

 [szimek](#) / [signature_pad](#)

HTML5 canvas based smooth signature drawing

 szimek.github.io/signature_pad/

 MIT License

☆ 6.1k stars 🍴 1.4k forks

☆ Star

👁 Watch

<> Code

! Issues 160

🔗 Pull requests 31

▶ Actions

📁 Projects

📖 Wiki

🛡 Sec

Join GitHub today

Dismiss

GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

Sign up

🔗 master ▾

Go to file



szimek Merge pull request [#481](#) from DocSpring/fix_strokeUpdate_ra... ... ✓ 17 days ago ⌚ 221

[View code](#)

README.md

Signature Pad

npm package

3.0.0-beta.3

build

passing



maintainability

C

Signature Pad is a JavaScript library for drawing smooth signatures. It's HTML5 canvas based and uses variable width Bézier curve interpolation based on [Smoother Signatures](#) post by [Square](#). It works in all modern desktop and mobile browsers and doesn't depend on any external libraries.



Demo

[Demo](#) works in desktop and mobile browsers. You can check out its [source code](#) for some tips on how to handle window resize and high DPI screens. You can also find more about the latter in [HTML5 Rocks tutorial](#).

Other demos

- Erase feature: <https://jsfiddle.net/szimek/jq9cyzuc/>
- Undo feature: <https://jsfiddle.net/szimek/osenvxjc/>

Installation

You can install the latest release using npm:

```
npm install --save signature_pad
```

or Yarn:

```
yarn add signature_pad
```

You can also add it directly to your page using `<script>` tag:

```
<script src="https://cdn.jsdelivr.net/npm/signature_pad@2.3.2/dist/signature_pad.min.js"></script>
```



You can select a different version at https://www.jsdelivr.com/package/npm/signature_pad.

This library is provided as UMD (Universal Module Definition) and ES6 module.

Usage

API

```
var canvas = document.querySelector("canvas");

var signaturePad = new SignaturePad(canvas);

// Returns signature image as data URL (see https://mdn.io/todataurl for the list of
signaturePad.toDataURL(); // save image as PNG
signaturePad.toDataURL("image/jpeg"); // save image as JPEG
signaturePad.toDataURL("image/svg+xml"); // save image as SVG

// Draws signature image from data URL.
// NOTE: This method does not populate internal data structure that represents drawn
signaturePad.fromDataURL("data:image/png;base64,iVBORw0K...");

// Returns signature image as an array of point groups
const data = signaturePad.toData();

// Draws signature image from an array of point groups
signaturePad.fromData(data);

// Clears the canvas
signaturePad.clear();

// Returns true if canvas is empty, otherwise returns false
signaturePad.isEmpty();

// Unbinds all event handlers
signaturePad.off();

// Rebinds all event handlers
signaturePad.on();
```



Options

dotSize

(float or function) Radius of a single dot.

minWidth

(float) Minimum width of a line. Defaults to 0.5 .

maxWidth

(float) Maximum width of a line. Defaults to 2.5 .

throttle

(integer) Draw the next point at most once per every x milliseconds. Set it to 0 to turn off throttling. Defaults to 16 .

minDistance

(integer) Add the next point only if the previous one is farther than x pixels. Defaults to 5 .

backgroundColor

(string) Color used to clear the background. Can be any color format accepted by `context.fillStyle` . Defaults to "rgba(0,0,0,0)" (transparent black). Use a non-transparent color e.g. "rgb(255,255,255)" (opaque white) if you'd like to save signatures as JPEG images.

penColor

(string) Color used to draw the lines. Can be any color format accepted by `context.fillStyle` . Defaults to "black" .

velocityFilterWeight

(float) Weight used to modify new velocity based on the previous velocity. Defaults to 0.7 .

onBegin

(function) Callback when stroke begin.

onEnd

(function) Callback when stroke end.

