defines some additional predefined counter styles, particularly ones which are common but complicated to represent with '@counter-style'.

§ 2. Counter Styles

A ***counter style*** defines how to convert a counter value into a string. Counter styles are composed of:

- a name, to identify the style
- an algorithm, which transforms integer counter values into a basic string representation
- a negative sign, which is prepended or appended to the representation of a negative counter value.
- a prefix, to prepend to the representation
- a suffix to append to the representation
- a range, which limits the values that a counter style handles
- a spoken form, which describes how to read out the counter style in a speech synthesizer
- and a fallback style, to render the representation with when the counter value is outside the counter style's range or the counter style otherwise can't render the counter value

When asked to ***generate a counter representation*** using a particular counter style for a particular counter value, follow these steps:

1. If the counter style is unknown, exit this algorithm and instead generate a counter representation using the 'decimal' style and the same counter value.
2. If the counter value is outside the 'range' of the counter style, exit this algorithm and instead generate a counter representation using the counter style's fallback style and the same counter value.

3. Using the counter value and the counter algorithm for the counter style, generate an ***initial representation for the counter value***. If the counter value is negative and the counter style uses a negative sign, instead generate an initial representation using the absolute value of the counter value.
4. Prepend symbols to the representation as specified in the 'pad' descriptor.
5. If the counter value is negative and the counter style uses a negative sign, wrap the representation in the counter style's negative sign as specified in the 'negative' descriptor.
6. Return the representation.

> Note: 'prefix' and 'suffix' don't play a part in this algorithm. This is intentional; the prefix and suffix aren't part of the string returned by the counter() or counters() functions. Instead, the prefix and suffix are added by the algorithm that constructs the value of the 'content' property for the '::marker' pseudo-element. This also implies that the prefix and suffix always come from the specified counter-style, even if the actual representation is constructed by a fallback style.

Some values of 'system' ('symbolic', 'additive') and some descriptors ('pad') can generate representations with size linear to an author-supplied number. This can potentially be abused to generate excessively large representations and consume undue amounts of the user's memory or even hang their browser. User agents must support representations at least 60 Unicode codepoints long, but they may choose to instead use the fallback style for representations that would be longer than 60 codepoints.

§ 3. Defining Custom Counter Styles: the '@counter-style' rule

The ***'@counter-style'*** rule allows authors to define a custom counter style. The components of a counter style are specified by descriptors in the '@counter-style' rule. The algorithm is specified implicitly by a combination of the 'system', 'symbols', and 'additive-symbols' properties.

The general form of an '@counter-style' rule is:

```
@counter-style <counter-style-name> { <declaration-list> }
```

'<counter-style-name>' is a <custom-ident> that is not an ASCII case-insensitive match for 'none'. The <counter-style-name> is a tree-scoped name.

The keywords 'decimal', 'disc', 'square', 'circle', 'disclosure-open', and 'disclosure-closed' are valid <counter-style-name>s, but are invalid when used here to name a counter style rule; doing so makes the rule invalid. (They can be used in other contexts, such as in the 'extend' system.)

> Note: Note that <custom-ident> also automatically excludes the CSS-wide keywords. In addition, some names, like 'inside', are valid as counter style names, but conflict with the existing values of properties like 'list-style', and so won't be usable there.

Counter style names are case-sensitive. However, the names defined in this specification are ASCII lower-cased on parse wherever they are used as counter styles, e.g. in the 'list-style' set of properties, in the '@counter-style' rule, and in the 'counter()' functions.

Each '@counter-style' rule specifies a value for every counter-style descriptor, either implicitly or explicitly. Those not given explicit value in the rule take the initial value listed with each descriptor in this specification. These descriptors apply solely within the context of the '@counter-style' rule in which they are defined, and do not apply to document language elements. There is no notion of which elements the descriptors apply to or whether the values are inherited by child elements. When a given descriptor occurs multiple times in a given '@counter-style' rule, only the last-specified valid value is used; all prior values for that descriptor must be ignored.

Defining a '@counter-style' makes it available to the entire document in which it is included. If multiple '@counter-style' rules are defined with the same name, only one wins, according to standard cascade rules. '@counter-style' rules cascade "atomically": if one replaces another of the same name, it replaces it *entirely*, rather than just replacing the specific descriptors it specifies.

> Note: Note that even the predefined counter styles can be overridden; the UA stylesheet occurs before any other stylesheets, so the predefined ones always lose in the cascade.

This at-rule conforms with the forward-compatible parsing requirement of CSS; conformant parsers that don't understand these rules will ignore them without error. Any descriptors that are not recognized or implemented by a given user agent, or whose value does not match the grammars given here or in a future version of this specification, must be ignored in their entirety; they do not make the '@counter-style' rule invalid.

§ 3.1. Counter algorithms: the 'system' descriptor

Name:	***'system'***
For:	'@counter-style'
Value:	cyclic | numeric | alphabetic | symbolic | additive | [fixed <integer>?] | [extends <counter-style-name>]
Initial:	symbolic

The 'system' descriptor specifies which algorithm will be used to construct the counter's representation based on the counter value. For example, 'cyclic' counter styles just cycle through their symbols repeatedly, while 'numeric' counter styles interpret their symbols as digits and build their representation accordingly. The systems are defined in the following subsections.

Each 'system' value is associated with either the 'symbols' or 'additive-symbols' descriptors, and has a minimum length that the appropriate descriptor must have; each entry below defines what this is. If a '@counter-style' rule fails to meet this requirement, it does not define a counter style. (The rule is still syntactically valid, but has no effect.)

§ 3.1.1. Cycling Symbols: the 'cyclic' system

The ***'cyclic'*** counter system cycles repeatedly through its provided symbols, looping back to the beginning when it reaches the end of the list. It can be used for simple bullets (just provide a single counter symbol), or for cycling through multiple symbols. The first counter symbol is used as the representation of the value 1, the second counter symbol (if it exists) is used as the representation of the value 2, etc.

If the system is 'cyclic', the 'symbols' descriptor must contain at least one counter symbol. This system is defined over all counter values.

EXAMPLE 1

A "triangle bullet" counter style can be defined as:

```
@counter-style triangle {
  system: cyclic;
  symbols: ‣;
  suffix: " ";
}
```

It will then produce lists that look like:

- One
- Two
- Three

If there are *N* counter symbols and a representation is being constructed for the integer *value*, the representation is the counter symbol at index ((*value*-1) mod *N*) of the list of counter symbols (0-indexed).

§ 3.1.2. Exhaustible Symbols: the 'fixed' system

The ***'fixed'*** counter system runs through its list of counter symbols once, then falls back. It is useful for representing counter styles that only have a finite number of representations. For example, Unicode defines several limited-length runs of special characters meant for lists, such as circled digits.

If the system is 'fixed', the 'symbols' descriptor must contain at least one counter symbol. This system is defined over counter values in a finite range, starting with the first symbol value and having a length equal to the length of the list of counter symbols.

When this system is specified, it may optionally have an integer provided after it, which sets the ***first symbol value***. If it is omitted, the first symbol value is 1.

EXAMPLE 2

A "box-corner" counter style can be defined as:

```
@counter-style box-corner {
  system: fixed;
  symbols: ◰ ◳ ◲ ◱;
  suffix: ': ';
}
```

It will then produce lists that look like:

```
◰:  One
◳:  Two
◲:  Three
◱:  Four
5:  Five
6:  Six
```

The first counter symbol is the representation for the first symbol value, and subsequent counter values are represented by subsequent counter symbols. Once the list of counter symbols is exhausted, further values cannot be represented by this counter style, and must instead be represented by the fallback counter style.

§ 3.1.3. Repeating Symbols: the 'symbolic' system

The ***'symbolic'*** counter system cycles repeatedly through its provided symbols, doubling, tripling, etc. the symbols on each successive pass through the list. For example, if the original symbols were "*" and "†", then on the second pass they would instead be "**" and "††", while on the third they would be "***"and "†††", etc. It can be used for footnote-style markers, and is also sometimes used for alphabetic-style lists for a slightly different presentation than what the 'alphabetic' system presents.

If the system is 'symbolic', the 'symbols' descriptor must contain at least one counter symbol. This system is defined only over strictly positive counter values.

EXAMPLE 3

An "footnote" counter style can be defined as:

```
@counter-style footnote {
  system: symbolic;
  symbols: '*' ⁑ † ‡;
  suffix: " ";
}
```

It will then produce lists that look like:

```
*   One
⁑   Two
†   Three
‡   Four
**  Five
⁑⁑  Six
```

EXAMPLE 4

Some style guides mandate a list numbering that looks similar to ‘upper-alpha’, but repeats differently after the first 26 values, instead going "AA", "BB", "CC", etc. This can be achieved with the symbolic system:

```
@counter-style upper-alpha-legal {
  system: symbolic;
  symbols: A B C D E F G H I J K L M
           N O P Q R S T U V W X Y Z;
}
```

This style is identical to ‘upper-alpha’ through the first 27 values, but they diverge after that, with ‘upper-alpha’ going "AB", "AC", "AD", etc. Starting at the 53rd value, ‘upper-alpha’ goes "BA", "BB", "BC", etc., while this style jumps into triple digits with "AAA", "BBB", "CCC", etc.

To construct the representation, run the following algorithm:

Let *N* be the length of the list of counter symbols, *value* initially be the counter value, *S* initially be the empty string, and *symbol(n)* be the nth counter symbol in the list of counter symbols (0-indexed).

1. Let the *chosen symbol* be `symbol( (value - 1) mod N)`.
2. Let the *representation length* be `ceil( value / N )`.

3. Append the *chosen symbol* to *S* a number of times equal to the *representation length*.

Finally, return *S*.

§ 3.1.4. Bijective Numerals: the 'alphabetic' system

The ***'alphabetic'*** counter system interprets the list of counter symbols as digits to an *alphabetic* numbering system, similar to the default 'lower-alpha' counter style, which wraps from "a", "b", "c", to "aa", "ab", "ac". Alphabetic numbering systems do not contain a digit representing 0; so the first value when a new digit is added is composed solely of the first digit. Alphabetic numbering systems are commonly used for lists, and also appear in many spreadsheet programs to number columns. The first counter symbol in the list is interpreted as the digit 1, the second as the digit 2, and so on.

If the system is 'alphabetic', the 'symbols' descriptor must contain at least two counter symbols. This system is defined only over strictly positive counter values.

EXAMPLE 5

A counter style using go stones can be defined as:

```
@counter-style go {
  system: alphabetic;
  symbols: url(white.svg) url(black.svg);
  suffix: " ";
}
```

It will then produce lists that look like:

 One
 Two
 Three
 Four
 Five
 Six
 Seven

Note: This example requires support for SVG images to display correctly.

If there are *N* counter symbols, the representation is a base *N* alphabetic number using the counter symbols as digits. To construct the representation, run the following algorithm:

Let *N* be the length of the list of counter symbols, *value* initially be the counter value, *S* initially be the empty string, and *symbol(n)* be the nth counter symbol in the list of counter symbols (0-indexed).

