



🏠 → [Browser: Document, Events, Interfaces](#) → [Document](#)

📅 29th January 2020

Coordinates

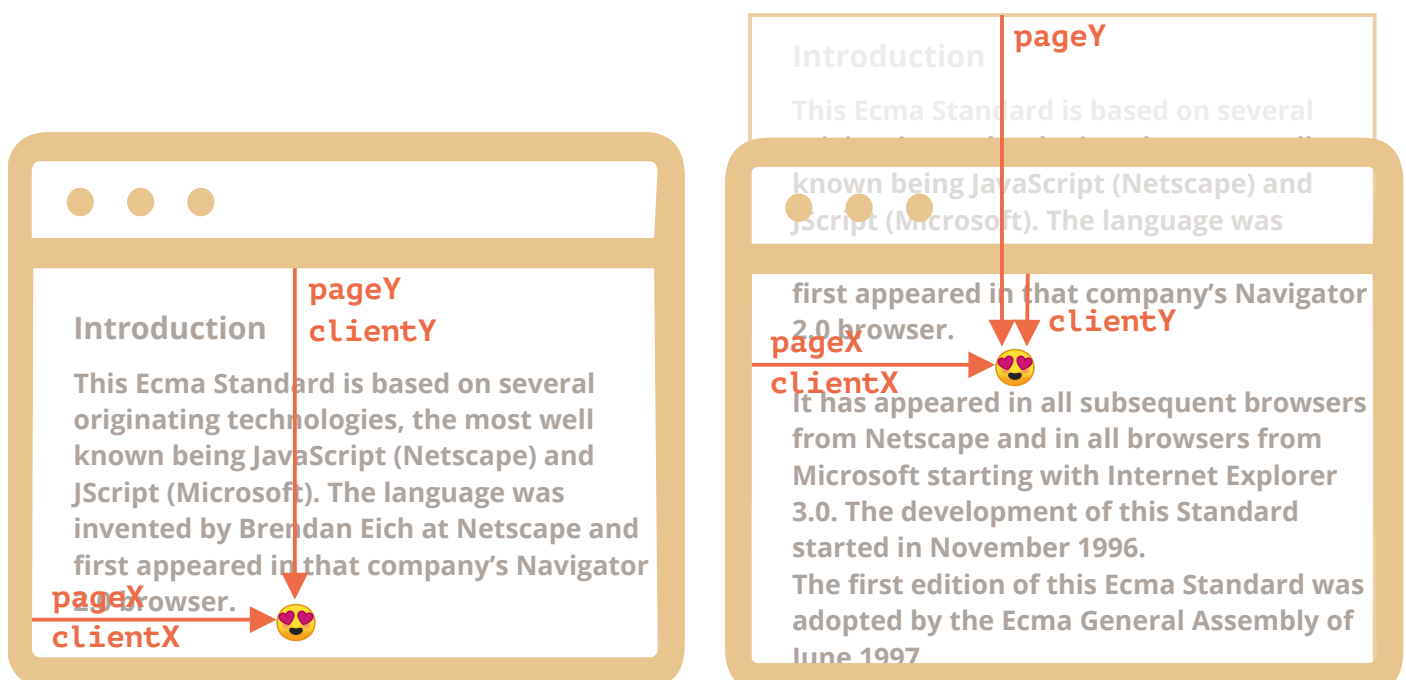
To move elements around we should be familiar with coordinates.

Most JavaScript methods deal with one of two coordinate systems:

1. **Relative to the window** – similar to `position:fixed`, calculated from the window top/left edge.
 - we'll denote these coordinates as `clientX/clientY`, the reasoning for such name will become clear later, when we study event properties.
2. **Relative to the document** – similar to `position:absolute` in the document root, calculated from the document top/left edge.
 - we'll denote them `pageX/pageY`.

When the page is scrolled to the very beginning, so that the top/left corner of the window is exactly the document top/left corner, these coordinates equal each other. But after the document shifts, window-relative coordinates of elements change, as elements move across the window, while document-relative coordinates remain the same.

On this picture we take a point in the document and demonstrate its coordinates before the scroll (left) and after it (right):



When the document scrolled:

- `pageY` – document-relative coordinate stayed the same, it's counted from the document top (now scrolled out).

