



Processor



PDF Generation



PROCESSOR &gt; GUIDES &gt; PDF GENERATION

# Create PDFs from Scratch in Linux



PSPDFKit Processor has been deprecated and replaced by [Document Engine](#). To migrate to Document Engine and unlock advanced document processing capabilities, refer to our migration guide. Learn more about these enhancements on our [blog](#).

This guide will take you through the process of designing a PDF using the PDF Generation feature.

Before you get started, make sure [Processor is up and running](#).

You can download and use either of the following sample documents for the examples in this guide:

✧ [Example eight-page PDF](#)

✧ [Example four-page PDF](#)

You'll be sending [multipart POST requests with instructions](#) to Processor's `/build` endpoint. To learn more about multipart requests, refer to our blog post on the topic, [A Brief Tour of Multipart Requests](#).

Check out the [API Reference](#) to learn more about the `/build` endpoint and all the actions you can perform on PDFs with PSPDFKit Processor.

## Document content

The PDF Generation component leverages HTML's prominence and wide-reaching support to render the content and layout of a desired PDF. To produce your first PDF, it's as simple as passing



ASK AI

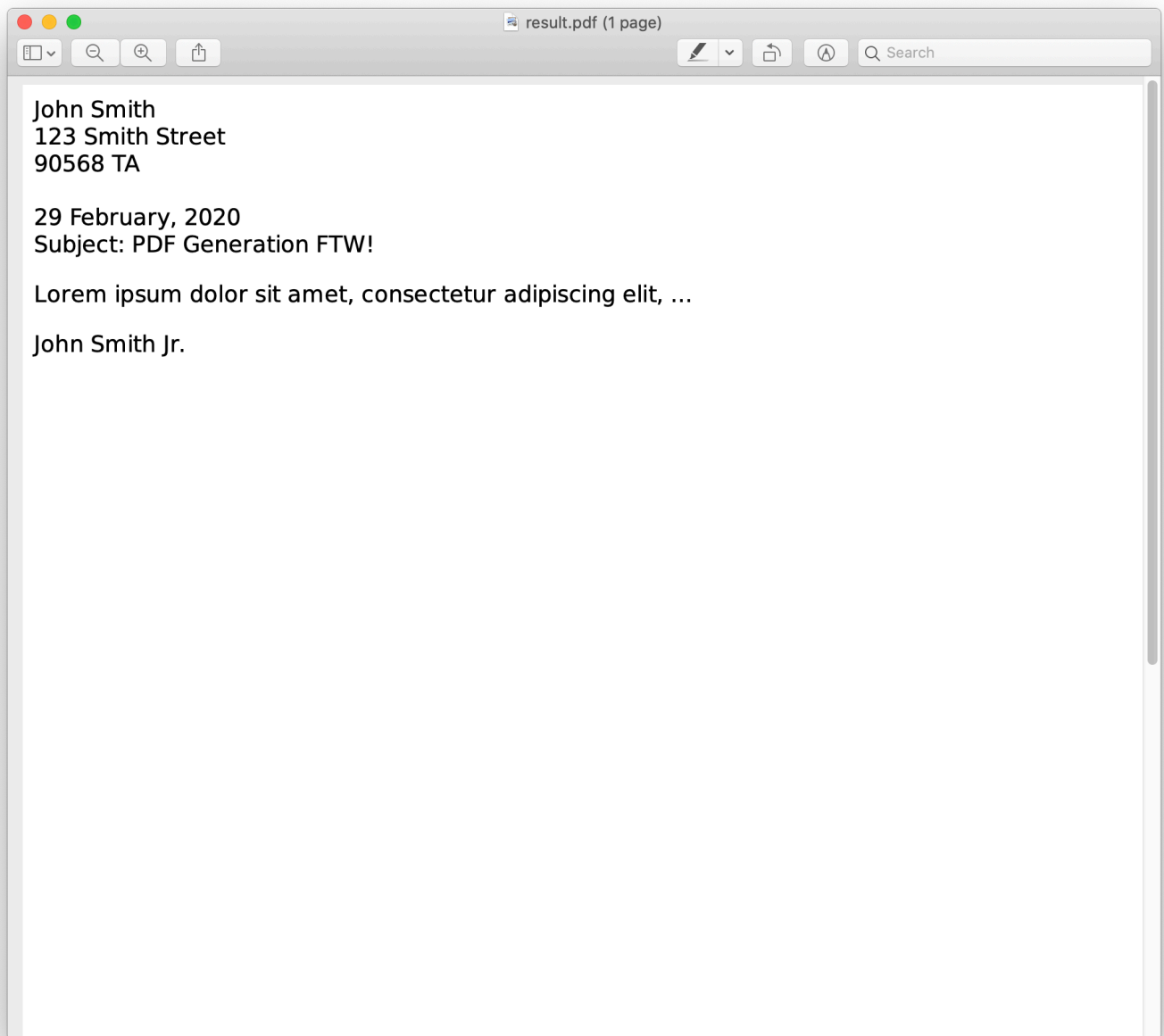
HTML document, as shown in the following letter example. The letter holds an address, subject, main body, and sign off, all in separate `div` blocks:

```
1  <!DOCTYPE html>
2  <html>
3    <body>
4      <div>
5        John Smith<br />
6        123 Smith Street <br />
7        90568 TA <br />
8        <br />
9        29 February 2020
10     </div>
11     <div>Subject: PDF Generation FTW!</div>
12     <div>
13       <p>
14         Lorem ipsum dolor sit amet, consectetur adipiscing elit, ...
15       </p>
16     </div>
17     <div>John Smith Jr. <br /></div>
18   </body>
19 </html>
```