While *value* is not equal to 0:

1. Set *value* to `value - 1`.
2. Prepend *symbol(value mod N)* to *S*.
3. Set *value* to `floor( value / N )`.

Finally, return *S*.

§ 3.1.5. Positional Numerals: the 'numeric' system

The ***'numeric'*** counter system interprets the list of counter symbols as digits to a "place-value" numbering system, similar to the default 'decimal' counter style. The first counter symbol in the list is interpreted as the digit 0, the second as the digit 1, and so on.

If the system is 'numeric', the 'symbols' descriptor must contain at least two counter symbols. This system is defined over all counter values.

> EXAMPLE 6
> A "trinary" counter style can be defined as:
>
> ```
> @counter-style trinary {
>   system: numeric;
>   symbols: '0' '1' '2';
> }
> ```
>
> It will then produce lists that look like:
>
> ```
> 1.  One
> 2.  Two
> 10. Three
> 11. Four
> 12. Five
> 20. Six
> ```

If there are *N* counter symbols, the representation is a base *N* number using the counter symbols as digits. To construct the representation, run the following algorithm:

Let *N* be the length of the list of counter symbols, *value* initially be the counter value, *S* initially be the empty string, and *symbol(n)* be the nth counter symbol in the list of counter symbols (0-indexed).

1. If *value* is 0, append `symbol(0)` to *S* and return *S*.
2. While *value* is not equal to 0:
    1. Prepend *symbol(value mod N)* to *S*.
    2. Set *value* to `floor( value / N )`.
3. Return *S*.

§ 3.1.6. Accumulating Numerals: the 'additive' system

The ***'additive'*** counter system is used to represent "sign-value" numbering systems, which, rather than reusing digits in different positions to change their value, define additional digits with much larger values, so that the value of the number can be obtained by adding all the digits together. This is used in Roman numerals and other numbering systems around the world.

If the system is 'additive', the 'additive-symbols' descriptor must contain at least one additive tuple. This system is nominally defined over all counter values (see algorithm, below, for exact details).

> EXAMPLE 7
> A "dice" counter style can be defined as:
>
> ```
> @counter-style dice {
>   system: additive;
>   additive-symbols: 6 ⚅, 5 ⚄, 4 ⚃, 3 ⚂, 2 ⚁, 1 ⚀;
>   suffix: " ";
> }
> ```
>
> It will then produce lists that look like:
>
> ```
>   ⚀ One
>   ⚁ Two
>   ⚂ Three
> ...
>  ⚅⚄ Eleven
>  ⚅⚅ Twelve
> ⚅⚅⚀ Thirteen
> ```

To construct the representation:

1. Let *value* initially be the counter value, *S* initially be the empty string, and *symbol list* initially be the list of additive tuples.
2. If *value* is zero:

    1. If *symbol list* contains a tuple with a weight of zero, append that tuple's counter symbol to *S* and return *S*.
    2. Otherwise, the given counter value cannot be represented by this counter style, and must instead be represented by the fallback counter style.

3. For each *tuple* in *symbol list*:

    1. Let *symbol* and *weight* be *tuple*'s counter symbol and weight, respectively.
    2. If *weight* is zero, or *weight* is greater than *value*, continue.
    3. Let *reps* be `floor( value / weight )`.
    4. Append *symbol* to *S* *reps* times.
    5. Decrement *value* by `weight * reps`.
    6. If *value* is zero, return *S*.

4. Assertion: *value* is still non-zero.

    The given counter value cannot be represented by this counter style, and must instead be represented by the fallback counter style.

> Note: All of the predefined additive '@counter-style' rules in this specification produce representations for every value in their range, but it's possible to produce values for 'additive-symbols' that will fail to find a representation with the algorithm defined above, even though theoretically a representation could be found. For example, if a '@counter-style' was defined with 'additive-symbols: 3 "a", 2 "b";', the algorithm defined above will fail to find a representation for a counter value of 4, even though theoretically a "bb" representation would work. While unfortunate, this is required to maintain the property that the algorithm runs in linear time relative to the size of the counter value.

§ 3.1.7. Building from Existing Counter Styles: the 'extends' system

The ***'extends'*** system allows an author to use the algorithm of another counter style, but alter other aspects, such as the negative sign or the suffix. If a counter style uses the 'extends' system, any unspecified descriptors must be taken from the extended counter style specified, rather than taking their initial values.

If a ‘@counter-style’ uses the ‘extends’ system, it must not contain a ‘symbols’ or ‘additive-symbols’ descriptor, or else the ‘@counter-style’ rule is invalid.

If the specified <counter-style-name> is an ASCII case-insensitive match for ‘disc’, ‘circle’, ‘square’, ‘disclosure-open’, or ‘disclosure-closed’ (any of the predefined symbolic counter styles), using ‘extend’ extends from the “standard” definition of the rules provided in the normative stylesheet (rather than the exception allowing them to be drawn in a different, user-agent-specific fashion.)

If the specified counter style name isn’t the name of any defined counter style, it must be treated as if it was extending the ‘decimal’ counter style. If one or more ‘@counter-style’ rules form a cycle with their ‘extends’ values, all of the counter styles participating in the cycle must be treated as if they were extending the ‘decimal’ counter style instead.

EXAMPLE 8

For example, if you wanted a counter style that was identical to decimal, but used a parenthesis rather than a period after it, like:

```
1) first item
2) second item
3) third item
```

Rather than writing up an entirely new counter style, this can be done by just extending ‘decimal’:

```
@counter-style decimal-paren {
  system: extends decimal;
  suffix: ") ";
}
```

§ 3.2. Formatting negative values: the ‘negative’ descriptor

Name:	***‘negative’***
For:	‘@counter-style’
Value:	<symbol> <symbol>?
Initial:	"\2D" ("-" hyphen-minus)

The ‘negative’ descriptor defines how to alter the representation when the counter value is negative.

The first <symbol> in the value is prepended to the representation when the counter value is negative. The second <symbol>, if specified, is appended to the representation when the counter value is negative.

EXAMPLE 9

For example, specifying ‘negative: "(" ")";’ will make negative values be wrapped in parentheses, which is sometimes used in financial contexts, like "(2) (1) 0 1 2 3...".

Not all ‘system’ values use a negative sign. In particular, a counter style ***uses a negative sign*** if its ‘system’ value is ‘symbolic’, ‘alphabetic’, ‘numeric’, ‘additive’, or ‘extends’ if the extended counter style itself uses a negative sign. If a counter style does not use a negative sign, it ignores the negative sign when generating a counter representation.

§ 3.3. Symbols before the marker: the ‘prefix’ descriptor

Name:	***‘prefix’***
For:	‘@counter-style’
Value:	<symbol>
Initial:	"" (the empty string)

The ‘prefix’ descriptor specifies a <symbol> that is prepended to the marker representation. Prefixes come before any negative sign.

Note: Prefixes are only added by the algorithm for constructing the default contents of the ‘::marker’ pseudo-element; the prefix is not added automatically when the ‘counter()’ or ‘counters()’ functions are used.

§ 3.4. Symbols after the marker: the ‘suffix’ descriptor

Name:	***'suffix'***
For:	'@counter-style'
Value:	<symbol>
Initial:	"\2E\20" ("." full stop followed by a space)

The 'suffix' descriptor specifies a <symbol> that is appended to the marker representation. Suffixes are added to the representation after negative signs.

Note: Suffixes are only added by the algorithm for constructing the default contents of the '::marker' pseudo-element; the suffix is not added automatically when the counter() or counters() functions are used.

§ 3.5. Limiting the counter scope: the 'range' descriptor

Name:	***'range'***
For:	'@counter-style'
Value:	[[<integer>	infinite]{2}]#	auto
Initial:	auto

The 'range' descriptor defines the ranges over which the counter style is defined. If a counter style is used to represent a counter value outside of its ranges, the counter style instead drops down to its fallback counter style.

'auto'

The range depends on the counter system:

- For 'cyclic', 'numeric', and 'fixed' systems, the range is negative infinity to positive infinity.
- For 'alphabetic' and 'symbolic' systems, the range is 1 to positive infinity.
- For 'additive' systems, the range is 0 to positive infinity.
- For 'extends' systems, the range is whatever 'auto' would produce for the extended system; if extending a complex predefined style (§ 7 Complex Predefined Counter Styles), the range

is the style's defined range.

[[<integer> | infinite]{2}]#

This defines a comma-separated list of ranges. For each individual range, the first value is the lower bound and the second value is the upper bound. This range is inclusive - it contains both the lower and upper bound numbers. If 'infinite' is used as the first value in a range, it represents negative infinity; if used as the second value, it represents positive infinity. The range of the counter style is the union of all the ranges defined in the list.

If the lower bound of any range is higher than the upper bound, the entire descriptor is invalid and must be ignored.

Implementations must support ranges with a lower bound of at least -2^{15} and an upper bound of at least $2^{15}-1$ (the range of a signed 2-byte int). They may support higher ranges. If any specified bound is outside of the implementation's supported bounds, it must be treated as the closest bound that the implementation does support.

§ 3.6. Zero-Padding and Constant-Width Representations: the 'pad' descriptor

Name:	***'pad'***
For:	'@counter-style'
Value:	<integer [0,∞]> && <symbol>
Initial:	0 ""

The 'pad' descriptor allows an author to specify a "fixed-width" counter style, where representations shorter than the pad value are padded with a particular <symbol>. Representations larger than the specified pad value are constructed as normal.

<integer [0,∞]> && <symbol>

The <integer> specifies a minimum length that all counter representations must reach.

Let *difference* be the provided <integer> minus the number of grapheme clusters in the initial representation for the counter value. (Note that, per the algorithm to generate a counter representation, this occurs before adding prefixes/suffixes/negatives.) If the counter value is negative and the counter style uses a negative sign, further reduce *difference* by the number of grapheme clusters in the counter style's 'negative' descriptor's <symbol>(s).

If *difference* is greater than zero, prepend *difference* copies of the specified <symbol> to the representation.

Negative <integer> values are not allowed.

EXAMPLE 10

The most common example of "fixed-width" numbering is zero-padded decimal numbering. If an author knows that the numbers used will be less than a thousand, for example, it can be zero-padded with a simple 'pad: 3 "0";' descriptor, ensuring that all of the representations are 3 digits wide.

This will cause, for example, 1 to be represented as "001", 20 to be represented as "020", 300 to be represented as "300", 4000 to be represented as "4000", and -5 to be represented as "-05".

Note: The 'pad' descriptor counts the number of grapheme clusters in the representation, but pads it with <symbol>s. If the specified 'pad' <symbol> is multi-character, this will likely not have the desired effect. Unfortunately, there's no way to use the number of grapheme clusters in the 'pad' <symbol> without violating useful constraints. It is recommended that authors only specify <symbol>s of a single grapheme cluster in the 'pad' descriptor.

§ 3.7. Defining fallback: the 'fallback' descriptor

Name:	***'fallback'***
For:	'@counter-style'
Value:	<counter-style-name>
Initial:	decimal

The 'fallback' descriptor specifies a fallback counter style to be used when the current counter style can't create a representation for a given counter value. For example, if a counter style defined with a range of 1-10 is asked to represent a counter value of 11, the counter value's representation is instead constructed with the fallback counter style (or possibly the fallback style's fallback style, if the fallback style can't represent that value, etc.).