- `clientY` – window-relative coordinate did change (the arrow became shorter), as the same point became closer to window top.

Element coordinates: `getBoundingClientRect`

The method `elem.getBoundingClientRect()` returns window coordinates for a minimal rectangle that encloses `elem` as an object of built-in `DOMRect` class.

Main `DOMRect` properties:

- `x/y` – X/Y-coordinates of the rectangle origin relative to window,
- `width/height` – width/height of the rectangle (can be negative).

Additionally, there are derived properties:

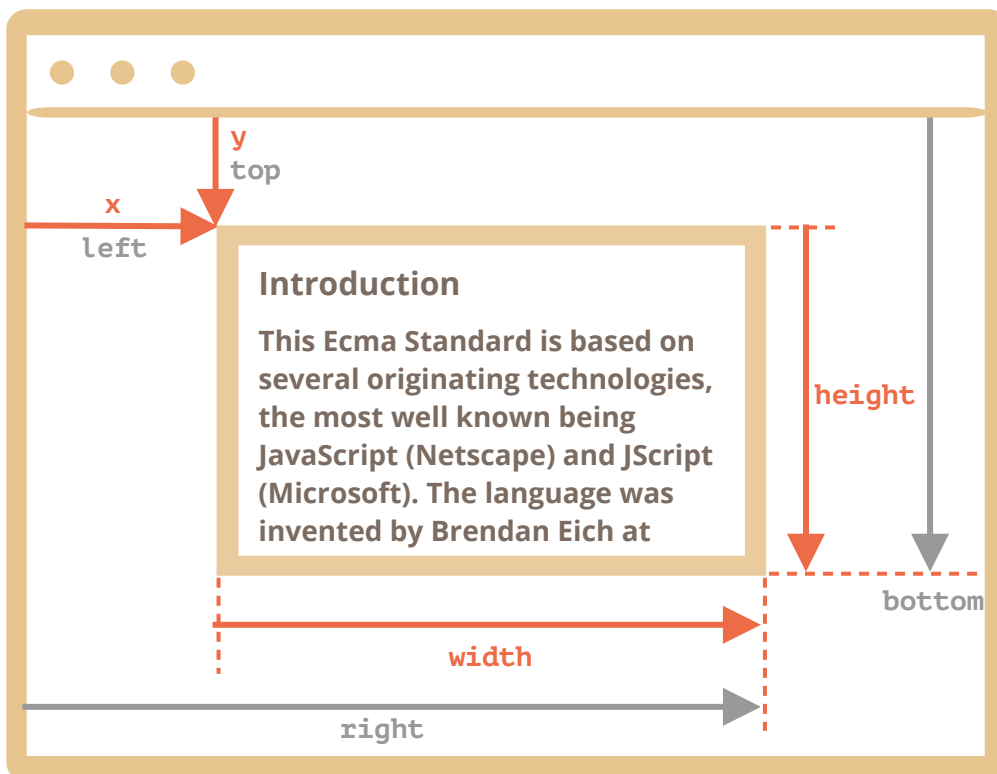
- `top/bottom` – Y-coordinate for the top/bottom rectangle edge,
- `left/right` – X-coordinate for the left/right rectangle edge.

For instance click this button to see its window coordinates:

Get coordinates using `button.getBoundingClientRect()` for this button

If you scroll the page and repeat, you'll notice that as window-relative button position changes, its window coordinates (`y/top/bottom` if you scroll vertically) change as well.

Here's the picture of `elem.getBoundingClientRect()` output:



As you can see, `x/y` and `width/height` fully describe the rectangle. Derived properties can be easily calculated from them:

- `left = x`
- `top = y`

- `right = x + width`
- `bottom = y + height`

Please note:

- Coordinates may be decimal fractions, such as `10.5`. That's normal, internally browser uses fractions in calculations. We don't have to round them when setting to `style.left/top`.
- Coordinates may be negative. For instance, if the page is scrolled so that `elem` is now above the window, then `elem.getBoundingClientRect().top` is negative.