Next, send the HTML file from above to Processor for generation. This is done by sending a multipart request to the `/build` endpoint. Attach the `instructions` JSON, along with the HTML file from above:

```
1  curl -X POST http://localhost:5000/api/build \
2    -F page.html=@/path/to/page.html \
3    -F style.css=@/path/to/style.css \
4    -F my-image.jpg=@/path/to/my-image.jpg \
5    -F instructions='{
6      "parts": [
7        {
8          "html": "page.html",
9          "assets": [
10            "style.css",
11            "my-image.jpg"
12          ]
13        }
14      ]
15    }' \
16    -o result.pdf
```

After performing the above `curl` command, you'll receive a PDF that looks like the following.



To help design your PDF, preview your HTML in Chrome or another Chromium-based browser. Minor differences, outlined in the [HTML layout and CSS considerations][html-css-considerations] guide, are to be expected. To further enhance the design experience, you can use the Chrome DevTools to resize your viewport to match your desired page size.

## Document layout

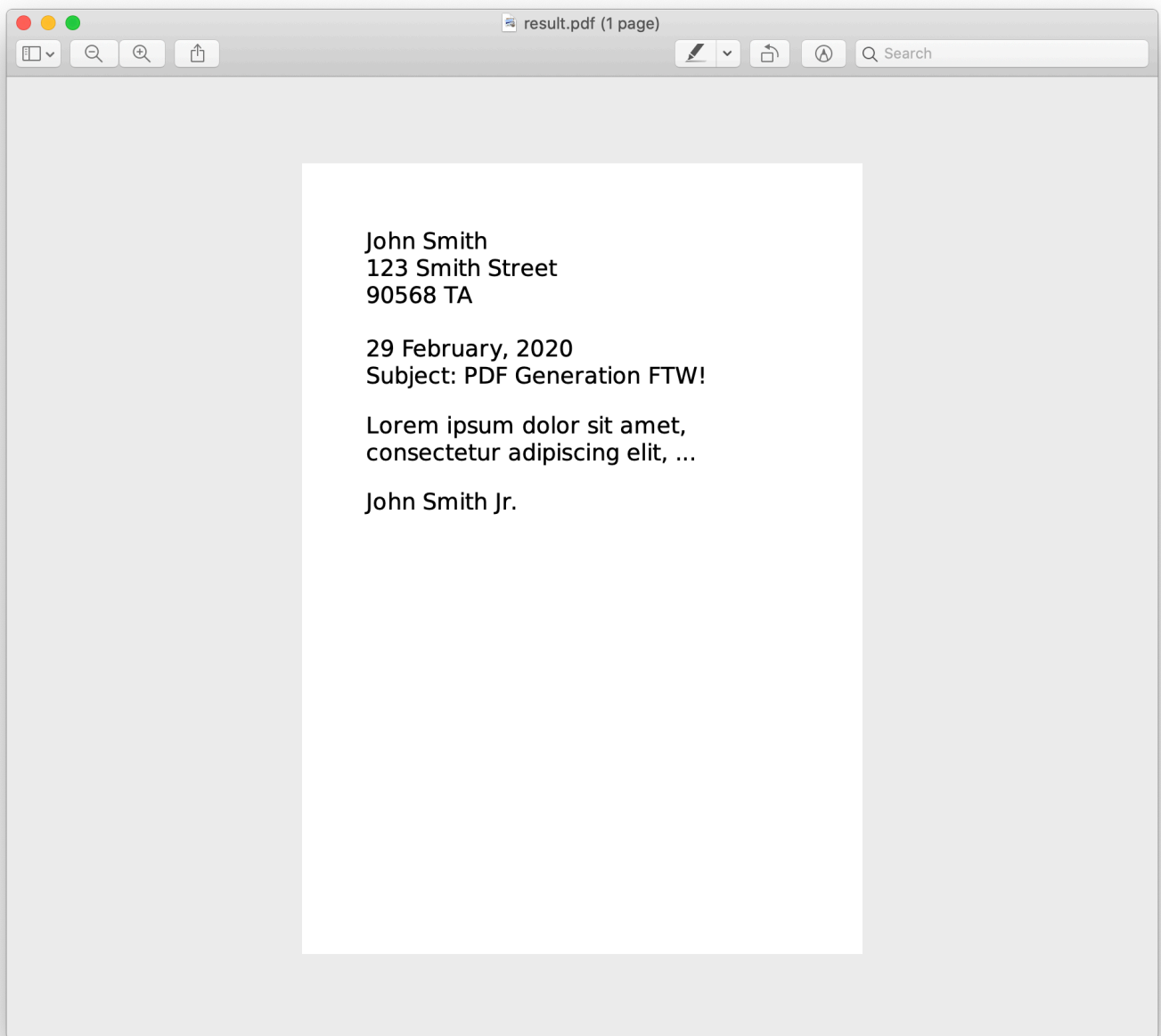
Now that you have the content you want, the next step is to improve the layout.

From the last example, you can see the text was pushing up to the side of the page, and the page size was A4. With the PDF Generation feature, the page size and margins are adjustable. To change them, include the extra information in the [PDF Generation schema](#):



```
1 curl -X POST http://localhost:5000/api/build \  
2   -F page.html=@/path/to/page.html \  