If the value of the ‘fallback’ descriptor isn’t the name of any defined counter style, the used value of the ‘fallback’ descriptor is ‘decimal’ instead. Similarly, while following fallbacks to find a counter style that can render the given counter value, if a loop in the specified fallbacks is detected, the ‘decimal’ style must be used instead.

> Note that it is not necessarily an error to specify fallback loops. For example, if an author desires a counter style with significantly different representations for even and odd counter values, they may find it easiest to define one style that can only represent odd values and one that can only represent even values, and specify each as the fallback for the other one. Though the fallback graph is circular, at no point do you encounter a loop while following these fallbacks - every counter value is represented by one or the other counter style.

§ 3.8. Marker characters: the ‘symbols’ and ‘additive-symbols’ descriptors

Name:	***‘symbols’***
For:	‘@counter-style’
Value:	<symbol>+
Initial:	n/a

Name:	***‘additive-symbols’***
For:	‘@counter-style’
Value:	[<integer [0,∞]> && <symbol>]#
Initial:	n/a

<symbol> = <string> | <image> | <custom-ident>

The ‘symbols’ and ‘additive-symbols’ descriptors specify the symbols used by the marker-construction algorithm specified by the ‘system’ descriptor. The ‘@counter-style’ rule must have a valid ‘symbols’ descriptor if the counter system is ‘cyclic’, ‘numeric’, ‘alphabetic’, ‘symbolic’, or ‘fixed’, or a valid ‘additive-symbols’ descriptor if the counter system is ‘additive’; otherwise, the ‘@counter-style’ does not define a counter style (but is still a valid at-rule).

Some counter systems specify that the 'symbols' descriptor must have at least two entries. If the counter style's system is such, and the 'symbols' descriptor has only a single entry, the '@counter-style' rule does not define a counter style.

Each entry in the 'symbols' descriptor's value defines a ***counter symbol***, which is interpreted differently based on the counter style's system. Each entry in the 'additive-symbols' descriptor's value defines an ***additive tuple***, which consists of a counter symbol and an integer weight. Each weight must be a non-negative integer, and the additive tuples must be specified in order of strictly descending weight; otherwise, the declaration is invalid and must be ignored.

Counter symbols may be strings, images, or identifiers, and the three types can be mixed in a single descriptor. Counter representations are constructed by concatenating counter symbols together. Identifiers are rendered as strings containing the same characters. Images are rendered as inline replaced elements. The default object size of an image counter symbol is a 1em by 1em square.

> Note: The <image> syntax in <symbol> is currently at-risk. No implementations have plans to implement it currently, and it complicates some usages of 'counter()' in ways that haven't been fully handled.

> Note: If using identifiers rather than strings to define the symbols, be aware of the syntax of identifiers. In particular, ascii non-letters like "*" are not identifiers, and so must be quoted in a string. Hex escapes, used in several of the counter styles defined in this specification, "eat" the following space (to allow a digit to follow a hex escape without ambiguity), so two spaces must be put after a hex escape to separate it from the following one, or else they'll be considered adjacent, and part of the same identifier. For example, 'symbols: \660 \661;' only defines a single symbol, consisting of the U+0660 and U+0661 characters, rather than the two that were intended; either quote the escapes in strings, like 'symbols: "\660" "\661"', or put two spaces between the escapes.

§ 3.9. Speech Synthesis: the 'speak-as' descriptor

Name:	***'speak-as'***
For:	'@counter-style'
Value:	auto | bullets | numbers | words | spell-out | <counter-style-name>
Initial:	auto

A counter style can be constructed with a meaning that is obvious visually, but impossible to meaningfully represent via a speech synthesizer or other non-visual means, or possible but nonsensical when naively read out loud. The ‘speak-as’ descriptor describes how to synthesize the spoken form of a counter formatted with the given counter style. Assistive technologies should use this spoken form when reading out the counter style, and may use the ‘speak-as’ value to inform transformations to outputs other than speech. Values have the following meanings:

‘auto’

If the counter style’s ‘system’ is ‘alphabetic’, this value has the same effect as ‘spell-out’. If the ‘system’ is ‘cyclic’, this value has the same effect as ‘bullets’. If the ‘system’ is ‘extends’, this value has the same effect as ‘auto’ would have for the extended style. Otherwise, this value has the same effect as ‘numbers’.

‘bullets’

The UA speaks a UA-defined phrase or audio cue that represents an unordered list item being read out.

‘numbers’

The counter’s numeric value is spoken as a number in the content language.

‘words’

Generate a counter representation for the value as normal, then speak it as normal text in the content language. If the counter representation contains images, instead handle the value as for ‘numbers’.

‘spell-out’

Generate a counter representation for the value as normal, then spell it out letter-by-letter in the content language. If the UA does not know how to pronounce the symbols (or the counter representation contains images), it must instead handle the value as for ‘numbers’.

For example, ‘lower-greek’ in English would be read out as "alpha", "beta", "gamma", etc. Conversely, ‘upper-latin’ in French would be read out as (in phonetic notation) /a/, /be/, /se/, etc.

‘<counter-style-name>’

The counter’s value is instead spoken out in the specified style (similar to the behavior of the ‘fallback’ descriptor when generating representations for a counter value). If the specified style does not exist, this value is treated as ‘auto’. If a loop is detected when following ‘speak-as’ references, this value is treated as ‘auto’ for the counter styles participating in the loop.

EXAMPLE 11

The ability to defer pronunciation to another counter style can help when the symbols being used aren't actually letters. For example, here's a possible definition of a 'circled-lower-latin' counter-style, using some special Unicode characters:

```
@counter-style circled-lower-latin {
  system: alphabetic;
  speak-as: lower-latin;
  symbols: ⓐ ⓑ ⓒ ⓓ ⓔ ⓕ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓜ ⓝ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ
ⓩ;
  suffix: " ";
}
```

Setting its 'system' to 'alphabetic' would normally make the UA try to read out the names of the characters, but in this case that might be something like "Circled Letter A", which is unlikely to make sense. Instead, explicitly setting 'speak-as' to 'lower-latin' ensures that they get read out as their corresponding latin letters, as intended.

§ 4. Defining Anonymous Counter Styles: the 'symbols()' function

The 'symbols()' function allows a counter style to be defined inline in a property value, for when a style is used only once in a stylesheet and defining a full '@counter-style' rule would be overkill. It does not provide the full feature-set of the '@counter-style' rule, but provides a sufficient subset to still be useful. The syntax of the 'symbols()' rule is:

```
symbols() = symbols( <symbols-type>? [ <string> | <image> ]+ )
<symbols-type> = cyclic | numeric | alphabetic | symbolic | fixed
```

The 'symbols()' function defines an anonymous counter style with no name, a 'prefix' of '""' (empty string) and 'suffix' of '" "' (U+0020 SPACE), a 'range' of 'auto', a 'fallback' of 'decimal', a 'negative' of '"\2D"' ("-" hyphen-minus), a 'pad' of '0 ""', and a 'speak-as' of 'auto'. The counter style's algorithm is constructed by consulting the previous chapter using the provided system — or 'symbolic' if the system was omitted — and the provided <string>s and <image>s as the value of the 'symbols' property. If the system is 'fixed', the first symbol value is '1'.

If the system is 'alphabetic' or 'numeric', there must be at least two <string>s or <image>s, or else the function is invalid.

EXAMPLE 12

This code:

```
ol { list-style: symbols("*" "\2020" "\2021" "\A7"); }
```

will produce lists that look like:

```
*   One
†   Two
‡   Three
§   Four
**  Five
††  Six
‡‡  Seven
```

On the other hand, specifying the system of counter, like so:

```
ol { list-style: symbols(cyclic "*" "\2020" "\2021" "\A7"); }
```

will produce lists that look like:

```
*   One
†   Two
‡   Three
§   Four
*   Five
†   Six
‡   Seven
```

Note: the ‘symbols()’ function only allows strings and images, while the ‘symbols’ descriptor of a ‘@counter-style’ rule also allows identifiers.

§ 5. Extending ‘list-style-type’, ‘counter()’, and ‘counters()’

In CSS Level 2 [CSS21] the ‘list-style-type’ property and the ‘counter()’ and ‘counters()’ notations accept various pre-defined keywords, each identifying a counter style. This module extends these features to take instead the <counter-style> type, defined below:

<counter-style> = <counter-style-name> | <symbols()>

If a <counter-style-name> is used that does not refer to any existing counter style, it must act identically to the 'decimal' counter style (but does not *compute* to 'decimal').

When used in these contexts, a <counter-style-name> is a tree-scoped reference.

§ 6. Simple Predefined Counter Styles

The following stylesheet uses the '@counter-style' rule to redefine all of the counter styles defined in CSS 2 and CSS 2.1. This stylesheet is normative—UAs must include it in their UA stylesheet (or at least act as if these rules were defined at that level).

§ 6.1. Numeric: 'decimal', 'decimal-leading-zero', 'arabic-indic', 'armenian', 'upper-armenian', 'lower-armenian', 'bengali', 'cambodian', 'khmer', 'cjk-decimal', 'devanagari', 'georgian', 'gujarati', 'gurmukhi', 'hebrew', 'kannada', 'lao', 'malayalam', 'mongolian', 'myanmar', 'oriya', 'persian', 'lower-roman', 'upper-roman', 'tamil', 'telugu', 'thai', 'tibetan'

'decimal'
: Western decimal numbers (e.g., 1, 2, 3, ..., 98, 99, 100).

'decimal-leading-zero'
: Decimal numbers padded by initial zeros (e.g., 01, 02, 03, ..., 98, 99, 100).

'arabic-indic'
: Arabic-indic numbering (e.g., ١, ٢, ٣, ٤, ..., ٩٨, ٩٩, ١٠٠).

'armenian'
'upper-armenian'
: Traditional uppercase Armenian numbering (e.g., Ա, Բ, Գ, ..., ՂՀ, ՂՁ, Ճ).

'lower-armenian'
: Lowercase Armenian numbering (e.g., ա, բ, գ, ..., ղհ, ղձ, ճ).

'bengali'
: Bengali numbering (e.g., ১, ২, ৩, ..., ৯৮, ৯৯, ১০০).

'cambodian'
'khmer'
: Cambodian/Khmer numbering (e.g., ១, ២, ៣, ..., ៩៨, ៩៩, ១០០).

'cjk-decimal'
: Han decimal numbers (e.g., 一, 二, 三, ..., 九八, 九九, 一〇〇).

'devanagari'
: devanagari numbering (e.g., १, २, ३, ..., ९८, ९९, १००).

'georgian'
: Traditional Georgian numbering (e.g., ა, ბ, გ, ..., ჟჵ, ჟთ, რ).

'gujarati'
: Gujarati numbering (e.g., ૧, ૨, ૩, ..., ૯૮, ૯૯, ૧૦૦).