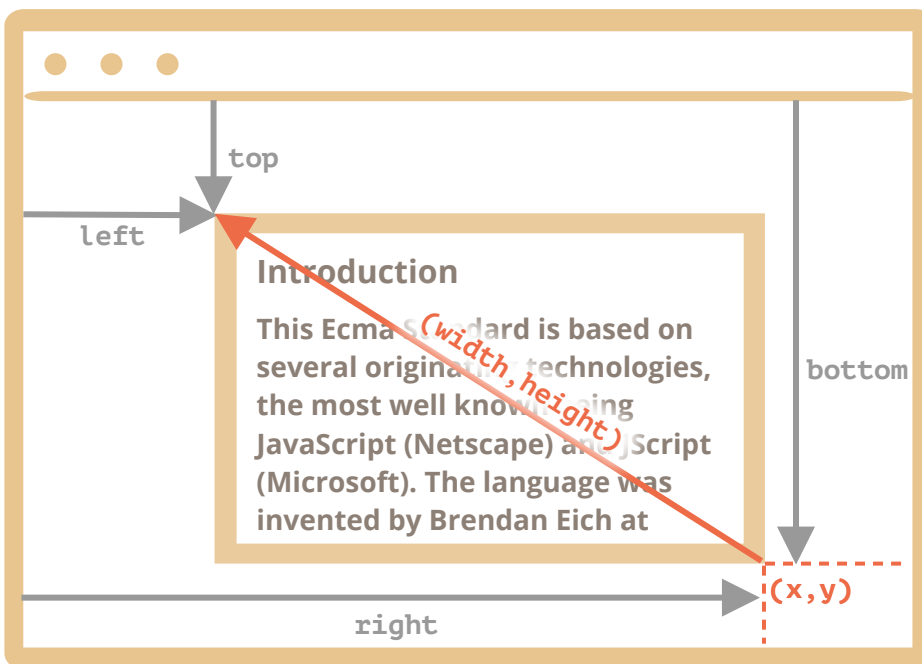
i Why derived properties are needed? Why does `top/left` exist if there's `x/y`?

Mathematically, a rectangle is uniquely defined with its starting point (x, y) and the direction vector $(width, height)$. So the additional derived properties are for convenience.

Technically it's possible for `width/height` to be negative, that allows for "directed" rectangle, e.g. to represent mouse selection with properly marked start and end.

Negative `width/height` values mean that the rectangle starts at its bottom-right corner and then "grows" left-upwards.

Here's a rectangle with negative `width` and `height` (e.g. `width=-200`, `height=-100`):



As you can see, `left/top` do not equal `x/y` in such case.

In practice though, `elem.getBoundingClientRect()` always returns positive `width/height`, here we mention negative `width/height` only for you to understand why these seemingly duplicate properties are not actually duplicates.

⚠ Internet Explorer and Edge: no support for `x/y`

Internet Explorer and Edge don't support `x/y` properties for historical reasons.

So we can either make a polyfill (add getters in `DomRect.prototype`) or just use `top/left`, as they are always the same as `x/y` for positive `width/height`, in particular in the result of `elem.getBoundingClientRect()`.

⚠ Coordinates `right/bottom` are different from CSS position properties

There are obvious similarities between window-relative coordinates and CSS `position:fixed`.

But in CSS positioning, `right` property means the distance from the right edge, and `bottom` property means the distance from the bottom edge.

If we just look at the picture above, we can see that in JavaScript it is not so. All window coordinates are counted from the top-left corner, including these ones.

elementFromPoint(x, y)

The call to `document.elementFromPoint(x, y)` returns the most nested element at window coordinates `(x, y)`.

The syntax is:

```
1 let elem = document.elementFromPoint(x, y);
```

For instance, the code below highlights and outputs the tag of the element that is now in the middle of the window:

```
1 let centerX = document.documentElement.clientWidth / 2;
2 let centerY = document.documentElement.clientHeight / 2;
3
4 let elem = document.elementFromPoint(centerX, centerY);
5
6 elem.style.background = "red";
7 alert(elem.tagName);
```



As it uses window coordinates, the element may be different depending on the current scroll position.

⚠ For out-of-window coordinates the `elementFromPoint` returns `null`

The method `document.elementFromPoint(x,y)` only works if `(x,y)` are inside the visible area.

