



Processor



Choose a Page



PROCESSOR &gt; GUIDES

# Robust build API for advanced PDF processing



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](#).

Processor's `/build` API allows you to assemble a PDF from multiple [parts](#), such as an existing PDF, a blank page, or an HTML page. You can apply one or more [actions](#), such as watermarking, rotating pages, or importing annotations. Once the entire PDF is generated from its parts, you can also apply additional actions, such as optical character recognition (OCR), to the assembled PDF itself.

## Authentication

The `/build` API supports multiple types of authentication, which you can read about in our [Authentication API](#) guide.

## Multipart Request

The basic use case for the `/build` API is to upload all inputs together in the build instructions with the `multipart/form-data` request. In the following example, the request to the `/build` API consists of multiple parts and actions:



ASK AI



```
1 curl -X POST http://localhost:5000/build \  
2   -H "Authorization: Token token=secret" \  
3   -o result.pdf \  
4   -F document=@document.pdf \  
5   -F index=@index.html \  
6   -F logo=@logo.png \  
7   -F instructions='{  
8       "parts": [  
9           {  
10              "file": "document",  
11              "pages": {  
12                  "start": 0,  
13                  "end": 0  
14              },  
15              "actions": [  
16                  {  
17                      "type": "flatten"  
18                  }  
19              ]  
20          },  
21          {  
22              "html": "index"  
23          },  
24          {  
25              "page": "new",  
26              "pageCount": 3,  
27              "backgroundColor": "green"  
28          }  
29      ],  
30      "actions": [  
31          {  
32              "type": "watermark",  
33              "image": "logo",  
34              "width": "25%"  
35          }  
36      ],  
37      "output": {  
38          "type": "pdf"  
39      }  
}
```

In the request, the output file is set to `result.pdf`. On the success of the request, the output will be written to this file.

The request requires three dependencies: `document.pdf`, `index.html`, and `logo.png`. The presence of any unused dependencies results in the request failing.

The `instructions` JSON consists of three fields: `parts`, `actions`, and `output`. In this example, the output file contains the following elements:

- ✧ The first page from the `document.pdf` file. This page is flattened, which embeds all annotations.

- ❖ The `index.html` file converted to a PDF document.
- ❖ Three newly created pages with a green background.

A watermark is then applied across all pages, and the output is converted to a PDF document.

For more information about all possible options for parts, actions, and output, refer to the relevant specification below.

## Simple Request

The `/build` API supports inputs provided from remote URLs. If all inputs are provided as remote URLs, the multipart request isn't necessary and can be simplified to a non-multipart request with the `application/json` content type.

For example, to provide all inputs from the previous example via URL, the processing request can be simplified to:

```
1 curl -X POST http://localhost:5000/build \
2   -H "Authorization: Token token=secret" \
3   -H "Content-Type: application/json" \
4   -o result.pdf \
5   --data '{
6     "parts": [
7       {
8         "file": {"url": "https://remote-files-storage/document.pdf"},
9         "pages": {
10          "start": 0,
11          "end": 0
12        },
13        "actions": [
14          {
15            "type": "flatten"
16          }
17        ]
18      },
19      {
20        "html": {"url": "https://remote-files-storage/index.html"}
21      },
22      {
23        "page": "new",
24        "pageCount": 3,
25        "backgroundColor": "green"
26      }
27    ],
28    "actions": [
29      {
30        "type": "watermark",
31        "image": {"url": "https://remote-files-storage/logo.png"},
```

```

32         "width": "25%"
33     }
34 ],
35     "output": {
36         "type": "pdf"
37     }
38 }'

```

## API Specification

This section provides a full reference of the `/build` API.

### Inputs

Input files and assets used during processing can be provided either as a part in the `multipart/form-data` request or as a URL.

- ✧ Part name — Parts in the `multipart/form-data` request are referenced by their names:

```
type InputName = string;
```



- ✧ Remote URL — Inputs for processing can be provided with a URL:

```

1  type InputUrl = {
2      // Valid absolute URL that's accessible by the Processor service.
3      url: string;;
4      // An optional SHA256 hash of the file content. Used to check download
5      sha256?: string;
6  };

```



### Instructions

When making requests to the API, the `instructions` object needs to follow the given specification. The `parts` parameter is required, and the `actions` and `output` are optional:

```

1  type Instructions = {
2      parts: Part[];
3      actions?: Action[];

```



```
4   output?: Output;  
5   };
```

## Parts

The `parts` object can have the following types outlined below.

