The stacking context

The **stacking context** is a three-dimensional conceptualization of HTML elements along an imaginary z-axis relative to the user, who is assumed to be facing the viewport or the webpage. HTML elements occupy this space in priority order based on element attributes.

The stacking context

In the previous part of this article, Using z-index, the rendering order of certain elements is influenced by their `z-index` value. This occurs because these elements have special properties which cause them to form a *stacking context*.

A stacking context is formed, anywhere in the document, by any element in the following scenarios:

- Root element of the document (`<html>`).
- Element with a `position` value `absolute` or `relative` and `z-index` value other than `auto`.
- Element with a `position` value `fixed` or `sticky` (sticky for all mobile browsers, but not older desktop).
- Element that is a child of a flex container, with `z-index` value other than `auto`.
- Element that is a child of a `grid` container, with `z-index` value other than `auto`.
- Element with a `opacity` value less than `1` (See the specification for opacity).
- Element with a `mix-blend-mode` value other than `normal`.
- Element with any of the following properties with value other than `none`:
  - `transform`
  - `filter`
  - `perspective`
  - `clip-path`
  - `mask` / `mask-image` / `mask-border`
- Element with a `isolation` value `isolate`.

- Element with a `-webkit-overflow-scrolling` value `touch`.
- Element with a `will-change` value specifying any property that would create a stacking context on non-initial value (see this post).

- Element with a `contain` value of `layout`, or `paint`, or a composite value that includes either of them (i.e. `contain: strict`, `contain: content`).

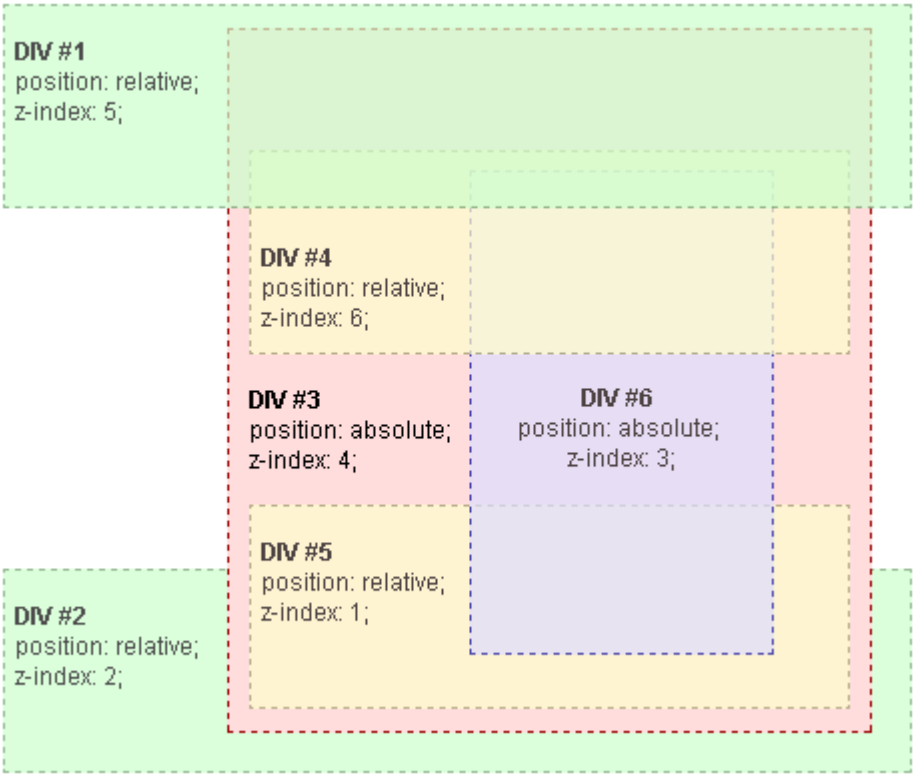
Within a stacking context, child elements are stacked according to the same rules previously explained. Importantly, the `z-index` values of its child stacking contexts only have meaning in this parent. Stacking contexts are treated atomically as a single unit in the parent stacking context.

In summary:

- Stacking contexts can be contained in other stacking contexts, and together create a hierarchy of stacking contexts.
- Each stacking context is completely independent of its siblings: only descendant elements are considered when stacking is processed.
- Each stacking context is self-contained: after the element's contents are stacked, the whole element is considered in the stacking order of the parent stacking context.

> **Note:** The hierarchy of stacking contexts is a subset of the hierarchy of HTML elements because only certain elements create stacking contexts. We can say that elements that do not create their own stacking contexts are *assimilated* by the parent stacking context.

The example



In this example, every positioned element creates its own stacking context, because of their positioning and `z-index` values. The hierarchy of stacking contexts is organized as follows:

- Root
  - DIV #1
  - DIV #2
  - DIV #3
    - DIV #4
    - DIV #5
    - DIV #6

It is important to note that DIV #4, DIV #5 and DIV #6 are children of DIV #3, so stacking of those elements is completely resolved within DIV#3. Once stacking and rendering within DIV #3 is completed, the whole DIV #3 element is passed for stacking in the root element with respect to its sibling's DIV.

