10 Visual formatting model details

Contents

10.1 Definition of "containing block"
10.2 Content width: the 'width' property
10.3 Calculating widths and margins
  10.3.1 Inline, non-replaced elements
  10.3.2 Inline, replaced elements
  10.3.3 Block-level, non-replaced elements in normal flow
  10.3.4 Block-level, replaced elements in normal flow
  10.3.5 Floating, non-replaced elements
  10.3.6 Floating, replaced elements
  10.3.7 Absolutely positioned, non-replaced elements
  10.3.8 Absolutely positioned, replaced elements
  10.3.9 'Inline-block', non-replaced elements in normal flow
  10.3.10 'Inline-block', replaced elements in normal flow
10.4 Minimum and maximum widths: 'min-width' and 'max-width'
10.5 Content height: the 'height' property
10.6 Calculating heights and margins
  10.6.1 Inline, non-replaced elements
  10.6.2 Inline replaced elements, block-level replaced elements in normal flow, 'inline-block' replaced elements in normal flow and floating replaced elements
  10.6.3 Block-level non-replaced elements in normal flow when 'overflow' computes to 'visible'
  10.6.4 Absolutely positioned, non-replaced elements
  10.6.5 Absolutely positioned, replaced elements
  10.6.6 Complicated cases
  10.6.7 'Auto' heights for block formatting context roots

10.7 Minimum and maximum heights: 'min-height' and 'max-height'
10.8 Line height calculations: the 'line-height' and 'vertical-align' properties
10.8.1 Leading and half-leading

Note: Several sections of this specification have been updated by other specifications. Please, see "Cascading Style Sheets (CSS) — The Official Definition" in the latest *CSS Snapshot* for a list of specifications and the sections they replace.

The CSS Working Group is also developing CSS level 2 revision 2 (CSS 2.2).

10.1 Definition of "containing block"

The position and size of an element's box(es) are sometimes calculated relative to a certain rectangle, called the *containing block* of the element. The containing block of an element is defined as follows:

1. The containing block in which the root element lives is a rectangle called the *initial containing block*. For continuous media, it has the dimensions of the viewport and is anchored at the canvas origin; it is the page area for paged media. The 'direction' property of the initial containing block is the same as for the root element.
2. For other elements, if the element's position is 'relative' or 'static', the containing block is formed by the content edge of the nearest block container ancestor box.
3. If the element has 'position: fixed', the containing block is established by the viewport in the case of continuous media or the page area in the case of paged media.
4. If the element has 'position: absolute', the containing block is established by the nearest ancestor with a 'position' of 'absolute', 'relative' or 'fixed', in the following way:
    1. In the case that the ancestor is an inline element, the containing block is the bounding box around the padding boxes of the first and the last inline boxes generated for that element. In CSS 2.1, if the inline element is split across multiple lines, the containing block is undefined.
    2. Otherwise, the containing block is formed by the padding edge of the ancestor.

    If there is no such ancestor, the containing block is the initial containing block.

In paged media, an absolutely positioned element is positioned relative to its containing block ignoring any page breaks (as if the document were continuous). The element may subsequently be broken over several pages.

For absolutely positioned content that resolves to a position on a page other than the page being laid out (the current page), or resolves to a position on the current page which has already been rendered for printing, printers may place the content

- on another location on the current page,
- on a subsequent page, or
- may omit it.

Note that a block-level element that is split over several pages may have a different width on each page and that there may be device-specific limits.

With no positioning, the containing blocks (C.B.) in the following document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<HTML>
   <HEAD>
      <TITLE>Illustration of containing blocks</TITLE>
   </HEAD>
   <BODY id="body">
      <DIV id="div1">
      <P id="p1">This is text in the first paragraph...</P>
      <P id="p2">This is text <EM id="em1"> in the
      <STRONG id="strong1">second</STRONG> paragraph.</EM></P>
      </DIV>
   </BODY>
</HTML>
```

are established as follows:

For box generated by	C.B. is established by
html	initial C.B. (UA-dependent)
body	html
div1	body
p1	div1

p2	div1
em1	p2
strong1	p2

If we position "div1":

```
#div1 { position: absolute; left: 50px; top: 50px }
```

its containing block is no longer "body"; it becomes the initial containing block (since there are no other positioned ancestor boxes).

If we position "em1" as well:

```
#div1 { position: absolute; left: 50px; top: 50px }
#em1  { position: absolute; left: 100px; top: 100px }
```

the table of containing blocks becomes:

For box generated by	C.B. is established by
html	initial C.B. (UA-dependent)
body	html
div1	initial C.B.
p1	div1
p2	div1
em1	div1
strong1	em1

By positioning "em1", its containing block becomes the nearest positioned ancestor box (i.e., that generated by "div1").

10.2 Content width: the 'width' property

'width'

Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements but non-replaced inline elements, table rows, and row groups
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	the percentage or 'auto' as specified or the absolute length

This property specifies the content width of boxes.

This property does not apply to non-replaced inline elements. The content width of a non-replaced inline element's boxes is that of the rendered content within them (*before* any relative offset of children). Recall that inline boxes flow into line boxes. The width of line boxes is given by the their containing block, but may be shorted by the presence of floats.

Values have the following meanings:

<length>
Specifies the width of the content area using a length unit.

<percentage>
Specifies a percentage width. The percentage is calculated with respect to the width of the generated box's containing block. If the containing block's width depends on this element's width, then the resulting layout is undefined in CSS 2.1.

Note: For absolutely positioned elements whose containing block is based on a block container element, the percentage is calculated with respect to the width of the *padding box* of that element. This is a change from CSS1, where the percentage width was always calculated with respect to the *content box* of the parent element.

auto
The width depends on the values of other properties. See the sections below.

Negative values for 'width' are illegal.

For example, the following rule fixes the content width of paragraphs at 100 pixels:

```
p { width: 100px }
```

10.3 Calculating widths and margins

The values of an element's 'width', 'margin-left', 'margin-right', 'left' and 'right' properties as used for layout depend on the type of box generated and on each other. (The value used for layout is sometimes referred to as the used value.) In principle, the values used are the same as the computed values, with 'auto' replaced by some suitable value, and percentages calculated based on the containing block, but there are exceptions. The following situations need to be distinguished:

1. inline, non-replaced elements
2. inline, replaced elements
3. block-level, non-replaced elements in normal flow
4. block-level, replaced elements in normal flow
5. floating, non-replaced elements
6. floating, replaced elements
7. absolutely positioned, non-replaced elements
8. absolutely positioned, replaced elements
9. 'inline-block', non-replaced elements in normal flow
10. 'inline-block', replaced elements in normal flow

For Points 1–6 and 9–10, the values of 'left' and 'right' in the case of relatively positioned elements are determined by the rules in section 9.4.3.

> ***Note.*** *The used value of 'width' calculated below is a tentative value, and may have to be calculated multiple times, depending on 'min-width' and 'max-width', see the section Minimum and maximum widths below.*

10.3.1 Inline, non-replaced elements

The 'width' property does not apply. A computed value of 'auto' for 'margin-left' or 'margin-right' becomes a used value of '0'.

10.3.2 Inline, replaced elements

A computed value of 'auto' for 'margin-left' or 'margin-right' becomes a used value of '0'.

If 'height' and 'width' both have computed values of 'auto' and the element also has an intrinsic width, then that intrinsic width is the used value of 'width'.

If 'height' and 'width' both have computed values of 'auto' and the element has no intrinsic width, but does have an intrinsic height and intrinsic ratio; or if 'width' has a computed value of 'auto', 'height' has some other computed value, and the element does have an intrinsic ratio; then the used value of 'width' is:

(used height) * (intrinsic ratio)

If 'height' and 'width' both have computed values of 'auto' and the element has an intrinsic ratio but no intrinsic height or width, then the used value of 'width' is undefined in CSS 2.1. However, it is suggested that, if the containing block's width does not itself depend on the replaced element's width, then the used value of 'width' is calculated from the constraint equation used for block-level, non-replaced elements in normal flow.

Otherwise, if 'width' has a computed value of 'auto', and the element has an intrinsic width, then that intrinsic width is the used value of 'width'.

Otherwise, if 'width' has a computed value of 'auto', but none of the conditions above are met, then the used value of 'width' becomes 300px. If 300px is too wide to fit the device, UAs should use the width of the largest rectangle that has a 2:1 ratio and fits the device instead.

10.3.3 Block-level, non-replaced elements in normal flow

The following constraints must hold among the used values of the other properties:

'margin-left' + 'border-left-width' + 'padding-left' + 'width' + 'padding-right' + 'border-right-width' + 'margin-right' = width of containing block

If 'width' is not 'auto' and 'border-left-width' + 'padding-left' + 'width' + 'padding-right' + 'border-right-width' (plus any of 'margin-left' or 'margin-right' that are not 'auto') is larger than the width of the containing block, then any 'auto' values for 'margin-left' or 'margin-right' are, for the following rules, treated as zero.

If all of the above have a computed value other than 'auto', the values are said to be "over-constrained" and one of the used values will have to be different from its computed value. If the 'direction' property of the containing block has the value 'ltr', the specified value of 'margin-right' is ignored and the value is calculated so as to make the equality true. If the value of 'direction' is 'rtl', this happens to 'margin-left' instead.

If there is exactly one value specified as 'auto', its used value follows from the equality.

If 'width' is set to 'auto', any other 'auto' values become '0' and 'width' follows from the resulting equality.

If both 'margin-left' and 'margin-right' are 'auto', their used values are equal. This horizontally centers the element with respect to the edges of the containing block.

10.3.4 Block-level, replaced elements in normal flow

The used value of 'width' is determined as for inline replaced elements. Then the rules for non-replaced block-level elements are applied to determine the margins.

10.3.5 Floating, non-replaced elements

If 'margin-left', or 'margin-right' are computed as 'auto', their used value is '0'.

If 'width' is computed as 'auto', the used value is the "shrink-to-fit" width.

Calculation of the shrink-to-fit width is similar to calculating the width of a table cell using the automatic table layout algorithm. Roughly: calculate the preferred width by formatting the content without breaking lines other than where explicit line breaks occur, and also calculate the preferred *minimum* width, e.g., by trying all possible line breaks. CSS 2.1 does not define the exact algorithm. Thirdly, find the *available width*: in this case, this is the width of the containing block minus the used values of 'margin-left', 'border-left-width', 'padding-left', 'padding-right', 'border-right-width', 'margin-right', and the widths of any relevant scroll bars.

Then the shrink-to-fit width is: min(max(preferred minimum width, available width), preferred width).

10.3.6 Floating, replaced elements

If 'margin-left' or 'margin-right' are computed as 'auto', their used value is '0'. The used value of 'width' is determined as for inline replaced elements.

10.3.7 Absolutely positioned, non-replaced elements

For the purposes of this section and the next, the term "static position" (of an element) refers, roughly, to the position an element would have had in the normal flow. More precisely:

- The *static-position containing block* is the containing block of a hypothetical box that would have been the first box of the element if its specified 'position' value had been 'static' and its specified 'float' had been 'none'. (Note that due to the rules in section 9.7 this hypothetical calculation might require also assuming a different computed value for 'display'.)
- The static position for 'left' is the distance from the left edge of the containing block to the left margin edge of a hypothetical box that would have been the first box of the element if its 'position' property had been 'static' and 'float' had been 'none'. The value is negative if the hypothetical box is to the left of the containing block.
- The static position for 'right' is the distance from the right edge of the containing block to the right margin edge of the same hypothetical box as above. The value is positive if the hypothetical box is to the left of the containing block's edge.

But rather than actually calculating the dimensions of that hypothetical box, user agents are free to make a guess at its probable position.

For the purposes of calculating the static position, the containing block of fixed positioned elements is the initial containing block instead of the viewport, and all scrollable boxes should be assumed to be scrolled to their origin.

The constraint that determines the used values for these elements is:

'left' + 'margin-left' + 'border-left-width' + 'padding-left' + 'width' + 'padding-right' + 'border-right-width' + 'margin-right' + 'right' = width of containing block

If all three of 'left', 'width', and 'right' are 'auto': First set any 'auto' values for 'margin-left' and 'margin-right' to 0. Then, if the 'direction' property of the element establishing the static-position containing block is 'ltr' set 'left' to the static position and apply rule number three below; otherwise, set 'right' to the static position and apply rule number one below.

If none of the three is 'auto': If both 'margin-left' and 'margin-right' are 'auto', solve the equation under the extra constraint that the two margins get equal values, unless this would make them negative, in which case when direction of the containing block is 'ltr' ('rtl'), set 'margin-left' ('margin-right') to zero and solve for 'margin-right' ('margin-left'). If one of 'margin-left' or 'margin-right' is 'auto', solve the equation for that value. If the values are over-constrained, ignore the value for 'left' (in case the 'direction' property of the containing block is 'rtl') or 'right' (in case 'direction' is 'ltr') and solve for that value.

Otherwise, set 'auto' values for 'margin-left' and 'margin-right' to 0, and pick the one of the following six rules that applies.

1. 'left' and 'width' are 'auto' and 'right' is not 'auto', then the width is shrink-to-fit. Then solve for 'left'