3   -F instructions='{  
4     "parts": [  
5       {  
6         "html": "page.html",  
7         "layout": {  
8           "size": "a6",  
9           "margin": {  
10            "top": 10,  
11            "left": 10,  
12            "bottom": 10,  
13            "right": 10  
14          }  
15        }  
16      ]  
17    }'  
18  \
```

The size of the page has been reduced to A6, and all the edges have a margin of 10 mm.



## Document styling

To style various aspects of the documents, it's possible to use CSS, much like you would on the web. CSS is well supported and expressive, which helps you achieve any look you desire.

Continuing with the letter example, move the address over to the right-hand side of the page, and style the subject line to make it more prominent:

```
1  <!DOCTYPE html>
2  <head>
3    <style type="text/css">
4      .address {
5        text-align: left;
6        float: right;
7        margin-bottom: 20px;
```



```

8     }
9     .subject {
10         clear: both;
11         font-weight: bold;
12     }
13 </style>
14 </head>
15 <html>
16     <body>
17         <div class="address">
18             John Smith<br />
19             123 Smith Street <br />
20             90568 TA <br />
21             <br />
22             29 February 2020
23         </div>
24         <div class="subject">Subject: PDF Generation FTW!</div>
25         <div>
26             <p>
27                 Lorem ipsum dolor sit amet, consectetur adipiscing elit, ...
28             </p>
29         </div>
30         <div>John Smith Jr. <br /></div>
31     </body>
32 </html>