Notes:

- DIV #4 is rendered under DIV #1 because DIV #1's z-index (5) is valid within the stacking context of the root element, while DIV #4's z-index (6) is valid within the

  stacking context of DIV #3. So, DIV #4 is under DIV #1, because DIV #4 belongs to DIV #3, which has a lower z-index value.
- For the same reason DIV #2 (z-index 2) is rendered under DIV#5 (z-index 1) because DIV #5 belongs to DIV #3, which has an higher z-index value.
- DIV #3's z-index is 4, but this value is independent from z-index of DIV #4, DIV #5 and DIV #6, because it belongs to a different stacking context.
- An easy way to figure out the *rendering order* of stacked elements along the Z axis is to think of it as a "version number" of sorts, where child elements are minor version numbers underneath their parent's major version numbers. This way we can easily see how an element with a z-index of 1 (DIV #5) is stacked above an element with a z-index of 2 (DIV #2), and how an element with a z-index of 6 (DIV #4) is stacked below an element with a z-index of 5 (DIV #1). In our example (sorted according to the final rendering order):
  - Root
    - DIV #2 - z-index is 2
    - DIV #3 - z-index is 4
      - DIV #5 - z-index is 1, stacked under an element with a z-index of 4, which results in a rendering order of 4.1
      - DIV #6 - z-index is 3, stacked under an element with a z-index of 4, which results in a rendering order of 4.3
      - DIV #4 - z-index is 6, stacked under an element with a z-index of 4, which results in a rendering order of 4.6
    - DIV #1 - z-index is 5

Example

HTML

```
<div id="div1">
  <h1>Division Element #1</h1>
  <code>position: relative;<br/>
```

```
  z-index: 5;</code>
</div>

<div id="div2">
  <h1>Division Element #2</h1>
  <code>position: relative;<br/>
  z-index: 2;</code>
</div>

<div id="div3">
  <div id="div4">
    <h1>Division Element #4</h1>
    <code>position: relative;<br/>
    z-index: 6;</code>
  </div>

  <h1>Division Element #3</h1>
  <code>position: absolute;<br/>
  z-index: 4;</code>

  <div id="div5">
    <h1>Division Element #5</h1>
    <code>position: relative;<br/>
    z-index: 1;</code>
  </div>

  <div id="div6">
    <h1>Division Element #6</h1>
    <code>position: absolute;<br/>
    z-index: 3;</code>
  </div>
</div>
```

CSS

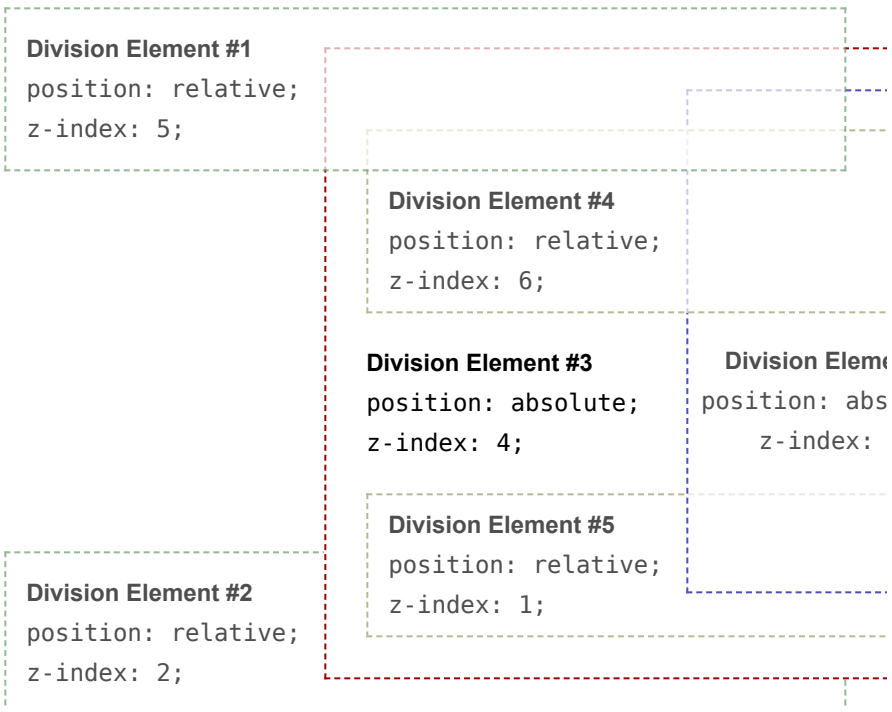
```
* {
  margin: 0;
}
html {
  padding: 20px;
  font: 12px/20px Arial, sans-serif;
}
```

```
div {
  opacity: 0.7;
  position: relative;
}
h1 {
  font: inherit;
  font-weight: bold;
}
#div1,
#div2 {
  border: 1px dashed #696;
  padding: 10px;
  background-color: #cfc;
}
#div1 {
  z-index: 5;
  margin-bottom: 190px;
}
#div2 {
  z-index: 2;
}
#div3 {
  z-index: 4;
  opacity: 1;
  position: absolute;
  top: 40px;
  left: 180px;
  width: 330px;
  border: 1px dashed #900;
  background-color: #fdd;
  padding: 40px 20px 20px;
}
#div4,
#div5 {
  border: 1px dashed #996;
  background-color: #ffc;
}
#div4 {
  z-index: 6;
  margin-bottom: 15px;
  padding: 25px 10px 5px;
}
#div5 {
```

```
#div5 {
  z-index: 1;
  margin-top: 15px;
  padding: 5px 10px;
}

#div6 {
  z-index: 3;
  position: absolute;
  top: 20px;
  left: 180px;
  width: 150px;
  height: 125px;
  border: 1px dashed #009;
  padding-top: 125px;
  background-color: #ddf;
  text-align: center;
}
```

Result



See also

- Stacking without the z-index property: The stacking rules that apply when `z-index` is not