### File Part

This is used to import an already existing PDF file:

```
1  type Part = {  
2    // The input file that will be used as a source for the part.  
3    file: InputName | InputUrl;  
4    // Password of the file, if it's password protected.  
5    password?: string;  
6    // The page range to include in the part.  
7    pages?: {  
8      // The page index can be negative, indicating an offset from the last page  
9      // being the last page itself.  
10     start?: number | 'first' | 'last';  
11     end?: number | 'first' | 'last';  
12   };  
13   // The following actions will only be applied to the given part, and  
14   // not the entire document.  
15   actions?: Action[];  
16 };
```

### HTML Part

This is used to generate a PDF from HTML. For detailed information on generating a PDF from HTML, check out our [PDF Generation](#) guide:

```
1  type PageSize =  
2    | 'A0'  
3    | 'A1'  
4    | 'A2'  
5    | 'A3'  
6    | 'A4'  
7    | 'A5'  
8    | 'A6'  
9    | 'A7'  
10   | 'A8'
```

```

11 | 'Letter'
12 | 'Legal';
13
14 type Part = {
15   // The HTML file that will be used as a source for the part.
16   html: InputName | InputUrl;
17   // All assets imported in the HTML. Reference the name passed in the multipa.
18   assets?: Array<string>;
19   layout?: {
20     orientation?: 'landscape' | 'portrait';
21     size?:
22       | {
23         width: number;
24         height: number;
25       }
26       | PageSize; // {width, height} in mm or page size preset.
27   margin?: {
28     // Margin sizes in mm.
29     left: number;
30     top: number;
31     right: number;
32     bottom: number;
33   };
34 };
35 actions?: Action[];
36 };

```

## New Page Part

This is used to create a new blank page:

```

1 type Part = {
2   page: 'new';
3   // The number of blank pages to add.
4   pageCount?: number;
5   // Background color of the blank pages.
6   // Specified with predefined color names,
7   // or with RGB, HEX, HSL, RGBA, or HSLA values.
8   backgroundColor?: string;
9   layout?: {
10    orientation?: 'landscape' | 'portrait';
11    size?: {
12      width: number;
13      height: number;
14    };
15    margin?: {
16      left?: number;
17      right?: number;
18      top?: number;
19      bottom?: number;

```



```
20     };
21   };
22   // The following actions will only be applied to the given part, and
23   // not the entire document.
24   actions?: Action[];
25 };
```

## Actions

**Actions** can be one of the following outlined below.

### Apply JSON

This applies the given Instant JSON file to the document, and it's used to import annotations to a document. It can also be used to fill forms:

```
1  type Action = {
2    type: 'applyInstantJson';
3    // The input file in Instant JSON format.
4    file: InputName | InputUrl;
5  };
```



### Apply XFDF

This applies the given XFDF file to the document, and it's used to import annotations to a document. It can also be used to fill forms:

```
1  type Action = {
2    type: 'applyXfdf';
3    // The input file in XFDF format.
4    file: InputName | InputUrl;
5  };
```



### Flatten

This flattens the annotations in the given part or document:

```
1  type Action = {
2    type: 'flatten';
3    // Optional list of annotation IDs to flatten. These can be annotation IDs o
4    // If provided, only the annotations with the given IDs will be flattened.
5    annotationIds?: string[];
6  };
```

## Insert Page

This inserts a blank page in the middle of a document:

```
1  type Action = {
2    type: 'insertPage';
3    // Either one of `afterPageIndex` or `beforePageIndex` is required.
4    afterPageIndex?: integer;
5    beforePageIndex?: integer;
6    pageHeight: number;
7    pageWidth: number;
8    rotateBy?: 90 | 180 | 270;
9    // Background color of the blank pages.
10   // Specified with predefined color names,
11   // or with RGB, HEX, HSL, RGBA, or HSLA values.
12   backgroundColor?: string;
13 };
```

## OCR

This performs optical character recognition (OCR) in the given document. The list of supported languages can be found in the supported languages guide:

```
1  type Action = {
2    type: 'ocr';
3    language: string;
4  };
```

## Rotate Page

This rotates all the pages of the given document by the angle specified:





```
1 type Action = {
2   type: 'rotate';
3   rotateBy: 90 | 180 | 270;
4 };
```

## Watermark

This adds an image or text watermark on all the pages of the given document:



```
1 type Action = TextWatermarkAction | ImageWatermarkAction;
2
3 type TextWatermarkAction = {
4   type: 'watermark';
5   text: string;
6   // Both width and height are required.
7   width: Dimension;
8   height: Dimension;
9   // Only one out of top or bottom is allowed.
10  top?: Position;
11  // Only one out of right or left is allowed.
12  right?: Position;
13  bottom?: Position;
14  left?: Position;
15  rotation?: integer;
16  opacity?: number;
17  fontSize?: integer;
18  fontColor?: string;
19  // `italic` and `bold` are supported styles.
20  // They can be used together.
21  fontStyle?: string[];
22  fontFamily?: string;
23 };
24
25 type ImageWatermarkAction = {
26   type: 'watermark';
27   // The input image file that will be used as a source for the image watermark.
28   image: InputName | InputUrl;
29   // Both width and height are required.
30   width: Dimension;
31   height: Dimension;
32   // Only one out of top or bottom is allowed.
33   top?: Position;
34   // Only one out of right or left is allowed.
35   right?: Position;
36   bottom?: Position;
37   left?: Position;
38   rotation?: integer;
39   opacity?: number;
40 };
```



```

41
42 type Dimension = {
43   value: integer;
44   unit: 'pt' | '%';
45 };
46
47 type Position = {
48   value: integer;
49   unit: 'pt' | '%';
50 };