'gurmukhi'
: Gurmukhi numbering (e.g., ੧, ੨, ੩, ..., ੯੮, ੯੯, ੧੦੦).

'hebrew'
: Traditional Hebrew numbering (e.g., א, ב, ג, ..., צח, צט, ק).

'kannada'
: Kannada numbering (e.g., ೧, ೨, ೩, ..., ೯೮, ೯೯, ೧೦೦).

'lao'
: Laotian numbering (e.g., ๑, ๒, ๓, ..., ๙๘, ๙๙, ๑๐๐).

'malayalam'
: Malayalam numbering (e.g., ൧, ൨, ൩, ..., ൹൱, ൹൹, ൧൦൦).

'mongolian'
: Mongolian numbering (e.g., ᠑, ᠒, ᠓, ..., ᠙᠘, ᠙᠙, ᠑᠐᠐).

'myanmar'
: Myanmar (Burmese) numbering (e.g., ၁, ၂, ၃, ..., ၉၈, ၉၉, ၁၀၀).

'oriya'
: Oriya numbering (e.g., ୧, ୨, ୩, ..., ୯୮, ୯୯, ୧୦୦).

'persian'
: Persian numbering (e.g., ۱, ۲, ۳, ۴, ..., ۹۸, ۹۹, ۱۰۰).

'lower-roman'
: Lowercase ASCII Roman numerals (e.g., i, ii, iii, ..., xcviii, xcix, c).

'upper-roman'
: Uppercase ASCII Roman numerals (e.g., I, II, III, ..., XCVIII, XCIX, C).

'tamil'
: Tamil numbering (e.g., க, உ, ங, ..., கூ�அ, கூகூ, க00).

'telugu'
: Telugu numbering (e.g., ౧, ౨, ౩, ..., ౯౮, ౯౯, ౧౦౦).

'thai'
: Thai (Siamese) numbering (e.g., ๑, ๒, ๓, ..., ๙๘, ๙๙, ๑๐๐).

'tibetan'
: Tibetan numbering (e.g., ༡, ༢, ༣, ..., ༩༨, ༩༩, ༡༠༠).

The following stylesheet fragment provides the normative definition of these predefined counter styles:

```
@counter-style decimal {
  system: numeric;
  symbols: '0' '1' '2' '3' '4' '5' '6' '7' '8' '9';
}

@counter-style decimal-leading-zero {
  system: extends decimal;
  pad: 2 '0';
}

@counter-style arabic-indic {
  system: numeric;
  symbols: "\660" "\661" "\662" "\663" "\664" "\665" "\666" "\667" "\668" "\669";
  /* ٠ ١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩ */
}

@counter-style armenian {
  system: additive;
  range: 1 9999;
  additive-symbols: 9000 \554, 8000 \553, 7000 \552, 6000 \551, 5000 \550, 4000
\54F, 3000 \54E, 2000 \54D, 1000 \54C, 900 \54B, 800 \54A, 700 \549, 600 \548, 500
\547, 400 \546, 300 \545, 200 \544, 100 \543, 90 \542, 80 \541, 70 \540, 60 \53F,
50 \53E, 40 \53D, 30 \53C, 20 \53B, 10 \53A, 9 \539, 8 \538, 7 \537, 6 \536, 5
\535, 4 \534, 3 \533, 2 \532, 1 \531;
  /* 9000 Ք, 8000 Փ, 7000 Ւ, 6000 Ց, 5000 Ր, 4000 Տ, 3000 Վ, 2000 Ս, 1000 Ռ, 900 Ջ,
800 Պ, 700 Չ, 600 Ո, 500 Շ, 400 Ն, 300 Յ, 200 Մ, 100 Ճ, 90 Ղ, 80 Ձ, 70 Հ, 60 Կ, 50
Ծ, 40 Խ, 30 Լ, 20 Ի, 10 Ժ, 9 Թ, 8 Ը, 7 Է, 6 Զ, 5 Ե, 4 Դ, 3 Գ, 2 Բ, 1 Ա */
}

@counter-style upper-armenian {
  system: extends armenian;
}

@counter-style lower-armenian {
  system: additive;
  range: 1 9999;
  additive-symbols: 9000 "\584", 8000 "\583", 7000 "\582", 6000 "\581", 5000
"\580", 4000 "\57F", 3000 "\57E", 2000 "\57D", 1000 "\57C", 900 "\57B", 800 "\57A",
700 "\579", 600 "\578", 500 "\577", 400 "\576", 300 "\575", 200 "\574", 100 "\573",
90 "\572", 80 "\571", 70 "\570", 60 "\56F", 50 "\56E", 40 "\56D", 30 "\56C", 20
"\56B", 10 "\56A", 9 "\569", 8 "\568", 7 "\567", 6 "\566", 5 "\565", 4 "\564", 3
"\563", 2 "\562", 1 "\561";
  /* 9000 ք, 8000 փ, 7000 ւ, 6000 ց, 5000 ր, 4000 տ, 3000 վ, 2000 ս, 1000 ռ, 900 ջ,
```

```
800 ҷ, 700 ӌ, 600 ո, 500 շ, 400 և, 300 ӻ, 200 ӎ, 100 ճ, 90 ղ, 80 ȣ, 70 հ, 60 կ, 50
ծ, 40 ӽ, 30 լ, 20 ի, 10 ժ, 9 թ, 8 ը, 7 է, 6 զ, 5 ե, 4 դ, 3 գ, 2 բ, 1 ա */
}

@counter-style bengali {
  system: numeric;
  symbols: "\9E6" "\9E7" "\9E8" "\9E9" "\9EA" "\9EB" "\9EC" "\9ED" "\9EE" "\9EF";
  /* ০ ১ ২ ৩ ৪ ৫ ৬ ৭ ৮ ৯ */
}

@counter-style cambodian {
  system: numeric;
  symbols: "\17E0" "\17E1" "\17E2" "\17E3" "\17E4" "\17E5" "\17E6" "\17E7" "\17E8"
"\17E9";
  /* ០ ១ ២ ៣ ៤ ៥ ៦ ៧ ៨ ៩ */
}

@counter-style khmer {
  system: extends cambodian;
}

@counter-style cjk-decimal {
  system: numeric;
  range: 0 infinite;
  symbols: \3007  \4E00  \4E8C  \4E09  \56DB  \4E94  \516D  \4E03  \516B  \4E5D;
  /* 〇 一 二 三 四 五 六 七 八 九 */
  suffix: "\3001";
  /* "、" */
}

@counter-style devanagari {
  system: numeric;
  symbols: "\966" "\967" "\968" "\969" "\96A" "\96B" "\96C" "\96D" "\96E" "\96F";
  /* ० १ २ ३ ४ ५ ६ ७ ८ ९ */
}

@counter-style georgian {
  system: additive;
  range: 1 19999;
  additive-symbols: 10000 \10F5, 9000 \10F0, 8000 \10EF, 7000 \10F4, 6000 \10EE,
5000 \10ED, 4000 \10EC, 3000 \10EB, 2000 \10EA, 1000 \10E9, 900 \10E8, 800 \10E7,
700 \10E6, 600 \10E5, 500 \10E4, 400 \10F3, 300 \10E2, 200 \10E1, 100 \10E0, 90
\10DF, 80 \10DE, 70 \10DD, 60 \10F2, 50 \10DC, 40 \10DB, 30 \10DA, 20 \10D9, 10
```

```
\10D8, 9 \10D7, 8 \10F1, 7 \10D6, 6 \10D5, 5 \10D4, 4 \10D3, 3 \10D2, 2 \10D1, 1
\10D0;
  /* 10000 ჸ, 9000 ჷ, 8000 ჶ, 7000 ჵ, 6000 ხ, 5000 ჴ, 4000 წ, 3000 ძ, 2000 ც, 1000
ჩ, 900 შ, 800 ყ, 700 ღ, 600 ქ, 500 ფ, 400 ჳ, 300 ტ, 200 ს, 100 რ, 90 ჟ, 80 პ, 70
ო, 60 ნ, 50 მ, 40 ლ, 30 კ, 20 ი, 10 თ, 9 ჱ, 8 ზ, 6 ვ, 5 ე, 4 დ, 3 გ, 2 ბ, 1
ა */
}

@counter-style gujarati {
  system: numeric;
  symbols: "\AE6" "\AE7" "\AE8" "\AE9" "\AEA" "\AEB" "\AEC" "\AED" "\AEE" "\AEF";
  /* ૦ ૧ ૨ ૩ ૪ ૫ ૬ ૭ ૮ ૯ */
}

@counter-style gurmukhi {
  system: numeric;
  symbols: "\A66" "\A67" "\A68" "\A69" "\A6A" "\A6B" "\A6C" "\A6D" "\A6E" "\A6F";
  /* ੦ ੧ ੨ ੩ ੪ ੫ ੬ ੭ ੮ ੯ */
}

@counter-style hebrew {
  system: additive;
  range: 1 10999;
  additive-symbols: 10000 \5D9\5F3, 9000 \5D8\5F3, 8000 \5D7\5F3, 7000 \5D6\5F3,
6000 \5D5\5F3, 5000 \5D4\5F3, 4000 \5D3\5F3, 3000 \5D2\5F3, 2000 \5D1\5F3, 1000
\5D0\5F3, 400 \5EA, 300 \5E9, 200 \5E8, 100 \5E7, 90 \5E6, 80 \5E4, 70 \5E2, 60
\5E1, 50 \5E0, 40 \5DE, 30 \5DC, 20 \5DB, 19 \5D9\5D8, 18 \5D9\5D7, 17 \5D9\5D6, 16
\5D8\5D6, 15 \5D8\5D5, 10 \5D9, 9 \5D8, 8 \5D7, 7 \5D6, 6 \5D5, 5 \5D4, 4 \5D3, 3
\5D2, 2 \5D1, 1 \5D0;
  /* 10000 י׳, 9000 ט׳, 8000 ח׳, 7000 ז׳, 6000 ו׳, 5000 ה׳, 4000 ד׳, 3000 ג׳, 2000 ב׳, 1000
א׳, 400 ת, 300 ש, 200 ר, 100 ק, 90 צ, 80 פ, 70 ע, 60 ס, 50 נ, 40 מ, 30 ל, 20 כ, 19 יט,
18 יח, 17 יז, 16 טז, 15 וט, 10 י, 9 ט, 8 ח, 7 ז, 6 ו, 5 ה, 4 ד, 3 ג, 2 ב, 1 א */
  /* This system manually specifies the values for 19-15 to force the correct
display of 15 and 16, which are commonly rewritten to avoid a close resemblance to
the Tetragrammaton. */
  /* Implementations MAY choose to implement this manually to a higher range; see
note below. */
}

@counter-style kannada {
  system: numeric;
  symbols: "\CE6" "\CE7" "\CE8" "\CE9" "\CEA" "\CEB" "\CEC" "\CED" "\CEE" "\CEF";
  /* ೦ ೧ ೨ ೩ ೪ ೫ ೬ ೭ ೮ ೯ */
```

```
}

@counter-style lao {
  system: numeric;
  symbols: "\ED0" "\ED1" "\ED2" "\ED3" "\ED4" "\ED5" "\ED6" "\ED7" "\ED8" "\ED9";
  /* ๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙ */
}

@counter-style malayalam {
  system: numeric;
  symbols: "\D66" "\D67" "\D68" "\D69" "\D6A" "\D6B" "\D6C" "\D6D" "\D6E" "\D6F";
  /* ൦ ൧ ൨ ൩ ൪ ൫ ൬ ൭ ൮ ൯ */
}

@counter-style mongolian {
  system: numeric;
  symbols: "\1810" "\1811" "\1812" "\1813" "\1814" "\1815" "\1816" "\1817" "\1818"
"\1819";
  /* ᠐ ᠑ ᠒ ᠓ ᠔ ᠕ ᠖ ᠗ ᠘ ᠙ */
}

@counter-style myanmar {
  system: numeric;
  symbols: "\1040" "\1041" "\1042" "\1043" "\1044" "\1045" "\1046" "\1047" "\1048"
"\1049";
  /* ๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙ */
}

@counter-style oriya {
  system: numeric;
  symbols: "\B66" "\B67" "\B68" "\B69" "\B6A" "\B6B" "\B6C" "\B6D" "\B6E" "\B6F";
  /* ୦ ୧ ୨ ୩ ୪ ୫ ୬ ୭ ୮ ୯ */
}

@counter-style persian {
  system: numeric;
  symbols: "\6F0" "\6F1" "\6F2" "\6F3" "\6F4" "\6F5" "\6F6" "\6F7" "\6F8" "\6F9";
  /* ۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ */
}

@counter-style lower-roman {
  system: additive;
  range: 1 3999;
```

```
  additive-symbols: 1000 m, 900 cm, 500 d, 400 cd, 100 c, 90 xc, 50 l, 40 xl, 10 x,
9 ix, 5 v, 4 iv, 1 i;
}

@counter-style upper-roman {
  system: additive;
  range: 1 3999;
  additive-symbols: 1000 M, 900 CM, 500 D, 400 CD, 100 C, 90 XC, 50 L, 40 XL, 10 X,
9 IX, 5 V, 4 IV, 1 I;
}

@counter-style tamil {
  system: numeric;
  symbols: "\BE6" "\BE7" "\BE8" "\BE9" "\BEA" "\BEB" "\BEC" "\BED" "\BEE" "\BEF";
  /* ൦ ക ഉ ങ സ ഩ ശ ഒ അ ക */
}

@counter-style telugu {
  system: numeric;
  symbols: "\C66" "\C67" "\C68" "\C69" "\C6A" "\C6B" "\C6C" "\C6D" "\C6E" "\C6F";
  /* ౦ ౧ ౨ ౩ ౪ ౫ ౬ ౭ ౮ ౯ */
}

@counter-style thai {
  system: numeric;
  symbols: "\E50" "\E51" "\E52" "\E53" "\E54" "\E55" "\E56" "\E57" "\E58" "\E59";
  /* ๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙ */
}

@counter-style tibetan {
  system: numeric;
  symbols: "\F20" "\F21" "\F22" "\F23" "\F24" "\F25" "\F26" "\F27" "\F28" "\F29";
  /* ༠ ༡ ༢ ༣ ༤ ༥ ༦ ༧ ༨ ༩ */
}
```