2. 'left' and 'right' are 'auto' and 'width' is not 'auto', then if the 'direction' property of the element establishing the static-position containing block is 'ltr' set 'left' to the static position, otherwise set 'right' to the static position. Then solve for 'left' (if 'direction is 'rtl') or 'right' (if 'direction' is 'ltr').
3. 'width' and 'right' are 'auto' and 'left' is not 'auto', then the width is shrink-to-fit . Then solve for 'right'
4. 'left' is 'auto', 'width' and 'right' are not 'auto', then solve for 'left'
5. 'width' is 'auto', 'left' and 'right' are not 'auto', then solve for 'width'
6. 'right' is 'auto', 'left' and 'width' are not 'auto', then solve for 'right'

Calculation of the shrink-to-fit width is similar to calculating the width of a table cell using the automatic table layout algorithm. Roughly: calculate the preferred width by formatting the content without breaking lines other than where explicit line breaks occur, and also calculate the preferred *minimum* width, e.g., by trying all possible line breaks. CSS 2.1 does not define the exact algorithm. Thirdly, calculate the *available width*: this is found by solving for 'width' after setting 'left' (in case 1) or 'right' (in case 3) to 0.

Then the shrink-to-fit width is: min(max(preferred minimum width, available width), preferred width).

10.3.8 Absolutely positioned, replaced elements

In this case, section 10.3.7 applies up through and including the constraint equation, but the rest of section 10.3.7 is replaced by the following rules:

1. The used value of 'width' is determined as for inline replaced elements. If 'margin-left' or 'margin-right' is specified as 'auto' its used value is determined by the rules below.

2. If both 'left' and 'right' have the value 'auto', then if the 'direction' property of the element establishing the static-position containing block is 'ltr', set 'left' to the static position; else if 'direction' is 'rtl', set 'right' to the static position.
3. If 'left' or 'right' are 'auto', replace any 'auto' on 'margin-left' or 'margin-right' with '0'.
4. If at this point both 'margin-left' and 'margin-right' are still 'auto', solve the equation under the extra constraint that the two margins must get equal values, unless this would make them negative, in which case when the direction of the containing block is 'ltr' ('rtl'), set 'margin-left' ('margin-right') to zero and solve for 'margin-right' ('margin-left').
5. If at this point there is an 'auto' left, solve the equation for that value.
6. If at this point the values are over-constrained, ignore the value for either 'left' (in case the 'direction' property of the containing block is 'rtl') or 'right' (in case 'direction' is 'ltr') and solve for that value.

10.3.9 'Inline-block', non-replaced elements in normal flow

If 'width' is 'auto', the used value is the shrink-to-fit width as for floating elements.

A computed value of 'auto' for 'margin-left' or 'margin-right' becomes a used value of '0'.

10.3.10 'Inline-block', replaced elements in normal flow

Exactly as inline replaced elements.

10.4 Minimum and maximum widths: 'min-width' and 'max-width'

'min-width'

Value:	<length> | <percentage> | inherit
Initial:	0
Applies to:	all elements but non-replaced inline elements, table rows, and row groups
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	the percentage as specified or the absolute length

'max-width'

Value:	<length> | <percentage> | none | inherit
Initial:	none
Applies to:	all elements but non-replaced inline elements, table rows, and row groups
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	the percentage as specified or the absolute length or 'none'

These two properties allow authors to constrain content widths to a certain range. Values have the following meanings:

<length>
Specifies a fixed minimum or maximum used width.

<percentage>
Specifies a percentage for determining the used value. The percentage is calculated with respect to the width of the generated box's containing block. If the containing block's width is negative, the used value is zero. If the containing block's width depends on this element's width, then the resulting layout is undefined in CSS 2.1.

none
(Only on 'max-width') No limit on the width of the box.

Negative values for 'min-width' and 'max-width' are illegal.

In CSS 2.1, the effect of 'min-width' and 'max-width' on tables, inline tables, table cells, table columns, and column groups is undefined.

The following algorithm describes how the two properties influence the used value of the 'width' property:

1. The tentative used width is calculated (without 'min-width' and 'max-width') following the rules under "Calculating widths and margins" above.
2. If the tentative used width is greater than 'max-width', the rules above are applied again, but this time using the computed value of 'max-width' as the computed value for 'width'.

3. If the resulting width is smaller than 'min-width', the rules above are applied again, but this time using the value of 'min-width' as the computed value for 'width'.

These steps do not affect the real computed values of the above properties.

However, for replaced elements with an intrinsic ratio and both 'width' and 'height' specified as 'auto', the algorithm is as follows:

Select from the table the resolved height and width values for the appropriate constraint violation. Take the max-width and max-height as max(min, max) so that min ⩽ max holds true. In this table w and h stand for the results of the width and height computations ignoring the 'min-width', 'min-height', 'max-width' and 'max-height' properties. Normally these are the intrinsic width and height, but they may not be in the case of replaced elements with intrinsic ratios.

Note: In cases where an explicit width or height is set and the other dimension is auto, applying a minimum or maximum constraint on the auto side can cause an over-constrained situation. The spec is clear in the behavior but it might not be what the author expects. The CSS3 object-fit property can be used to obtain different results in this situation.

Constraint Violation	Resolved Width	Resolved Height
none	w	h
w > max-width	max-width	max(max-width * h/w, min-height)
w < min-width	min-width	min(min-width * h/w, max-height)
h > max-height	max(max-height * w/h, min-width)	max-height
h < min-height	min(min-height * w/h, max-width)	min-height
(w > max-width) and (h > max-height), where (max-width/w ⩽ max-height/h)	max-width	max(min-height, max-width * h/w)

Constraint Violation	Resolved Width	Resolved Height
(w > max-width) and (h > max-height), where (max-width/w > max-height/h)	max(min-width, max-height * w/h)	max-height
(w < min-width) and (h < min-height), where (min-width/w ⩽ min-height/h)	min(max-width, min-height * w/h)	min-height
(w < min-width) and (h < min-height), where (min-width/w > min-height/h)	min-width	min(max-height, min-width * h/w)
(w < min-width) and (h > max-height)	min-width	max-height
(w > max-width) and (h < min-height)	max-width	min-height

Then apply the rules under "Calculating widths and margins" above, as if 'width' were computed as this value.

10.5 Content height: the 'height' property

'height'

Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements but non-replaced inline elements, table columns, and column groups
Inherited:	no
Percentages:	see prose
Media:	visual
Computed value:	the percentage or 'auto' (see prose under <percentage>) or the absolute length

This property specifies the content height of boxes.

This property does not apply to non-replaced inline elements. See the section on computing heights and margins for non-replaced inline elements for the rules used instead.

Values have the following meanings:

<length>

Specifies the height of the content area using a length value.

<percentage>

Specifies a percentage height. The percentage is calculated with respect to the height of the generated box's containing block. If the height of the containing block is not specified explicitly (i.e., it depends on content height), and this element is not absolutely positioned, the value computes to 'auto'. A percentage height on the root element is relative to the initial containing block. Note: For absolutely positioned elements whose containing block is based on a block-level element, the percentage is calculated with respect to the height of the *padding box* of that element. This is a change from CSS1, where the percentage was always calculated with respect to the *content box* of the parent element.

auto

The height depends on the values of other properties. See the prose below.

Note that the height of the containing block of an absolutely positioned element is independent of the size of the element itself, and thus a percentage height on such an element can always be resolved. However, it may be that the height is not known until elements that come later in the document have been processed.

Negative values for 'height' are illegal.

For example, the following rule sets the content height of paragraphs to 100 pixels:

```
p { height: 100px }
```

Paragraphs of which the height of the contents exceeds 100 pixels will overflow according to the 'overflow' property.

10.6 Calculating heights and margins

For calculating the values of 'top', 'margin-top', 'height', 'margin-bottom', and 'bottom' a distinction must be made between various kinds of boxes:

1. inline, non-replaced elements
2. inline, replaced elements
3. block-level, non-replaced elements in normal flow
4. block-level, replaced elements in normal flow

5. floating, non-replaced elements
6. floating, replaced elements
7. absolutely positioned, non-replaced elements
8. absolutely positioned, replaced elements
9. 'inline-block', non-replaced elements in normal flow
10. 'inline-block', replaced elements in normal flow

For Points 1–6 and 9–10, the used values of 'top' and 'bottom' are determined by the rules in section 9.4.3.

Note: these rules apply to the root element just as to any other element.

Note. *The used value of 'height' calculated below is a tentative value, and may have to be calculated multiple times, depending on 'min-height' and 'max-height', see the section Minimum and maximum heights below.*

10.6.1 Inline, non-replaced elements

The 'height' property does not apply. The height of the content area should be based on the font, but this specification does not specify how. A UA may, e.g., use the em-box or the maximum ascender and descender of the font. (The latter would ensure that glyphs with parts above or below the em-box still fall within the content area, but leads to differently sized boxes for different fonts; the former would ensure authors can control background styling relative to the 'line-height', but leads to glyphs painting outside their content area.)

Note: level 3 of CSS will probably include a property to select which measure of the font is used for the content height.

The vertical padding, border and margin of an inline, non-replaced box start at the top and bottom of the content area, and has nothing to do with the 'line-height'. But only the 'line-height' is used when calculating the height of the line box.

If more than one font is used (this could happen when glyphs are found in different fonts), the height of the content area is not defined by this specification. However, we suggest that the height is chosen such that the content area is just high enough for either (1) the em-boxes, or (2) the maximum ascenders and descenders, of *all* the fonts in the element. Note that this may be larger than any of the font sizes involved, depending on the baseline alignment of the fonts.

10.6.2 Inline replaced elements, block-level replaced elements in normal flow, 'inline-block' replaced elements in normal flow and floating replaced elements

If 'margin-top', or 'margin-bottom' are 'auto', their used value is 0.

If 'height' and 'width' both have computed values of 'auto' and the element also has an intrinsic height, then that intrinsic height is the used value of 'height'.

Otherwise, if 'height' has a computed value of 'auto', and the element has an intrinsic ratio then the used value of 'height' is:

(used width) / (intrinsic ratio)

Otherwise, if 'height' has a computed value of 'auto', and the element has an intrinsic height, then that intrinsic height is the used value of 'height'.

Otherwise, if 'height' has a computed value of 'auto', but none of the conditions above are met, then the used value of 'height' must be set to the height of the largest rectangle that has a 2:1 ratio, has a height not greater than 150px, and has a width not greater than the device width.

10.6.3 Block-level non-replaced elements in normal flow when 'overflow' computes to 'visible'

This section also applies to block-level non-replaced elements in normal flow when 'overflow' does not compute to 'visible' but has been propagated to the viewport.

If 'margin-top', or 'margin-bottom' are 'auto', their used value is 0. If 'height' is 'auto', the height depends on whether the element has any block-level children and whether it has padding or borders:

The element's height is the distance from its top content edge to the first applicable of the following:

1. the bottom edge of the last line box, if the box establishes a inline formatting context with one or more lines
2. the bottom edge of the bottom (possibly collapsed) margin of its last in-flow child, if the child's bottom margin does not collapse with the element's bottom margin
3. the bottom border edge of the last in-flow child whose top margin doesn't collapse with the element's bottom margin
4. zero, otherwise

Only children in the normal flow are taken into account (i.e., floating boxes and absolutely positioned boxes are ignored, and relatively positioned boxes are considered without their offset). Note that the child box may be an anonymous block box.

10.6.4 Absolutely positioned, non-replaced elements

For the purposes of this section and the next, the term "static position" (of an element) refers, roughly, to the position an element would have had in the normal flow. More precisely, the static position for 'top' is the distance from the top edge of the containing block to the top margin edge of a hypothetical box that would have been the first box of the element if its specified 'position' value had been 'static' and its specified 'float' had been 'none' and its specified 'clear' had been 'none'. (Note that due to the rules in section 9.7 this might require also assuming a different computed value for 'display'.) The value is negative if the hypothetical box is above the containing block.

But rather than actually calculating the dimensions of that hypothetical box, user agents are free to make a guess at its probable position.

For the purposes of calculating the static position, the containing block of fixed positioned elements is the initial containing block instead of the viewport.

For absolutely positioned elements, the used values of the vertical dimensions must satisfy this constraint:

'top' + 'margin-top' + 'border-top-width' + 'padding-top' + 'height' + 'padding-bottom' + 'border-bottom-width' + 'margin-bottom' + 'bottom' = height of containing block

