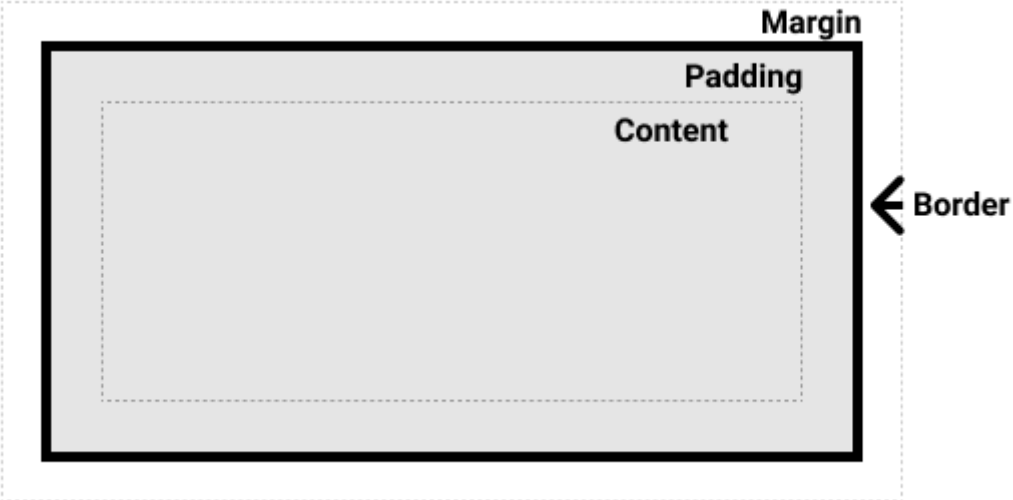
Layout and the containing block

The size and position of an element are often impacted by its containing block. Most often, the containing block is the content area of an element's nearest block-level ancestor, but this is not always the case. In this article, we examine the factors that determine an element's containing block.

When a user agent (such as your browser) lays out a document, it generates a box for every element. Each box is divided into four areas:

1. Content area
2. Padding area
3. Border area
4. Margin area



Many developers believe that the containing block of an element is always the content area of its parent, but that isn't necessarily true. Let's investigate the factors that determine what an element's containing block is.

Effects of the containing block

Before learning what determines the containing block of an element, it's useful to know why it matters in the first place.

The size and position of an element are often impacted by its containing block. Percentage values that are applied to the width , height , padding , margin , and offset properties of an absolutely positioned element (i.e., which has its position set to absolute or fixed) are computed from the element's containing block.

Identifying the containing block

The process for identifying the containing block depends entirely on the value of the element's position property:

1. If the position property is static, relative, or sticky, the containing block is formed by the edge of the *content box* of the nearest ancestor element that is either a block container (such as an inline-block, block, or list-item element) or establishes a formatting context (such as a table container, flex container, grid container, or the block container itself).
2. If the position property is absolute, the containing block is formed by the edge of the *padding box* of the nearest ancestor element that has a position value other than static (fixed, absolute, relative, or sticky).
3. If the position property is 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 position property is absolute or fixed, the containing block may also be formed by the edge of the *padding box* of the nearest ancestor element that has the following:
    i. A transform or perspective value other than none
    ii. A will-change value of transform or perspective
    iii. A filter value other than none or a will-change value of filter (only works on Firefox).
    iv. A contain value of paint (e.g. contain: paint;)
    v. A backdrop-filter other than none (e.g. backdrop-filter: blur(10px);)

> Note: The containing block in which the root element (<html>) resides is a rectangle called the initial containing block. It has the dimensions of the viewport (for continuous media) or the page area (for paged media).

Calculating percentage values from the containing block

As noted above, when certain properties are given a percentage value, the computed value depends on the element's containing block. The properties that work this way are box model properties and offset properties:

1. The height, top, and bottom properties compute percentage values from the height of the containing block.
2. The width, left, right, padding, and margin properties compute percentage values from the width of the containing block.

Some examples

The HTML code for all our examples is:

```
<body>
  <section>
    <p>This is a paragraph!</p>
  </section>
</body>
```

Only the CSS is altered in each instance below.

Example 1

In this example, the paragraph is statically positioned, so its containing block is `<section>` because it's the nearest ancestor that is a block container.

```
body {
  background: beige;
}

section {
  display: block;
  width: 400px;
  height: 160px;
  background: lightgray;
}

p {
  width: 50%;   /* == 400px * .5 = 200px */
  height: 25%;  /* == 160px * .25 = 40px */
  margin: 5%;   /* == 400px * .05 = 20px */
  padding: 5%;  /* == 400px * .05 = 20px */
  background: cyan;
}
```

Example 2

In this example, the paragraph's containing block is the `<body>` element, because `<section>` is not a block container (because of `display: inline`) and doesn't establish a formatting context.

```
body {
  background: beige;
}

section {
  display: inline;
  background: lightgray;
}

p {
  width: 50%;     /* == half the body's width */
  height: 200px;  /* Note: a percentage would be 0 */
  background: cyan;
}
```

Example 3

In this example, the paragraph's containing block is `<section>` because the latter's `position` is `absolute` . The paragraph's percentage values are affected by the `padding` of its containing block, though if the containing block's `box-sizing` value were `border-box` this would not be the case.

```
body {
  background: beige;
}

section {
  position: absolute;
  left: 30px;
  top: 30px;
  width: 400px;
  height: 160px;
  padding: 30px 20px;
  background: lightgray;
}

p {
  position: absolute;
  width: 50%;   /* == (400px + 20px + 20px) * .5 = 220px */
  height: 25%;  /* == (160px + 30px + 30px) * .25 = 55px */
  margin: 5%;   /* == (400px + 20px + 20px) * .05 = 22px */
  padding: 5%;  /* == (400px + 20px + 20px) * .05 = 22px */
  background: cyan;
}
```

Example 4

In this example, the paragraph's `position` is `fixed`, so its containing block is the initial containing block (on screens, the viewport). Thus, the paragraph's dimensions change based on the size of the browser window.

```
body {
  background: beige;
}

section {
  width: 400px;
  height: 480px;
  margin: 30px;
  padding: 15px;
  background: lightgray;
}

p {
  position: fixed;
  width: 50%;   /* == (50vw - (width of vertical scrollbar)) */
  height: 50%;  /* == (50vh - (height of horizontal scrollbar)) */
  margin: 5%;   /* == (5vw - (width of vertical scrollbar)) */
  padding: 5%;  /* == (5vw - (width of vertical scrollbar)) */
  background: cyan;
}
```

Example 5

In this example, the paragraph's `position` is `absolute`, so its containing block is `<section>`, which is the nearest ancestor with a `transform` property that isn't `none`.

```
body {
  background: beige;
}

section {
  transform: rotate(0deg);
  width: 400px;
  height: 160px;
  background: lightgray;
}

p {
  position: absolute;
  left: 80px;
  top: 30px;
  width: 50%;   /* == 200px */
  height: 25%;  /* == 40px */
  margin: 5%;   /* == 20px */
  padding: 5%;  /* == 20px */
  background: cyan;
}
```

See also

- The `all` property resets all CSS declarations to a given known state
- CSS key concepts:
  - CSS syntax
  - At-rules
  - Comments
  - Specificity
  - Inheritance
  - Box model
  - Layout modes
  - Visual formatting models
  - Margin collapsing
  - Values
    - Initial values
    - Computed values
    - Used values
    - Actual values
  - Value definition syntax
  - Shorthand properties
  - Replaced elements

Last modified: Sep 28, 2022, by MDN contributors