Implementations must implement ‘hebrew’ at least to the range specified in the ‘@counter-style’ rule above, but may implement it to a higher range. If they do so, the corresponding ‘range’ descriptor must reflect the implemented range.

§ 6.2. Alphabetic: ‘lower-alpha’, ‘lower-latin’, ‘upper-alpha’, ‘upper-latin’, ‘lower-greek’, ‘hiragana’, ‘hiragana-iroha’, ‘katakana’, ‘katakana-iroha’

'lower-alpha'
'lower-latin'
: Lowercase ASCII letters (e.g., a, b, c, ..., z, aa, ab).

'upper-alpha'
'upper-latin'
: Uppercase ASCII letters (e.g., A, B, C, ..., Z, AA, AB).

'lower-greek'
: Lowercase classical Greek (e.g., α, β, γ, ..., ω, αα, αβ).

'hiragana'
: Dictionary-order hiragana lettering (e.g., あ, い, う, ..., ん, ああ, あい).

'hiragana-iroha'
: Iroha-order hiragana lettering (e.g., い, ろ, は, ..., す, いい, いろ).

'katakana'
: Dictionary-order katakana lettering (e.g., ア, イ, ウ, ..., ン, アア, アイ).

'katakana-iroha'
: Iroha-order katakana lettering (e.g., イ, ロ, ハ, ..., ス, イイ, イロ)

The following stylesheet fragment provides the normative definition of these predefined counter styles:

```
@counter-style lower-alpha {
  system: alphabetic;
  symbols: a b c d e f g h i j k l m n o p q r s t u v w x y z;
}

@counter-style lower-latin {
  system: extends lower-alpha;
}

@counter-style upper-alpha {
  system: alphabetic;
  symbols: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z;
}

@counter-style upper-latin {
  system: extends upper-alpha;
}

@counter-style lower-greek {
  system: alphabetic;
  symbols: "\3B1" "\3B2" "\3B3" "\3B4" "\3B5" "\3B6" "\3B7" "\3B8" "\3B9" "\3BA"
```

```
"\3BB" "\3BC" "\3BD" "\3BE" "\3BF" "\3C0" "\3C1" "\3C3" "\3C4" "\3C5" "\3C6" "\3C7"
"\3C8" "\3C9";
  /* α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω */
}

@counter-style hiragana {
  system: alphabetic;
  symbols: "\3042" "\3044" "\3046" "\3048" "\304A" "\304B" "\304D" "\304F" "\3051"
"\3053" "\3055" "\3057" "\3059" "\305B" "\305D" "\305F" "\3061" "\3064" "\3066"
"\3068" "\306A" "\306B" "\306C" "\306D" "\306E" "\306F" "\3072" "\3075" "\3078"
"\307B" "\307E" "\307F" "\3080" "\3081" "\3082" "\3084" "\3086" "\3088" "\3089"
"\308A" "\308B" "\308C" "\308D" "\308F" "\3090" "\3091" "\3092" "\3093";
  /* あ い う え お か き く け こ さ し す せ そ た ち つ て と な に ぬ ね の は ひ ふ
へ ほ ま み む め も や ゆ よ ら り る れ ろ わ ゐ ゑ を ん */
  suffix: "、";
}

@counter-style hiragana-iroha {
  system: alphabetic;
  symbols: "\3044" "\308D" "\306F" "\306B" "\307B" "\3078" "\3068" "\3061" "\308A"
"\306C" "\308B" "\3092" "\308F" "\304B" "\3088" "\305F" "\308C" "\305D" "\3064"
"\306D" "\306A" "\3089" "\3080" "\3046" "\3090" "\306E" "\304A" "\304F" "\3084"
"\307E" "\3051" "\3075" "\3053" "\3048" "\3066" "\3042" "\3055" "\304D" "\3086"
"\3081" "\307F" "\3057" "\3091" "\3072" "\3082" "\305B" "\3059";
  /* い ろ は に ほ へ と ち り ぬ る を わ か よ た れ そ つ ね な ら む う ゐ の お く
や ま け ふ こ え て あ さ き ゆ め み し ゑ ひ も せ す */
  suffix: "、";
}

@counter-style katakana {
  system: alphabetic;
  symbols: "\30A2" "\30A4" "\30A6" "\30A8" "\30AA" "\30AB" "\30AD" "\30AF" "\30B1"
"\30B3" "\30B5" "\30B7" "\30B9" "\30BB" "\30BD" "\30BF" "\30C1" "\30C4" "\30C6"
"\30C8" "\30CA" "\30CB" "\30CC" "\30CD" "\30CE" "\30CF" "\30D2" "\30D5" "\30D8"
"\30DB" "\30DE" "\30DF" "\30E0" "\30E1" "\30E2" "\30E4" "\30E6" "\30E8" "\30E9"
"\30EA" "\30EB" "\30EC" "\30ED" "\30EF" "\30F0" "\30F1" "\30F2" "\30F3";
  /* ア イ ウ エ オ カ キ ク ケ コ サ シ ス セ ソ タ チ ツ テ ト ナ ニ ヌ ネ ノ ハ ヒ フ
ヘ ホ マ ミ ム メ モ ヤ ユ ヨ ラ リ ル レ ロ ワ ヰ ヱ ヲ ン */
  suffix: "、";
}

@counter-style katakana-iroha {
  system: alphabetic;
```

```
  symbols: "\30A4" "\30ED" "\30CF" "\30CB" "\30DB" "\30D8" "\30C8" "\30C1" "\30EA"
"\30CC" "\30EB" "\30F2" "\30EF" "\30AB" "\30E8" "\30BF" "\30EC" "\30BD" "\30C4"
"\30CD" "\30CA" "\30E9" "\30E0" "\30A6" "\30F0" "\30CE" "\30AA" "\30AF" "\30E4"
"\30DE" "\30B1" "\30D5" "\30B3" "\30A8" "\30C6" "\30A2" "\30B5" "\30AD" "\30E6"
"\30E1" "\30DF" "\30B7" "\30F1" "\30D2" "\30E2" "\30BB" "\30B9";
  /* イ ロ ハ ニ ホ ヘ ト チ リ ヌ ル ヲ ワ カ ヨ タ レ ソ ツ ネ ナ ラ ム ウ ヰ ノ オ ク
ヤ マ ケ フ コ エ テ ア サ キ ユ メ ミ シ ヱ ヒ モ セ ス */
  suffix: "、";
}
```

§ 6.3. Symbolic: ‘disc’, ‘circle’, ‘square’, ‘disclosure-open’, ‘disclosure-closed’

‘disc’

A filled circle, similar to • U+2022 BULLET.

‘circle’

A hollow circle, similar to ◦ U+25E6 WHITE BULLET.

‘square’

A filled square, similar to ▪ U+25AA BLACK SMALL SQUARE.

‘disclosure-open’
‘disclosure-closed’

Symbols appropriate for indicating an open or closed disclosure widget, such as the HTML `<details>` element.

The following stylesheet fragment provides the normative definition of these predefined counter styles:

```
@counter-style disc {
  system: cyclic;
  symbols: \2022;
  /* • */
  suffix: " ";
}

@counter-style circle {
  system: cyclic;
  symbols: \25E6;
  /* ◦ */
  suffix: " ";
}

@counter-style square {
```

```
  system: cyclic;
  symbols: \25AA;
  /* ▪ */
  suffix: " ";
}

@counter-style disclosure-open {
  system: cyclic;
  suffix: " ";
  /* for symbols, see normative text below */
}

@counter-style disclosure-closed {
  system: cyclic;
  suffix: " ";
  /* for symbols, see normative text below */
}
```

When used in 'list-style-type', a UA may instead render these styles using a UA-generated image or a UA-chosen font instead of rendering the specified character in the element's own font. If using an image, it must look similar to the character, and must be sized to attractively fill a 1em by 1em square.