If all three of 'top', 'height', and 'bottom' are auto, set 'top' to the static position and apply rule number three below.

If none of the three are 'auto': If both 'margin-top' and 'margin-bottom' are 'auto', solve the equation under the extra constraint that the two margins get equal values. If one of 'margin-top' or 'margin-bottom' is 'auto', solve the equation for that value. If the values are over-constrained, ignore the value for 'bottom' and solve for that value.

Otherwise, pick the one of the following six rules that applies.

1. 'top' and 'height' are 'auto' and 'bottom' is not 'auto', then the height is based on the content per 10.6.7, set 'auto' values for 'margin-top' and 'margin-bottom' to 0, and solve for 'top'
2. 'top' and 'bottom' are 'auto' and 'height' is not 'auto', then set 'top' to the static position, set 'auto' values for 'margin-top' and 'margin-bottom' to 0, and solve for 'bottom'

3. 'height' and 'bottom' are 'auto' and 'top' is not 'auto', then the height is based on the content per 10.6.7, set 'auto' values for 'margin-top' and 'margin-bottom' to 0, and solve for 'bottom'
4. 'top' is 'auto', 'height' and 'bottom' are not 'auto', then set 'auto' values for 'margin-top' and 'margin-bottom' to 0, and solve for 'top'
5. 'height' is 'auto', 'top' and 'bottom' are not 'auto', then 'auto' values for 'margin-top' and 'margin-bottom' are set to 0 and solve for 'height'
6. 'bottom' is 'auto', 'top' and 'height' are not 'auto', then set 'auto' values for 'margin-top' and 'margin-bottom' to 0 and solve for 'bottom'

10.6.5 Absolutely positioned, replaced elements

This situation is similar to the previous one, except that the element has an intrinsic height. The sequence of substitutions is now:

1. The used value of 'height' is determined as for inline replaced elements. If 'margin-top' or 'margin-bottom' is specified as 'auto' its used value is determined by the rules below.
2. If both 'top' and 'bottom' have the value 'auto', replace 'top' with the element's static position.
3. If 'bottom' is 'auto', replace any 'auto' on 'margin-top' or 'margin-bottom' with '0'.
4. If at this point both 'margin-top' and 'margin-bottom' are still 'auto', solve the equation under the extra constraint that the two margins must get equal values.
5. If at this point there is only one 'auto' left, solve the equation for that value.
6. If at this point the values are over-constrained, ignore the value for 'bottom' and solve for that value.

10.6.6 Complicated cases

This section applies to:

- Block-level, non-replaced elements in normal flow when 'overflow' does not compute to 'visible' (except if the 'overflow' property's value has been propagated to the viewport).

- 'Inline-block', non-replaced elements.
- Floating, non-replaced elements.

If 'margin-top', or 'margin-bottom' are 'auto', their used value is 0. If 'height' is 'auto', the height depends on the element's descendants per 10.6.7.

For 'inline-block' elements, the margin box is used when calculating the height of the line box.

10.6.7 'Auto' heights for block formatting context roots

In certain cases (see, e.g., sections 10.6.4 and 10.6.6 above), the height of an element that establishes a block formatting context is computed as follows:

If it only has inline-level children, the height is the distance between the top of the topmost line box and the bottom of the bottommost line box.

If it has block-level children, the height is the distance between the top margin-edge of the topmost block-level child box and the bottom margin-edge of the bottommost block-level child box.

Absolutely positioned children are ignored, and relatively positioned boxes are considered without their offset. Note that the child box may be an anonymous block box.

In addition, if the element has any floating descendants whose bottom margin edge is below the element's bottom content edge, then the height is increased to include those edges. Only floats that participate in this block formatting context are taken into account, e.g., floats inside absolutely positioned descendants or other floats are not.

10.7 Minimum and maximum heights: 'min-height' and 'max-height'

It is sometimes useful to constrain the height of elements to a certain range. Two properties offer this functionality:

'min-height'

Value:	<length> | <percentage> | inherit
Initial:	0
Applies to:	all elements but non-replaced inline elements, table columns, and column groups

Inherited: no
Percentages: see prose
Media: visual
Computed value: the percentage as specified or the absolute length

'max-height'

Value: <length> | <percentage> | none | inherit
Initial: none
Applies to: all elements but non-replaced inline elements, table columns, and column groups
Inherited: no
Percentages: see prose
Media: visual
Computed value: the percentage as specified or the absolute length or 'none'

These two properties allow authors to constrain box heights to a certain range. Values have the following meanings:

<length>
Specifies a fixed minimum or maximum computed height.

<percentage>
Specifies a percentage for determining the used value. The percentage is calculated with respect to the height of the generated box's containing block. If the height of the containing block is not specified explicitly (i.e., it depends on content height), and this element is not absolutely positioned, the percentage value is treated as '0' (for 'min-height') or 'none' (for 'max-height').

none
(Only on 'max-height') No limit on the height of the box.

Negative values for 'min-height' and 'max-height' are illegal.

In CSS 2.1, the effect of 'min-height' and 'max-height' on tables, inline tables, table cells, table rows, and row groups is undefined.

The following algorithm describes how the two properties influence the used value of the 'height' property:

1. The tentative used height is calculated (without 'min-height' and 'max-height') following the rules under "Calculating heights and margins" above.
2. If this tentative height is greater than 'max-height', the rules above are applied again, but this time using the value of 'max-height' as the computed value for 'height'.
3. If the resulting height is smaller than 'min-height', the rules above are applied again, but this time using the value of 'min-height' as the computed value for 'height'.

