



Web



PSPDFKit Server



WEB > GUIDES > PSPDFKIT SERVER

Generate DOCX and PDF documents



PSPDFKit Server 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 populating Office DOCX templates with data. The resulting DOCX file can then optionally be converted into a PDF document.



Populating DOCX templates with data and converting DOCX files into PDF documents require special licenses. Contact [Sales](#) for more information.

General principles

Word templating consists of the following elements:

- 1 A DOCX file that will be used as the template.
- 2 A template model that contains the placeholder values to replace in the DOCX template.
- 3 Configuration for the template/model.

Template model



ASK AI

The template model must contain:

- 1 A configuration containing both a start and end delimiter. The default delimiters are `{` and `}`, and both can be configured in the request.
- 2 At least one placeholder-value pair. The placeholder name must correspond to the placeholder defined in the DOCX template.

Populating a document

PSPDFKit supports replacing placeholder text strings, automatic reflow, loops, and dynamic tables.

Text replacement

Consider the following DOCX content:

```
1 Hello my name is {name}.
2 There is {more}.
```



Use the following model:

```
1 "model": {
2   "name": "Petey Eff",
3   "more": "lorem ipsum dolor sit amet."
4 }
```



The outcome in the output DOCX document will be:

```
1 Hello my name is Petey Eff.
2 There is lorem ipsum dolor sit amet.
```



Loops

Consider the following DOCX content:

```
1 {ledger}:
2 {#items} {name} {price} {/items}
```

Here, `items` is the name of the loop, and `name` and `price` are placeholders for repetitive elements. Consider the following model:

```
1 "model": {
2   "ledger": "Tom's groceries",
3   "items": [
4     { "name": "A", "price": 10 },
5     { "name": "B", "price": 15 }
6   ]
7 }
```

The outcome in the output DOCX document will be:

```
1 Tom's groceries:
2 A 10
3 B 15
```

Request example with custom delimiters

This example performs dynamic population with custom delimiters (`{{` and `}}`) using a request to the `/api/process_office_template` endpoint:

CURL

HTTP

```
1 curl -X POST http://localhost:5000/api/process_office_template \
2   -H 'Authorization: Token token=<API token>' \
3   -H 'content-type: multipart/form-data' \
4   -F 'document=@/path/to/template.docx'
5   -F 'model={
6     "config": {
7       "delimiter": {
8         "start": "{{",
9         "end": "}}"
10      }
11    },
12    "model": {
```

```
13     "placeholder": "replacement value",
14     "loop-name": [
15         {
16             "placeholder-within-loop": "replacement value",
17             "another-placeholder-within-loop": "replacement value 2"
18         },
19         {
20             "placeholder-within-loop": "another replacement value",
21             "another-placeholder-within-loop": "another replacement value 2"
22         }
23     ]
24 }
25 }' \
26 --output result.docx
```

Was this helpful?

✓ YES

✗ NO

Questions? [Contact us](#)