For the 'disclosure-open' and 'disclosure-closed' counter styles, the marker must be an image or character suitable for indicating the open and closed states of a disclosure widget, such as HTML's `<details>` element. If the image is directional, it must respond to the writing mode of the element, similar to the bidi-sensitive images feature of the Images 4 module. For example, the 'disclosure-closed' style might use the characters U+25B8 BLACK RIGHT-POINTING SMALL TRIANGLE (▸) and U+25C2 BLACK LEFT-POINTING SMALL TRIANGLE (◂), while the 'disclosure-open' style might use the character U+25BE BLACK DOWN-POINTING SMALL TRIANGLE (▾).

§ 6.4. Fixed: 'cjk-earthly-branch', 'cjk-heavenly-stem'

'cjk-earthly-branch'
: Han "Earthly Branch" ordinals (e.g., 子, 丑, 寅, ..., 亥).

'cjk-heavenly-stem'
: Han "Heavenly Stem" ordinals (e.g., 甲, 乙, 丙, ..., 癸)

The following stylesheet fragment provides the normative definition of these predefined counter styles:

```
@counter-style cjk-earthly-branch {
  system: fixed;
  symbols: "\5B50" "\4E11" "\5BC5" "\536F" "\8FB0" "\5DF3" "\5348" "\672A" "\7533"
"\9149" "\620C" "\4EA5";
  /* 子 丑 寅 卯 辰 巳 午 未 申 酉 戌 亥 */
  suffix: "、";
}

@counter-style cjk-heavenly-stem {
  system: fixed;
  symbols: "\7532" "\4E59" "\4E19" "\4E01" "\620A" "\5DF1" "\5E9A" "\8F9B" "\58EC"
"\7678";
  /* 甲 乙 丙 丁 戊 己 庚 辛 壬 癸 */
  suffix: "、";
}
```

§ 7. Complex Predefined Counter Styles

While authors may define their own counter styles using the ‘@counter-style’ rule or rely on the set of predefined counter styles, a few counter styles are described by rules that are too complex to be captured by the predefined algorithms. These counter styles are described in this section.

Some of the counter styles specified in this section have custom algorithms for generating counter values, but are otherwise identical to a counter style defined via the ‘@counter-style’ rule. For example, an author can reference one of these styles in an ‘extends’ system, reusing the algorithm but swapping out some of the other descriptors.

All of the counter styles defined in this section have a ‘spoken form’ of ‘numbers’, and use a negative sign.

§ 7.1. Longhand East Asian Counter Styles

Chinese, Japanese, and Korean have counter styles which have a “longhand” nature, similar to “thirteen thousand one hundred and twenty-three” in English. Each has both formal and informal variants. The formal styles are typically used in financial and legal documents, as their characters are more difficult to alter into each other.

EXAMPLE 13

The following table shows examples of these styles, particularly some ways in which they differ.

Counter Style	0	1	2	3	10	11	99	100	101	6001
‘japanese-informal’	〇	一	二	三	十	十一	九十九	百	百一	六千一
‘japanese-formal’	零	壱	弐	参	壱拾	壱拾壱	九拾九	壱百	壱百壱	六阡壱
‘korean-hangul-formal’	영	일	이	삼	일십	일십일	구십구	일백	일백일	육천일
‘korean-hanja-informal’	零	一	二	三	十	十一	九十九	百	百一	六千一
‘korean-hanja-formal’	零	壹	貳	參	壹拾	壹拾壹	九拾九	壹百	壹百壹	六仟壹
‘simp-chinese-informal’	零	一	二	三	十	十一	九十九	一百	一百零一	六千零一
‘simp-chinese-formal’	零	壹	贰	叁	壹拾	壹拾壹	玖拾玖	壹佰	壹佰零壹	陆仟零壹
‘trad-chinese-informal’	零	一	二	三	十	十一	九十九	一百	一百零一	六千零一

Counter Style	0	1	2	3	10	11	99	100	101	6001
‘trad-chinese-formal’	零	壹	貳	參	壹拾	壹拾壹	玖拾玖	壹佰	壹佰零壹	陸仟零壹

Because opinions differ on how best to represent numbers 10k or greater using the longhand CJK styles, all of the counter styles defined in this section are defined to have a range of -9999 to 9999, but implementations may support a larger range. Outside the implementation-supported range, the fallback is ‘cjk-decimal’.

Note: Implementations are encouraged to research and implement counter representations beyond 10k and report back to the CSS Working Group with data when a generally-accepted answer is discovered. Some previous research on this topic is contained in an earlier draft.

§ 7.1.1. Japanese: ‘japanese-informal’ and ‘japanese-formal’

‘japanese-informal’
: Informal Japanese Kanji numbering (e.g., 千百十一)

‘japanese-formal’
: Formal Japanese Kanji numbering (e.g. 壱阡壱百壱拾壱)

```
@counter-style japanese-informal {
  system: additive;
  range: -9999 9999;
  additive-symbols: 9000 \4E5D\5343, 8000 \516B\5343, 7000 \4E03\5343, 6000
\516D\5343, 5000 \4E94\5343, 4000 \56DB\5343, 3000 \4E09\5343, 2000 \4E8C\5343,
1000 \5343, 900 \4E5D\767E, 800 \516B\767E, 700 \4E03\767E, 600 \516D\767E, 500
\4E94\767E, 400 \56DB\767E, 300 \4E09\767E, 200 \4E8C\767E, 100 \767E, 90
\4E5D\5341, 80 \516B\5341, 70 \4E03\5341, 60 \516D\5341, 50 \4E94\5341, 40
\56DB\5341, 30 \4E09\5341, 20 \4E8C\5341, 10 \5341, 9 \4E5D, 8 \516B, 7 \4E03, 6
\516D, 5 \4E94, 4 \56DB, 3 \4E09, 2 \4E8C, 1 \4E00, 0 \3007;
  /* 9000 九千, 8000 八千, 7000 七千, 6000 六千, 5000 五千, 4000 四千, 3000 三千,
2000 二千, 1000 千, 900 九百, 800 八百, 700 七百, 600 六百, 500 五百, 400 四百, 300 三
百, 200 二百, 100 百, 90 九十, 80 八十, 70 七十, 60 六十, 50 五十, 40 四十, 30 三十,
20 二十, 10 十, 9 九, 8 八, 7 七, 6 六, 5 五, 4 四, 3 三, 2 二, 1 一, 0 〇 */
  suffix: '\3001';
```

```
  /* 、 */
  negative: "\30DE\30A4\30CA\30B9";
  /* マイナス */
  fallback: cjk-decimal;
}

@counter-style japanese-formal {
  system: additive;
  range: -9999 9999;
  additive-symbols: 9000 \4E5D\9621, 8000 \516B\9621, 7000 \4E03\9621, 6000
\516D\9621, 5000 \4F0D\9621, 4000 \56DB\9621, 3000 \53C2\9621, 2000 \5F10\9621,
1000 \58F1\9621, 900 \4E5D\767E, 800 \516B\767E, 700 \4E03\767E, 600 \516D\767E,
500 \4F0D\767E, 400 \56DB\767E, 300 \53C2\767E, 200 \5F10\767E, 100 \58F1\767E, 90
\4E5D\62FE, 80 \516B\62FE, 70 \4E03\62FE, 60 \516D\62FE, 50 \4F0D\62FE, 40
\56DB\62FE, 30 \53C2\62FE, 20 \5F10\62FE, 10 \58F1\62FE, 9 \4E5D, 8 \516B, 7 \4E03,
6 \516D, 5 \4F0D, 4 \56DB, 3 \53C2, 2 \5F10, 1 \58F1, 0 \96F6;
  /* 9000 九阡, 8000 八阡, 7000 七阡, 6000 六阡, 5000 伍阡, 4000 四阡, 3000 参阡,
2000 弐阡, 1000 壱阡, 900 九百, 800 八百, 700 七百, 600 六百, 500 伍百, 400 四百, 300
参百, 200 弐百, 100 壱百, 90 九拾, 80 八拾, 70 七拾, 60 六拾, 50 伍拾, 40 四拾, 30 参
拾, 20 弐拾, 10 壱拾, 9 九, 8 八, 7 七, 6 六, 5 伍, 4 四, 3 参, 2 弐, 1 壱, 0 零 */
  suffix: '\3001';
  /* 、 */
  negative: "\30DE\30A4\30CA\30B9";
  /* マイナス */
  fallback: cjk-decimal;
}
```

§ 7.1.2. Korean: ‘korean-hangul-formal’, ‘korean-hanja-informal’, and ‘korean-hanja-formal’

‘korean-hangul-formal’

Korean Hangul numbering (e.g., 일천일백일십일)

‘korean-hanja-informal’

Informal Korean Hanja numbering (e.g., 千百十一)

‘korean-hanja-formal’