You can set options during initialization:

```
var signaturePad = new SignaturePad(canvas, {  
  minWidth: 5,  
  maxWidth: 10,  
  penColor: "rgb(66, 133, 244)"  
});
```

or during runtime:

```
var signaturePad = new SignaturePad(canvas);
signaturePad.minWidth = 5;
signaturePad.maxWidth = 10;
signaturePad.penColor = "rgb(66, 133, 244)";
```

Tips and tricks

Handling high DPI screens

To correctly handle canvas on low and high DPI screens one has to take `devicePixelRatio` into account and scale the canvas accordingly. This scaling is also necessary to properly display signatures loaded via `SignaturePad#fromDataURL`. Here's an example how it can be done:

```
function resizeCanvas() {
    var ratio = Math.max(window.devicePixelRatio || 1, 1);
    canvas.width = canvas.offsetWidth * ratio;
    canvas.height = canvas.offsetHeight * ratio;
    canvas.getContext("2d").scale(ratio, ratio);
    signaturePad.clear(); // otherwise isEmpty() might return incorrect value
}

window.addEventListener("resize", resizeCanvas);
resizeCanvas();
```

Instead of `resize` event you can listen to screen orientation change, if you're using this library only on mobile devices. You can also throttle the `resize` event - you can find some examples on [this MDN page](#).

When you modify width or height of a canvas, it will be automatically cleared by the browser. `SignaturePad` doesn't know about it by itself, so you can call

`signaturePad.clear()` to make sure that `signaturePad.isEmpty()` returns correct value in this case.

This clearing of the canvas by the browser can be annoying, especially on mobile devices e.g. when screen orientation is changed. There are a few workarounds though, e.g. you can [lock screen orientation](#), or read an image from the canvas before resizing it and write the image back after.

Handling data URI encoded images on the server side

If you are not familiar with data URI scheme, you can read more about it on [Wikipedia](#).

There are 2 ways you can handle data URI encoded images.

You could simply store it in your database as a string and display it in HTML like this:

```

```

but this way has many disadvantages - it's not easy to get image dimensions, you can't manipulate it e.g. to create a thumbnail and it also [has some performance issues on mobile devices](#).

Thus, more common way is to decode it and store as a file. Here's an example in Ruby:

```
require "base64"

data_uri = "data:image/png;base64,iVBORw0K..."
encoded_image = data_uri.split(",")[1]
decoded_image = Base64.decode64(encoded_image)
File.open("signature.png", "wb") { |f| f.write(decoded_image) }
```

Here's an example in PHP:

```
$data_uri = "data:image/png;base64,iVBORw0K...";
$encoded_image = explode(",", $data_uri)[1];
$decoded_image = base64_decode($encoded_image);
file_put_contents("signature.png", $decoded_image);
```

Here's an example in C# for ASP.NET:

```
var dataUri = "data:image/png;base64,iVBORw0K...";
var encodedImage = dataUri.Split(",")[1];
var decodedImage = Convert.FromBase64String(encodedImage);
System.IO.File.WriteAllBytes("signature.png", decodedImage);
```

Removing empty space around a signature

If you'd like to remove (trim) empty space around a signature, you can do it on the server side or the client side. On the server side you can use e.g. ImageMagic and its `trim` option: `convert -trim input.jpg output.jpg`. If you don't have access to the server, or just want to trim the image before submitting it to the server, you can do it on the client side as well. There are a few examples how to do it, e.g. [here](#) or [here](#) and there's also a tiny library [trim-canvas](#) that provides this functionality.


Drawing over an image

Demo: <https://jsfiddle.net/szimek/d6a78gwq/>

License

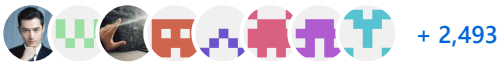
Released under the [MIT License](#).

Releases 28

 **v3.0.0-beta.3** Latest
on Jul 29, 2018

[+ 27 releases](#)

Used by 2,501



Contributors 24



[+ 13 contributors](#)

Languages

● TypeScript 93.0% ● JavaScript 7.0%