These steps do not affect the real computed values of the above properties. The change of used 'height' has no effect on margin collapsing except as specifically required by rules for 'min-height' or 'max-height' in "Collapsing margins" (8.3.1).

However, for replaced elements with both 'width' and 'height' computed as 'auto', use the algorithm under Minimum and maximum widths above to find the used width and height. Then apply the rules under "Computing heights and margins" above, using the resulting width and height as if they were the computed values.

10.8 Line height calculations: the 'line-height' and 'vertical-align' properties

As described in the section on inline formatting contexts, user agents flow inline-level boxes into a vertical stack of line boxes. The height of a line box is determined as follows:

1. The height of each inline-level box in the line box is calculated. For replaced elements, inline-block elements, and inline-table elements, this is the height of their margin box; for inline boxes, this is their 'line-height'. (See "Calculating heights and margins" and the height of inline boxes in "Leading and half-leading".)
2. The inline-level boxes are aligned vertically according to their 'vertical-align' property. In case they are aligned 'top' or 'bottom', they must be aligned so as to minimize the line box height. If such boxes are tall enough, there are multiple solutions and CSS 2.1 does not define the position of the line box's baseline (i.e., the position of the strut, see below).
3. The line box height is the distance between the uppermost box top and the lowermost box bottom. (This includes the strut, as explained under 'line-height' below.)

Empty inline elements generate empty inline boxes, but these boxes still have margins, padding, borders and a line height, and thus influence these calculations just like elements with content.

10.8.1 Leading and half-leading

CSS assumes that every font has font metrics that specify a characteristic height above the baseline and a depth below it. In this section we use A to mean that height (for a given font at a given size) and D the depth. We also define AD = A + D, the distance from the top to the bottom. (See the note below for how to find A and D for TrueType and OpenType fonts.) Note that these are metrics of the font as a whole and need not correspond to the ascender and descender of any individual glyph.

User agent must align the glyphs in a non-replaced inline box to each other by their relevant baselines. Then, for each glyph, determine the A and D. Note that glyphs in a single element may come from different fonts and thus need not all have the same A and D. If the inline box contains no glyphs at all, it is considered to contain a strut (an invisible glyph of zero width) with the A and D of the element's first available font.

Still for each glyph, determine the leading L to add, where L = 'line-height' − AD. Half the leading is added above A and the other half below D, giving the glyph and its leading a total height above the baseline of A' = A + L/2 and a total depth of D' = D + L/2.

Note. L may be negative.

The height of the inline box encloses all glyphs and their half-leading on each side and is thus exactly 'line-height'. Boxes of child elements do not influence this height.

Although margins, borders, and padding of non-replaced elements do not enter into the line box calculation, they are still rendered around inline boxes. This means that if the height specified by 'line-height' is less than the content height of contained boxes, backgrounds and colors of padding and borders may "bleed" into adjoining line boxes. User agents should render the boxes in document order. This will cause the borders on subsequent lines to paint over the borders and text of previous lines.

Note. *CSS 2.1 does not define what the content area of an inline box is (see 10.6.1 above) and thus different UAs may draw the backgrounds and borders in different places.*

Note. *It is recommended that implementations that use OpenType or TrueType fonts use the metrics "sTypoAscender" and "sTypoDescender" from the font's OS/2 table for A and D (after scaling to the current element's font size). In the absence of these metrics, the "Ascent" and "Descent" metrics from the HHEA table should be used.*

'line-height'

Value:	normal | <number> | <length> | <percentage> | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	refer to the font size of the element itself
Media:	visual
Computed value:	for <length> and <percentage> the absolute value; otherwise as specified

On a block container element whose content is composed of inline-level elements, 'line-height' specifies the *minimal* height of line boxes within the element. The minimum height consists of a minimum height above the baseline and a minimum depth below it, exactly as if each line box starts with a zero-width inline box with the element's font and line height properties. We call that imaginary box a "strut." (The name is inspired by TeX.).

The height and depth of the font above and below the baseline are assumed to be metrics that are contained in the font. (For more details, see CSS level 3.)

On a non-replaced inline element, 'line-height' specifies the height that is used in the calculation of the line box height.

Values for this property have the following meanings:

normal
: Tells user agents to set the used value to a "reasonable" value based on the font of the element. The value has the same meaning as <number>. We recommend a used value for 'normal' between 1.0 to 1.2. The computed value is 'normal'.

<length>
: The specified length is used in the calculation of the line box height. Negative values are illegal.

<number>
: The used value of the property is this number multiplied by the element's font size. Negative values are illegal. The computed value is the same as the specified value.

<percentage>
: The computed value of the property is this percentage multiplied by the element's computed font size. Negative values are illegal.

The three rules in the example below have the same resultant line height:

```
div { line-height: 1.2; font-size: 10pt }     /* number */
div { line-height: 1.2em; font-size: 10pt }   /* length */
div { line-height: 120%; font-size: 10pt }    /* percentage */
```

When an element contains text that is rendered in more than one font, user agents may determine the 'normal' 'line-height' value according to the largest font size.

Note. *When there is only one value of 'line-height' for all inline boxes in a block container box and they are all in the same font (and there are no replaced elements, inline-block elements, etc.), the above will ensure that baselines of successive lines are exactly 'line-height' apart. This is important when columns of text in different fonts have to be aligned, for example in a table.*

'vertical-align'

Value:	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit
Initial:	baseline
Applies to:	inline-level and 'table-cell' elements
Inherited:	no
Percentages:	refer to the 'line-height' of the element itself
Media:	visual
Computed value:	for <percentage> and <length> the absolute length, otherwise as specified

This property affects the vertical positioning inside a line box of the boxes generated by an inline-level element.

Note. *Values of this property have different meanings in the context of tables. Please consult the section on table height algorithms for details.*

The following values only have meaning with respect to a parent inline element, or to the strut of a parent block container element.

In the following definitions, for inline non-replaced elements, the box used for alignment is the box whose height is the 'line-height' (containing the box's glyphs and the half-leading on each side, see above). For all other elements, the box used for

alignment is the margin box.

baseline
: Align the baseline of the box with the baseline of the parent box. If the box does not have a baseline, align the bottom margin edge with the parent's baseline.

middle
: Align the vertical midpoint of the box with the baseline of the parent box plus half the x-height of the parent.

sub
: Lower the baseline of the box to the proper position for subscripts of the parent's box. (This value has no effect on the font size of the element's text.)

super
: Raise the baseline of the box to the proper position for superscripts of the parent's box. (This value has no effect on the font size of the element's text.)

text-top
: Align the top of the box with the top of the parent's content area (see 10.6.1).

text-bottom
: Align the bottom of the box with the bottom of the parent's content area (see 10.6.1).

<percentage>
: Raise (positive value) or lower (negative value) the box by this distance (a percentage of the 'line-height' value). The value '0%' means the same as 'baseline'.

<length>
: Raise (positive value) or lower (negative value) the box by this distance. The value '0cm' means the same as 'baseline'.

The following values align the element relative to the line box. Since the element may have children aligned relative to it (which in turn may have descendants aligned relative to them), these values use the bounds of the aligned subtree. The *aligned subtree* of an inline element contains that element and the aligned subtrees of all children inline elements whose computed 'vertical-align' value is not 'top' or 'bottom'. The top of the aligned subtree is the highest of the tops of the boxes in the subtree, and the bottom is analogous.

top

Align the top of the aligned subtree with the top of the line box.

bottom

Align the bottom of the aligned subtree with the bottom of the line box.

The baseline of an 'inline-table' is the baseline of the first row of the table.

The baseline of an 'inline-block' is the baseline of its last line box in the normal flow, unless it has either no in-flow line boxes or if its 'overflow' property has a computed value other than 'visible', in which case the baseline is the bottom margin edge.