```

## Create Redactions

This creates redaction annotations according to the given strategy. Once redactions are created, they need to be applied using the `applyRedactions` action:

```

1  type Content = {
2    fillColor?: string; // default is "#000000"
3    overlayText?: string; // default is null
4    repeatOverlayText?: boolean; // default is false
5    color?: string; // default is "#F82400"
6    outlineColor?: string; // default is "#F82400"
7    creatorName?: string; // default is null
8    customData?: object;
9  };
10
11 type Action = {
12   type: 'createRedactions';
13   strategy: 'preset' | 'regex' | 'text';
14   strategyOptions: PresetOption | RegexOptions | TextOption;
15   content?: Content;
16 };
17
18 type PresetOption = {
19   preset:
20     | 'credit-card-number'
21     | 'date'
22     | 'email-address'
23     | 'international-phone-number'
24     | 'ipv4'
25     | 'ipv6'
26     | 'mac-address'
27     | 'north-american-phone-number'
28     | 'social-security-number'
29     | 'time'
30     | 'url'
31     | 'us-zip-code'
32     | 'vin';
33   include_annotations?: boolean;
34   case_sensitive?: boolean;

```

```

35 };
36
37 type RegexOptions = {
38     regex: string;
39     include_annotations?: boolean;
40     case_sensitive?: boolean;
41 };
42
43 type TextOption = {
44     text: string;
45     include_annotations?: boolean;
46     case_sensitive?: boolean;
47 };

```

## Apply Redactions

This applies the redactions created by an earlier `createRedactions` action:

```

1 type Action = {
2     type: 'applyRedactions';
3 };

```



## Output

The `output` object needs to follow this specification:

```

1 type Output = PDFOutput | PDFOutput | ImageOutput | JsonContentOutput;
2
3 type BasePDFOutput = {
4     owner_password?: string;
5     user_password?: string;
6     user_permissions?: UserPermissions[];
7     metadata?: Metadata;
8     labels?: Label[];
9     optimize?: Optimize;
10 };
11
12 type PDFOutput = BasePDFOutput & { type: 'pdf' };
13
14 type PDFOutput = BasePDFOutput & {
15     type: 'pdfa';
16     // Default is "pdfa-1b".
17     conformance?:
18         | 'pdfa-1a'
19         | 'pdfa-1b'

```



```

20 | 'pdfa-2a'
21 | 'pdfa-2u'
22 | 'pdfa-2b'
23 | 'pdfa-3a'
24 | 'pdfa-3u'; // The currently supported conformance levels.
25 vectorization?: boolean; // `true` by default.
26 rasterization?: boolean; // `true` by default.
27 };
28
29 type ImageOutput = {
30   type: 'image';
31   format: 'jpg' | 'jpeg' | 'png' | 'webp';
32   // The default is to render the first page.
33   pages?: {
34     // The page index can be negative, indicating an offset from the last page
35     // being the last page itself.
36     start?: number | 'first' | 'last';
37     end?: number | 'first' | 'last';
38   };
39   // One of width, height, or DPI needs to be specified.
40   width?: number;
41   height?: number;
42   dpi?: number;
43 };
44
45 // Extracts document contents and returns them as a JSON.
46 type JsonContentOutput = {
47   type: 'json-content';
48   // When set to true, extracts document text. Text is extracted via OCR process.
49   plainText: boolean;
50   // When set to true, extracts structured document text. This includes text within
51   structuredText: boolean;
52   // When set to true, extracts key-value pairs detected within the document content.
53   keyValuePairs: boolean;
54   // When set to true, extracts table data from the document.
55   tables: boolean;
56   // Specifies the language to be used for OCR text extraction. Supports the standard
57   language: string;
58 }
59
60 type UserPermissions =
61 | 'printing'
62 | 'modification'
63 | 'extract'
64 | 'annotations_and_forms'
65 | 'fill_forms'
66 | 'extract_accessibility'
67 | 'assemble'
68 | 'print_high_quality';
69
70 // Represents metadata for a PDF.
71 type Metadata = {
72   title?: string;
73   author?: string;
74 };

```



```
75
76 // Represents a label and the pages associated with it.
77 type Label = {
78     pages: integer[]; // 0-based page index.
79     label: string;
80 };
81
82 // Represents the available compression options for the PDF output.
83 type Optimize = {
84     grayscaleText?: boolean; // `false` by default.
85     grayscaleGraphics?: boolean; // `false` by default.
86     grayscaleFormFields?: boolean; // `false` by default.
87     grayscaleAnnotations?: boolean; // `false` by default.
88     disableImages?: boolean; // `false` by default.
89     mrcCompression?: boolean; // `false` by default.
90     linearize?: boolean; // `false` by default.
91     imageOptimizationQuality?: integer; // The range is between 1 and 4, where 1
92 };
```

---

Was this helpful?

☒ YES

☐ NO

---

Questions? [Contact us](#)