Formal Korean Han (Hanja) numbering (e.g., 壹仟壹百壹拾壹)

```
@counter-style korean-hangul-formal {
  system: additive;
  range: -9999 9999;
  additive-symbols: 9000 \AD6C\CC9C, 8000 \D314\CC9C, 7000 \CE60\CC9C, 6000
\C721\CC9C, 5000 \C624\CC9C, 4000 \C0AC\CC9C, 3000 \C0BC\CC9C, 2000 \C774\CC9C,
1000 \C77C\CC9C, 900 \AD6C\BC31, 800 \D314\BC31, 700 \CE60\BC31, 600 \C721\BC31,
```

```
500 \C624\BC31, 400 \C0AC\BC31, 300 \C0BC\BC31, 200 \C774\BC31, 100 \C77C\BC31, 90
\AD6C\C2ED, 80 \D314\C2ED, 70 \CE60\C2ED, 60 \C721\C2ED, 50 \C624\C2ED, 40
\C0AC\C2ED, 30 \C0BC\C2ED, 20 \C774\C2ED, 10 \C77C\C2ED, 9 \AD6C, 8 \D314, 7 \CE60,
6 \C721, 5 \C624, 4 \C0AC, 3 \C0BC, 2 \C774, 1 \C77C, 0 \C601;
  /* 9000 구천, 8000 팔천, 7000 칠천, 6000 육천, 5000 오천, 4000 사천, 3000 삼천,
2000 이천, 1000 일천, 900 구백, 800 팔백, 700 칠백, 600 육백, 500 오백, 400 사백, 300
삼백, 200 이백, 100 일백, 90 구십, 80 팔십, 70 칠십, 60 육십, 50 오십, 40 사십, 30 삼
십, 20 이십, 10 일십, 9 구, 8 팔, 7 칠, 6 육, 5 오, 4 사, 3 삼, 2 이, 1 일, 0 영 */
  suffix: ', ';
  negative: "\B9C8\C774\B108\C2A4  ";
  /* 마이너스 (followed by a space) */
}

@counter-style korean-hanja-informal {
  system: additive;
  range: -9999 9999;
  additive-symbols: 9000 \4E5D\5343, 8000 \516B\5343, 7000 \4E03\5343, 6000
\516D\5343, 5000 \4E94\5343, 4000 \56DB\5343, 3000 \4E09\5343, 2000 \4E8C\5343,
1000 \5343, 900 \4E5D\767E, 800 \516B\767E, 700 \4E03\767E, 600 \516D\767E, 500
\4E94\767E, 400 \56DB\767E, 300 \4E09\767E, 200 \4E8C\767E, 100 \767E, 90
\4E5D\5341, 80 \516B\5341, 70 \4E03\5341, 60 \516D\5341, 50 \4E94\5341, 40
\56DB\5341, 30 \4E09\5341, 20 \4E8C\5341, 10 \5341, 9 \4E5D, 8 \516B, 7 \4E03, 6
\516D, 5 \4E94, 4 \56DB, 3 \4E09, 2 \4E8C, 1 \4E00, 0 \96F6;
  /* 9000 九千, 8000 八千, 7000 七千, 6000 六千, 5000 五千, 4000 四千, 3000 三千,
2000 二千, 1000 千, 900 九百, 800 八百, 700 七百, 600 六百, 500 五百, 400 四百, 300 三
百, 200 二百, 100 百, 90 九十, 80 八十, 70 七十, 60 六十, 50 五十, 40 四十, 30 三十,
20 二十, 10 十, 9 九, 8 八, 7 七, 6 六, 5 五, 4 四, 3 三, 2 二, 1 一, 0 零 */
  suffix: ', ';
  negative: "\B9C8\C774\B108\C2A4  ";
  /* 마이너스 (followed by a space) */
}

@counter-style korean-hanja-formal {
  system: additive;
  range: -9999 9999;
  additive-symbols: 9000 \4E5D\4EDF, 8000 \516B\4EDF, 7000 \4E03\4EDF, 6000
\516D\4EDF, 5000 \4E94\4EDF, 4000 \56DB\4EDF, 3000 \53C3\4EDF, 2000 \8CB3\4EDF,
1000 \58F9\4EDF, 900 \4E5D\767E, 800 \516B\767E, 700 \4E03\767E, 600 \516D\767E,
500 \4E94\767E, 400 \56DB\767E, 300 \53C3\767E, 200 \8CB3\767E, 100 \58F9\767E, 90
\4E5D\62FE, 80 \516B\62FE, 70 \4E03\62FE, 60 \516D\62FE, 50 \4E94\62FE, 40
\56DB\62FE, 30 \53C3\62FE, 20 \8CB3\62FE, 10 \58F9\62FE, 9 \4E5D, 8 \516B, 7 \4E03,
6 \516D, 5 \4E94, 4 \56DB, 3 \53C3, 2 \8CB3, 1 \58F9, 0 \96F6;
  /* 9000 九仟, 8000 八仟, 7000 七仟, 6000 六仟, 5000 五仟, 4000 四仟, 3000 參仟,
```

```
2000 貳仟, 1000 壹仟, 900 九百, 800 八百, 700 七百, 600 六百, 500 五百, 400 四百, 300
參百, 200 貳百, 100 壹百, 90 九拾, 80 八拾, 70 七拾, 60 六拾, 50 五拾, 40 四拾, 30 參
拾, 20 貳拾, 10 壹拾, 9 九, 8 八, 7 七, 6 六, 5 五, 4 四, 3 參, 2 貳, 1 壹, 0 零 */
  suffix: ', ';
  negative: "\B9C8\C774\B108\C2A4  ";
  /* 마이너스 (followed by a space) */
}
```

§ 7.1.3. Chinese: ‘simp-chinese-informal’, ‘simp-chinese-formal’, ‘trad-chinese-informal’, and ‘trad-chinese-formal’

‘simp-chinese-informal’
: Simplified Chinese informal numbering (e.g., 一千一百一十一)

‘simp-chinese-formal’
: Simplified Chinese formal numbering (e.g. 壹仟壹佰壹拾壹)

‘trad-chinese-informal’
: Traditional Chinese informal numbering (e.g., 一千一百一十一)

‘trad-chinese-formal’
: Traditional Chinese formal numbering (e.g., 壹仟壹佰壹拾壹)

‘cjk-ideographic’
: This counter style is identical to ‘trad-chinese-informal’. (It exists for legacy reasons.)

The Chinese longhand styles are defined by almost identical algorithms (specified as a single algorithm here, with the differences called out when relevant), but use different sets of characters, as specified by the table following the algorithm.

1. If the counter value is 0, the representation is the character for 0 specified for the given counter style. Skip the rest of this algorithm.
2. Initially represent the counter value as a decimal number. For each digit that is not 0, append the appropriate digit marker to the digit. The ones digit has no marker.
3. For the informal styles, if the counter value is between ten and nineteen, remove the tens digit (leave the digit marker).
4. Drop any trailing zeros and collapse any remaining zeros into a single zero digit.
5. Replace the digits 0-9 with the appropriate character for the given counter style. Return the resultant string as the representation of the counter value.

For all of these counter styles, the ‘suffix’ is "、" U+3001, the ‘fallback’ is ‘cjk-decimal’, the ‘range’ is ‘-9999 9999’, and the ‘negative’ value is given in the table of symbols for each style.

The following tables define the characters used in these styles:

Values	Codepoints			
	simp-chinese-informal	**simp-chinese-formal**	**trad-chinese-informal**	**trad-chinese-formal**
Digit 0	零 U+96F6	零 U+96F6	零 U+96F6	零 U+96F6
Digit 1	一 U+4E00	壹 U+58F9	一 U+4E00	壹 U+58F9
Digit 2	二 U+4E8C	贰 U+8D30	二 U+4E8C	貳 U+8CB3
Digit 3	三 U+4E09	叁 U+53C1	三 U+4E09	參 U+53C3
Digit 4	四 U+56DB	肆 U+8086	四 U+56DB	肆 U+8086
Digit 5	五 U+4E94	伍 U+4F0D	五 U+4E94	伍 U+4F0D
Digit 6	六 U+516D	陆 U+9646	六 U+516D	陸 U+9678
Digit 7	七 U+4E03	柒 U+67D2	七 U+4E03	柒 U+67D2
Digit 8	八 U+516B	捌 U+634C	八 U+516B	捌 U+634C
Digit 9	九 U+4E5D	玖 U+7396	九 U+4E5D	玖 U+7396
Tens Digit Marker	十 U+5341	拾 U+62FE	十 U+5341	拾 U+62FE
Hundreds Digit Marker	百 U+767E	佰 U+4F70	百 U+767E	佰 U+4F70
Thousands Digit Marker	千 U+5343	仟 U+4EDF	千 U+5343	仟 U+4EDF
Negative Sign	负 U+8D1F	负 U+8D1F	負 U+8CA0	負 U+8CA0

For reference, here are the first 120 values for the ‘simp-chinese-informal’ style:

1	一	41	四十一	81	八十一
2	二	42	四十二	82	八十二
3	三	43	四十三	83	八十三
4	四	44	四十四	84	八十四
5	五	45	四十五	85	八十五
6	六	46	四十六	86	八十六
7	七	47	四十七	87	八十七
8	八	48	四十八	88	八十八
9	九	49	四十九	89	八十九
10	十	50	五十	90	九十
11	十一	51	五十一	91	九十一
12	十二	52	五十二	92	九十二
13	十三	53	五十三	93	九十三
14	十四	54	五十四	94	九十四
15	十五	55	五十五	95	九十五
16	十六	56	五十六	96	九十六
17	十七	57	五十七	97	九十七
18	十八	58	五十八	98	九十八
19	十九	59	五十九	99	九十九
20	二十	60	六十	100	一百
21	二十一	61	六十一	101	一百零一
22	二十二	62	六十二	102	一百零二
23	二十三	63	六十三	103	一百零三
24	二十四	64	六十四	104	一百零四
25	二十五	65	六十五	105	一百零五
26	二十六	66	六十六	106	一百零六
27	二十七	67	六十七	107	一百零七
28	二十八	68	六十八	108	一百零八
29	二十九	69	六十九	109	一百零九
30	三十	70	七十	110	一百一十
31	三十一	71	七十一	111	一百一十一
32	三十二	72	七十二	112	一百一十二
33	三十三	73	七十三	113	一百一十三
34	三十四	74	七十四	114	一百一十四
35	三十五	75	七十五	115	一百一十五
36	三十六	76	七十六	116	一百一十六
37	三十七	77	七十七	117	一百一十七
38	三十八	78	七十八	118	一百一十八
39	三十九	79	七十九	119	一百一十九
40	四十	80	八十	120	一百二十

§ 7.2. Ethiopic Numeric Counter Style: ‘ethiopic-numeric’

The ***‘ethiopic-numeric’*** counter style is defined for all positive non-zero numbers. The following algorithm converts decimal digits to ethiopic numbers:

1. If the number is 1, return "፩" (U+1369).
2. Split the number into groups of two digits, starting with the least significant decimal digit.
3. Index each group sequentially, starting from the least significant as group number zero.
4. If the group has the value zero, or if the group is the most significant one and has the value 1, or if the group has an odd index (as given in the previous step) and has the value 1, then remove the digits (but leave the group, so it still has a separator appended below).
5. For each remaining digit, substitute the relevant ethiopic character from the list below.

Tens			Units		
Values	**Codepoints**		**Values**	**Codepoints**	
10	፲	U+1372	1	፩	U+1369
20	፳	U+1373	2	፪	U+136A
30	፴	U+1374	3	፫	U+136B
40	፵	U+1375	4	፬	U+136C
50	፶	U+1376	5	፭	U+136D
60	፷	U+1377	6	፮	U+136E
70	፸	U+1378	7	፯	U+136F
80	፹	U+1379	8	፰	U+1370
90	፺	U+137A	9	፱	U+1371

6. For each group with an odd index (as given in the second step), except groups which originally had a value of zero, append ፟ U+137B.
7. For each group with an even index (as given in the second step), except the group with index 0, append ፼ U+137C.
8. Concatenate the groups into one string, and return it.

For this system, the name is "ethiopic-numeric", the ‘range’ is ‘1 infinite’, the ‘suffix’ is "`/ `" (U+002F SOLIDUS followed by a U+0020 SPACE), and the rest of the descriptors have their initial value.

> EXAMPLE 14
> The decimal number 100, in ethiopic, is ፻ U+137B
>
> The decimal number 78010092, in ethiopic, is ፸፰፻፩፼፺፪ U+1378 U+1370 U+137B U+1369 U+137C U+137A U+136A.
>
> The decimal number 780100000092, in ethiopic, is ፸፰፻፩፼፼፼፺፪ U+1378 U+1370 U+137B U+1369 U+137C U+137C U+137A U+136A.

§ 8. Additional “Ready-made” Counter Styles

The Internationalization Working Group maintains a large list of ready-made ‘@counter-style’ rules for various world languages in their Ready-made Counter Styles document. [predefined-counter-styles]

These additional counter styles are not intended to be supported by user-agents by default, but can be used by users or authors copying them directly into style sheets.