```

Send the same multipart request to Processor with the new HTML file. The result is a PDF document with certain elements styled as described in the following CSS file:

```

SHELL  HTTP
1  curl -X POST http://localhost:5000/api/build \
2      -F page.html=@/path/to/page.html \
3      -F instructions='{
4          "parts": [
5              {
6                  "html": "page.html",
7                  "layout": {
8                      "size": "a6",
9                      "margin": {
10                         "top": 10,
11                         "left": 10,
12                         "bottom": 10,
13                         "right": 10
14                     }
15                 }
16             }
17         ]
18
19

```

# Font selection

Rather than using the default font, it may be desirable to use a custom font to enhance the look of your final document.

You can do so by providing a font file in the [generation schema](#) and specifying the font in your HTML.

In the following example, you can use an Open Sans font, which you can retrieve from the Google Fonts repository.

First, adjust the HTML to both use the font file and specify the font family. Note that the `src` of the font file is referenced with no subdirectories, as though the file were residing next to the HTML. This is because PDF Generation only supports a flat-like directory structure:

```
1  <!DOCTYPE html>
2  <head>
3    <style type="text/css">
4      @font-face {
5        font-family: "Open Sans";
6        src: url("OpenSans-Regular.ttf") format("truetype");
7      }
8      body {
9        font-family: "Open Sans", sans-serif;
10     }
11     .address {
12       text-align: left;
13       float: right;
14       margin-bottom: 20px;
15     }
16     .subject {
17       clear: both;
18       font-weight: bold;
19     }
20   </style>
21 </head>
22 <html>
23   <body>
24     <div class="address">
25       John Smith<br />
26       123 Smith Street <br />
27       90568 TA <br />
28       <br />
29       29 February 2020
30     </div>
31     <div class="subject">Subject: PDF Generation FTW!</div>
```

```
32     <div>
33         <p>
34             Lorem ipsum dolor sit amet, consectetur adipiscing elit, ...
35         </p>
36     </div>
37     <div>John Smith Jr. <br /></div>
38 </body>
39 </html>
```

Send the multipart request to Processor. Be sure that the extra `OpenSans-Regular.ttf` asset is referenced in the HTML. The example below shows how the multipart request is formed.

The font file, `OpenSans-Regular.ttf`, doesn't reside in a subdirectory, and its name in the multipart request is the same name that's referenced in the HTML file.

SHELL

HTTP

```
1 curl -X POST http://localhost:5000/api/build \
2     -F page.html=@/path/to/page.html \
3     -F OpenSans-Regular.ttf=@/path/to/OpenSans-Regular.ttf \
4     -F instructions='{
5     "parts": [
6     {
7         "html": "page.html",
8         "layout": {
9             "size": "a6",
10            "margin": {
11                "top": 10,
12                "left": 10,
13                "bottom": 10,
14                "right": 10
15            },
16            "assets": [
17                "OpenSans-Regular.ttf"
18            ]
19        }
20    }
21 ]
22 }' \
23     -o result.pdf
```

`result.pdf` will now have the font applied to the whole document and will render like the following.



# Next steps

You've walked through each step of defining, aligning, and styling, and now you should have the skills to design a wide range of unique PDFs. For an enhanced design experience, we suggest using a Chromium-based browser, which will speed up the development process.

After refining your skills, you may realize you have the need to inject elements and values into the HTML prior to sending the document for generation. In the following guides, we demonstrate how to inject data into an HTML template to produce highly customized PDFs — such as invoices with multiple line items, order forms with unique products, or runtime-generated graphs — so as to provide additional context for your customers.

Choose your language-specific Variable Data guide to learn how to inject elements and values at runtime:

✧ [JavaScript](#)

✧ [Python](#)

✧ [Java](#)

✧ [C# \(.NET\)](#)

✧ [PHP](#)

---

Was this helpful?

✓ YES

✗ NO

---

Questions? [Contact us](#)