If any of the coordinates is negative or exceeds the window width/height, then it returns `null`.

Here's a typical error that may occur if we don't check for it:

```
1 let elem = document.elementFromPoint(x, y);
2 // if the coordinates happen to be out of the window, then elem = null
3 elem.style.background = ''; // Error!
```

Using for “fixed” positioning

Most of time we need coordinates in order to position something.

To show something near an element, we can use `getBoundingClientRect` to get its coordinates, and then CSS position together with `left/top` (or `right/bottom`).

For instance, the function `createMessageUnder(elem, html)` below shows the message under `elem`:

```
1 let elem = document.getElementById("coords-show-mark");
2
3 function createMessageUnder(elem, html) {
4   // create message element
5   let message = document.createElement('div');
6   // better to use a css class for the style here
7   message.style.cssText = "position:fixed; color: red";
8
9   // assign coordinates, don't forget "px"!
10  let coords = elem.getBoundingClientRect();
11
12  message.style.left = coords.left + "px";
13  message.style.top = coords.bottom + "px";
14
15  message.innerHTML = html;
16
17  return message;
18 }
19
20 // Usage:
21 // add it for 5 seconds in the document
22 let message = createMessageUnder(elem, 'Hello, world!');
23 document.body.append(message);
24 setTimeout(() => message.remove(), 5000);
```

Click the button to run it:

Button with id="coords-show-mark", the message will appear under it

The code can be modified to show the message at the left, right, below, apply CSS animations to “fade it in” and so on. That's easy, as we have all the coordinates and sizes of the element.

But note the important detail: when the page is scrolled, the message flows away from the button.

The reason is obvious: the message element relies on `position:fixed`, so it remains at the same place of the window while the page scrolls away.

To change that, we need to use document-based coordinates and `position:absolute`.

Document coordinates

Document-relative coordinates start from the upper-left corner of the document, not the window.

In CSS, window coordinates correspond to `position:fixed`, while document coordinates are similar to `position:absolute` on top.

We can use `position:absolute` and `top/left` to put something at a certain place of the document, so that it remains there during a page scroll. But we need the right coordinates first.

There's no standard method to get the document coordinates of an element. But it's easy to write it.

The two coordinate systems are connected by the formula:

- $\text{pageY} = \text{clientY} + \text{height of the scrolled-out vertical part of the document.}$
- $\text{pageX} = \text{clientX} + \text{width of the scrolled-out horizontal part of the document.}$

The function `getCoords(elem)` will take window coordinates from `elem.getBoundingClientRect()` and add the current scroll to them:

```
1 // get document coordinates of the element
2 function getCoords(elem) {
3   let box = elem.getBoundingClientRect();
4
5   return {
6     top: box.top + window.pageYOffset,
7     left: box.left + window.pageXOffset
8   };
9 }
```

If in the example above we used it with `position:absolute`, then the message would stay near the element on scroll.

The modified `createMessageUnder` function:

```
1 function createMessageUnder(elem, html) {
2   let message = document.createElement('div');
3   message.style.cssText = "position:absolute; color: red";
4
5   let coords = getCoords(elem);
6
7   message.style.left = coords.left + "px";
8   message.style.top = coords.bottom + "px";
9
10  message.innerHTML = html;
11
12  return message;
13 }
```

Summary

Any point on the page has coordinates:

1. Relative to the window – `elem.getBoundingClientRect()` .
2. Relative to the document – `elem.getBoundingClientRect()` plus the current page scroll.

Window coordinates are great to use with `position:fixed` , and document coordinates do well with `position:absolute` .

Both coordinate systems have their pros and cons; there are times we need one or the other one, just like CSS `position absolute` and `fixed` .

✓ Tasks



Find window coordinates of the field

importance: 5

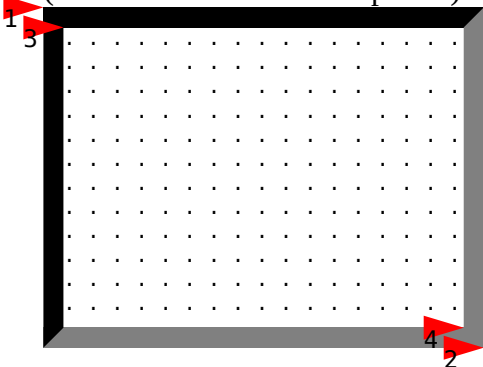
In the iframe below you can see a document with the green “field”.