§ 9. APIs

§ 9.1. Extensions to the `CSSRule` interface

The `CSSRule` interface is extended as follows:

```
partial interface CSSRule {
    const unsigned short COUNTER_STYLE_RULE = 11;
};
```

§ 9.2. The `CSSCounterStyleRule` interface

The CSSCounterStyleRule interface represents a ‘@counter-style’ rule.

```
[Exposed=Window]
interface CSSCounterStyleRule : CSSRule {
  attribute CSSOMString name;
  attribute CSSOMString system;
  attribute CSSOMString symbols;
  attribute CSSOMString additiveSymbols;
  attribute CSSOMString negative;
  attribute CSSOMString prefix;
  attribute CSSOMString suffix;
  attribute CSSOMString range;
  attribute CSSOMString pad;
  attribute CSSOMString speakAs;
  attribute CSSOMString fallback;
};
```

***name*, of type CSSOMString**

The *name* attribute on getting must return a `CSSOMString` object that contains the serialization of the <counter-style-name> defined for the associated rule.

On setting the *name* attribute, run the following steps:

1. If the value is an ASCII case-insensitive match for any of the predefined counter styles, lowercase it.
2. If the value is not "decimal", "disc", or "none", replace the associated rule's name with an identifier equal to the value.
3. Otherwise, do nothing.

***system*, of type CSSOMString**
***symbols*, of type CSSOMString**
***additiveSymbols*, of type CSSOMString**
***negative*, of type CSSOMString**
***prefix*, of type CSSOMString**
***suffix*, of type CSSOMString**
***range*, of type CSSOMString**
***pad*, of type CSSOMString**
***speakAs*, of type CSSOMString**
***fallback*, of type CSSOMString**

The remaining attributes on getting must return a `CSSOMString` object that contains the serialization of the associated descriptor defined for the associated rule. If the descriptor was not specified in the associated rule, the attribute must return an empty string.

On setting, run the following steps:

1. Parse a list of component values from the value.
2. If the returned value is invalid according to the given descriptor's grammar, or would cause the '@counter-style' rule to become invalid, do nothing and abort these steps. (For example, some systems require the 'symbols' descriptor to contain two values.)
3. If the attribute being set is system, and the new value would change the algorithm used, do nothing and abort these steps. It's okay to change an aspect of the algorithm, like the first symbol value of a 'fixed' system.
4. Set the descriptor to the value.

§ 10. Sample style sheet for HTML

This section is informative, not normative. HTML itself defines the styles that apply to its elements, and in some cases defers to the user agent's discretion.

```
details > summary {
  display: list-item;
  list-style: disclosure-closed inside;
}

details[open] > summary {
  list-style: disclosure-open inside;
}
```

§ Changes

§ Changes since the December 2017 Candidate Recommendation

Significant changes since the December 14 2017 Candidate Recommendation:

- Made 'none', 'decimal', 'disc', 'circle', 'square', 'disclosure-open', 'disclosure-close' non-overridable. (Issue 3584)
- Clarified counter-style lookups in Shadow DOM. (Issue 5693)
- Clarified what happens in various invalid '@counter-style' situations. (Issue 5698, Issue 5717)
- Fixed divide-by-zero error in additive algorithm. (Issue 5784)
- Fixed 'square' symbol to not use an emoji symbol. (Issue 6200)
- Allowed UAs to override the font choice of predefined symbolic counter styles. (Issue 6201)

- Restricted special rendering of predefined symbolic counter styles to usage as a list marker via 'list-style-type'. (Issue 6201)
- Clarified that 'speak-as' represents spoken output; it may be used for other AT. (Issue 6040)

> A counter style can be constructed with a meaning that is obvious visually, but impossible to meaningfully represent via a speech synthesizer or other non-visual means, or possible but nonsensical when naively read out loud . The 'speak-as' descriptor describes how to synthesize the spoken form of a counter formatted with the given counter style. Assistive technologies should use this spoken form when reading out the counter style, and may use the 'speak-as' value to inform transformations to outputs other than speech.

A Disposition of Comments is available.

§ Changes since the June 2015 Candidate Recommendation

Significant changes since the June 11 2015 Candidate Recommendation:

- Exclude 'none' and 'disc' from being the name of a counter style.
- When setting CSSCounterStyle.name, take the string directly; don't parse it as an ident.
- Clarify that counter styles are read out in the element's content language.
- Clarified that 'additive-symbols' tuples must be of *strictly* decreasing weight.
- Specified that invalid values just invalidate the declaration, not the whole rule.
- '@counter-style' rules that are invalid due to missing descriptors just fail to create a counter style; they're otherwise still valid rules.
- Changed syntax to use CSS bracketed range notation to reflect the prose restrictions on negative values.

A Disposition of Comments is available.

§ Changes since the Feb 2015 Candidate Recommendation

- Allowed UAs to extend the 'hebrew' style past the spec-defined limits (since the current limits are mostly just an artifact of how annoying it is to go higher with the '@counter-style'-based definition).

§ Acknowledgments

The following people and documentation they wrote were very useful for defining the numbering systems: Alexander Savenkov, Arron Eicholz, Aryeh Gregor, Christopher Hoess, Daniel Yacob, Frank Tang, Jonathan Rosenne, Karl Ove Hufthammer, Musheg Arakelyan, Nariné Renard Karapetyan, Randall Bart, Richard Ishida, Simon Montagu (Mozilla, smontagu@smontagu.org)

Special thanks to Xidorn Quan for *extensive* reviews of all aspects of the spec, and also to Simon Sapin and Håkon Wium Lie for their review comments.

§ Privacy and Security Considerations

This specification introduces no new privacy or security considerations.

§ 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 15
>
> 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.

§ CR exit criteria

For this specification to be advanced to Proposed Recommendation, there must be at least two independent, interoperable implementations of each feature. Each feature may be implemented by a different set of products, there is no requirement that all features be implemented by a single product. For the purposes of this criterion, we define the following terms:

independent
: each implementation must be developed by a different party and cannot share, reuse, or derive from code used by another qualifying implementation. Sections of code that have no bearing on the implementation of this specification are exempt from this requirement.

interoperable
: passing the respective test case(s) in the official CSS test suite, or, if the implementation is not a Web browser, an equivalent test. Every relevant test in the test suite should have an equivalent test created if such a user agent (UA) is to be used to claim interoperability. In addition if such a UA is to be used to claim interoperability, then there must one or more additional UAs which can also pass those equivalent tests in the same way for the purpose of interoperability. The equivalent tests must be made publicly available for the purposes of peer review.

implementation
a user agent which:

1. implements the specification.
2. is available to the general public. The implementation may be a shipping product or other publicly available version (i.e., beta version, preview release, or "nightly build"). Non-shipping product releases must have implemented the feature(s) for a period of at least one month in order to demonstrate stability.
3. is not experimental (i.e., a version specifically designed to pass the test suite and is not intended for normal usage going forward).

The specification will remain Candidate Recommendation for at least six months.

§ Index

§ Terms defined by this specification

§ Terms defined by reference

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

computed value

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

content

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

<image>

default object size

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

counter()

counters()

inside

list-style

list-style-type

none

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

::marker

[css-scoping-1] defines the following terms:

tree-scoped name

tree-scoped reference

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

- <declaration-list>
- at-rule
- parse
- parse a list of component values

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

- content language
- grapheme cluster

[css-text-decor-4] defines the following terms:

- symbols

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

- identifier

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

- #
- &&
- +
- <custom-ident>
- <integer>
- <string>
- ?
- css bracketed range notation
- css-wide keywords
- {a}
- |

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

- writing mode

[cssom-1] defines the following terms:

- CSSOMString
- CSSRule

[HTML] defines the following terms:

- details

[INFRA] defines the following terms:

- ascii case-insensitive
- continue

[WebIDL] defines the following terms:

- Exposed
- unsigned short

§ References

§ Normative References

[CSS-CASCADE-5]
Elika Etemad; Miriam Suzanne; Tab Atkins Jr.. CSS Cascading and Inheritance Level 5. 8 June 2021. WD. URL: https://www.w3.org/TR/css-cascade-5/

[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-LISTS-3]
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/

[CSS-SCOPING-1]
Tab Atkins Jr.; Elika Etemad. CSS Scoping Module Level 1. 3 April 2014. WD. URL: https://www.w3.org/TR/css-scoping-1/

[CSS-SYNTAX-3]
Tab Atkins Jr.; Simon Sapin. CSS Syntax Module Level 3. 16 July 2019. CR. URL: https://www.w3.org/TR/css-syntax-3/

[CSS-TEXT-3]
Elika Etemad; Koji Ishii; Florian Rivoal. CSS Text Module Level 3. 22 April 2021. CR. URL: https://www.w3.org/TR/css-text-3/

[CSS-TEXT-DECOR-4]
Elika Etemad; Koji Ishii. CSS Text Decoration Module Level 4. 6 May 2020. WD. URL: https://www.w3.org/TR/css-text-decor-4/

[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. 15 July 2021. WD. URL: https://www.w3.org/TR/css-values-4/

[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/

[CSSOM-1]
Simon Pieters; Glenn Adams. CSS Object Model (CSSOM). 17 March 2016. WD. URL: https://www.w3.org/TR/cssom-1/

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

[INFRA]
Anne van Kesteren; Domenic Denicola. Infra Standard. Living Standard. URL: https://infra.spec.whatwg.org/

[RFC2119]

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

[WebIDL]
Boris Zbarsky. Web IDL. 15 December 2016. ED. URL: https://heycam.github.io/webidl/

§ Informative References

[CSS-CONTENT-3]
Elika Etemad; Dave Cramer. CSS Generated Content Module Level 3. 2 August 2019. WD. URL: https://www.w3.org/TR/css-content-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/

[PREDEFINED-COUNTER-STYLES]
Richard Ishida. Ready-made Counter Styles. 9 June 2021. NOTE. URL: https://www.w3.org/TR/predefined-counter-styles/

§ Property Index

No properties defined.

§ '@counter-style' Descriptors

Name	Value	Initial
'additive-symbols'	[<integer [0,∞]> && <symbol>]#	n/a
'fallback'	<counter-style-name>	decimal
'negative'	<symbol> <symbol>?	"\2D" ("-" hyphen-minus)
'pad'	<integer [0,∞]> && <symbol>	0 ""
'prefix'	<symbol>	"" (the empty string)
'range'	[[<integer> | infinite]{2}]# | auto	auto
'speak-as'	auto | bullets | numbers | words | spell-out | <counter-style-name>	auto
'suffix'	<symbol>	"\2E\20" ("." full stop followed by a space)

Name	Value	Initial
'symbols'	<symbol>+	n/a
'system'	cyclic | numeric | alphabetic | symbolic | additive | [fixed <integer>?] | [extends <counter-style-name>]	symbolic

§ IDL Index

```
partial interface CSSRule {
    const unsigned short COUNTER_STYLE_RULE = 11;
};

[Exposed=Window]
interface CSSCounterStyleRule : CSSRule {
  attribute CSSOMString name;
  attribute CSSOMString system;
  attribute CSSOMString symbols;
  attribute CSSOMString additiveSymbols;
  attribute CSSOMString negative;
  attribute CSSOMString prefix;
  attribute CSSOMString suffix;
  attribute CSSOMString range;
  attribute CSSOMString pad;
  attribute CSSOMString speakAs;
  attribute CSSOMString fallback;
};
```