Use JavaScript to find window coordinates of corners pointed by with arrows.

There's a small feature implemented in the document for convenience. A click at any place shows coordinates there.

Click anywhere to get window coordinates.
That's for testing, to check the result you get by JavaScript.
(click coordinates show up here)



Your code should use DOM to get window coordinates of:

1. Upper-left, outer corner (that's simple).
2. Bottom-right, outer corner (simple too).
3. Upper-left, inner corner (a bit harder).
4. Bottom-right, inner corner (there are several ways, choose one).

The coordinates that you calculate should be the same as those returned by the mouse click.

P.S. The code should also work if the element has another size or border, not bound to any fixed values.

[Open a sandbox for the task.](#)

solution

Show a note near the element

importance: 5

Create a function `positionAt(anchor, position, elem)` that positions `elem`, depending on `position` near anchor element.

The `position` must be a string with any one of 3 values:

- "top" – position `elem` right above anchor
- "right" – position `elem` immediately at the right of anchor
- "bottom" – position `elem` right below anchor

It's used inside function `showNote(anchor, position, html)`, provided in the task source code, that creates a "note" element with given `html` and shows it at the given `position` near the anchor.

Here's the demo of notes:

>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Reprehenderit sint atque dolorum fuga ad incidunt voluptatum error fugiat animi amet! Odio temporibus nulla id unde quaerat dignissimos enim nisi rem provident molestias sit tempore omnis recusandae esse sequi officia sapiente.

“
Teacher: Why are you late?
Student: There was a man who lost a hundred dollar bill.
Teacher: That's nice. Were you helping him look for it?
Student: No. I was standing on it.

..

note above

note at the right

note below

..

>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Reprehenderit sint atque dolorum fuga ad incidunt voluptatum error fugiat animi amet! Odio temporibus nulla id unde quaerat dignissimos enim nisi rem provident molestias sit tempore omnis recusandae esse sequi officia sapiente.

[Open a sandbox for the task.](#)

solution

Show a note near the element (absolute)

importance: 5

Modify the solution of the [previous task](#) so that the note uses `position: absolute` instead of `position: fixed`.

That will prevent its “runaway” from the element when the page scrolls.

Take the solution of that task as a starting point. To test the scroll, add the style `<body style="height: 2000px">`.

solution

Position the note inside (absolute)

importance: 5

Extend the previous task [Show a note near the element \(absolute\)](#): teach the function `positionAt(anchor, position, elem)` to insert `elem` inside the `anchor`.

New values for `position`:

- `top-out`, `right-out`, `bottom-out` – work the same as before, they insert the `elem` over/right/under `anchor`.
- `top-in`, `right-in`, `bottom-in` – insert `elem` inside the `anchor`: stick it to the upper/right/bottom edge.

For instance:

```
1 // shows the note above blockquote
2 positionAt(blockquote, "top-out", note);
3
4 // shows the note inside blockquote, at the top
5 positionAt(blockquote, "top-in", note);
```

The result:

>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Reprehenderit sint atque dolorum fuga ad incidunt voluptatum error fugiat animi amet! Odio temporibus nulla id unde quaerat dignissimos enim nisi rem provident molestias sit tempore omnis recusandae esse sequi officia sapiente.

“

Teacher: Why are you late?

Student: There was a man who lost a hundred dollar bill.

Teacher: That's nice. Were you helping him look for it?

Student: No. I was standing on it.

>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Reprehenderit sint atque dolorum fuga ad incidunt voluptatum error fugiat animi amet! Odio temporibus nulla id unde quaerat dignissimos enim nisi rem

As the source code, take the solution of the task [Show a note near the element \(absolute\)](#).

solution



Previous lesson

Next lesson



Share



[Tutorial map](#)

Comments

- If you have suggestions what to improve - please [submit a GitHub issue](#) or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>` , for more than 10 lines – use a sandbox ([plnkr](#), [JSBin](#), [codepen](#)...)