

# Efficient Invoice Generation with PyMuPDF

PyMuPDF is a Python library that provides a wide range of features for working with document files. Among the emerging, most useful features of PyMuPDF is its ability to generate professional reports with a minimum of coding effort.

PyMuPDF report generation is based on its Story feature, which is explained in articles [PyMuPDF's New 'Story' Feature Provides Advanced PDF Layout Styling](#) and [How to Layout Articles Using PyMuPDF's Story Feature](#).

We have put all the technical details under the hood, so the developer can focus on the functional aspects of the reports.

In this article, we will create a typical invoice report, where the single items are stored in a database, a JSON file, or a CSV file.

## Background

When we speak of reports, then this usually refers to classifiable types of PDF output, based on data contained in databases. Typical examples include invoices, financial statements or text-oriented articles, possibly with interspersed images.

PyMuPDF's approach to produce a report includes the following steps:

1. Analyze the desired layout of the report pages. There may be a company logo and common header to show on all pages. Or some introductory content, only to be shown on page one or, similarly, some closing text with legal information on the last page. Variable data like single invoice items are normally to be read from some database and reported in tabular format across multiple pages (and to be squeezed in between desired page headers and footers).
2. Create HTML sources that represent the report building blocks defined previously. These HTML sources may have a constant, invariable content, or may contain insertion points for data fetched from databases or other sources.
3. For a more complex building block like a table, the developer will need to define the source of the variable data and typically develop appropriate code to access data storage.
4. Select one of PyMuPDF's predefined report types. Report types are defined as Python classes with a range of properties needed to fulfill its respective purpose. The collection of available reports will be continuously extended to cover new reporting situations.
5. For producing an invoice, in this article we have selected a report type called "FrontMatterReport". It allows us to distinguish between the layout of the first and the remaining pages. In the end, our invoice will have the following layout:
  - a. On page one we show the header, an introductory section on page one called "prolog" and (the start of) the table with the invoice items.
  - b. On page 2 and all subsequent pages, only the header and the (continuation of) the invoice item table will be shown.
  - c. All the layout details like where to put data on the page, selecting adequate column widths for tables or page breaks are either defined by the prepared HTML sources or otherwise automatically determined by PyMuPDF. They are of no concern to the developer.

## Creating an Invoice

Our invoice will show a different layout on page one than on the other pages. All pages however share the header and the invoice table.

To meet these requirements, we will select the report type “FrontMatterReport”.

We will need to define three HTML sources: “header.html”, “prolog.html” and “items.html”. For the header and the prolog, the building block “SimpleBlock” is sufficient.

The invoice items are located in a database and need to be shown by some tabular layout on the pages. Because we cannot know the size of the resulting table beforehand, we choose the “LongTable” building block to represent them. This class supports automatic top row repeat, choice of top row background color and a callback function for supplying table data.

The fields of the items table line are represented by identifiers (HTML “id” tags). The building block “LongTable” needs to know these identifiers in order to fill in values.

Let us have a look at the code representing this preparation work looks:

```
import pathlib # for easy reading of text file content
import sqlite3

import fitz

# import required components from PyMuPDF Report collections
from Reports import FrontMatterReport, LongTable, SimpleBlock

mediabox = fitz.paper_rect("letter-1") # page format: Letter landscape

# Create the overall report object
report = FrontMatterReport(
    mediabox, # the only mandatory parameter
    logo="logo.png", # put this image at the top-left of all pages
)

# Predefined HTML to define the header for all pages
hdr_html = pathlib.Path("header.html").read_bytes().decode()
header = SimpleBlock(html=hdr_html)

# The prolog HTML basically is a skeleton with 4 variables
prolog_html = pathlib.Path("prolog.html").read_bytes().decode()

# After reading the HTML source, we access the content and fill in
# data for the variables.
prolog_story = fitz.Story(prolog_html)
body = prolog_story.body
body.find(None, "id", "supplier").add_text(supplier)
body.find(None, "id", "contact").add_text(contact)
body.find(None, "id", "billto").add_text(billto)
body.find(None, "id", "shipto").add_text(shipto)

# To create the building block, we use the prepared Story this time.
prolog = SimpleBlock(story=prolog_story)
```

When defining the “prolog” block, we did not pass in the HTML source directly.

Instead, we created a **Story** to enable us replacing the variables “supplier”, “contact”, “billto” and “shipto” with corresponding data. Please take a look at the four statements above, which find the variables and replace them with values.

To define our “LongTable” building block, more considerations are required. Let us start by looking at the HTML definition in “items.html” – which is quite simple:

```

<body style="font-family: sans-serif;">
  <table>
    <tr id="header"> /* an "id" is required for "top row repeat"*/
      <th>Line</th>
      <th>H&P ID</th>
      <th>Description</th>
      <th>Part No.</th>
      <th>Qty</th>
      <th>UOM</th>
      <th>Date</th>
      <th>Unit Price</th>
      <th>Total Price</th>
    <tr id="template"> /* required for cloning the row per item */
      <td id="line"></td>
      <td id="hp-id"></td>
      <td id="desc"></td>
      <td id="part"></td>
      <td id="qty"></td>
      <td id="uom"></td>
      <td id="date"></td>
      <td id="uprice"></td>
      <td id="tprice"></td>
    </tr>
  </table>
</body>

```

In essence we define the **top row** of the table and make it identifiable by some “id” (“header” – can be chosen as required). This row will be repeated on every page where parts of the table appear.

The second row is a template for showing the actual item details. This row **must** be named “template”. The “id” values for the row detail fields must be communicated to the building block definition as follows.

Code inside `LongTable` will clone the template row as many times as there are items in the database. Here is the building block definition:

```

# Read the HTML source code for the items table
items_html = pathlib.Path("items.html").read_bytes().decode()

items = LongTable( # generate a table object that can cross page boundaries
  html=items_html, # HTML source
  top_row="header", # identifies the table's top row
  top_row_bg="#aaceeb", # top row background color
  fetch_rows=fetch_rows, # callback to fetch invoice items
  report=report, # pointer to owning report object
)

```

By specifying parameter `top_row` (“header”) we automatically request that every output segment of the table should receive this header line. If the parameter is `None` (the default), top row repetition will not happen.

Parameter `top_row_bg` can be used to specify a background color for the top row. If omitted, any value specified in the HTML source is taken.

The most important parameter is `fetch_rows`. This must be a callable which returns all rows of invoice items. Where these data come from is completely up to this function. It could be an SQL database, a CSV or JSON file, a pandas `DataFrame` or anything else. However, the first of the returned rows must contain the list of id's specified in the "template" row of the HTML source. So, in our example that row **must** be

```
[ "line", "hp-id", "desc", "part", "qty", "uom", "date", "uprice", "tprice" ].
```

Up to this point, we have defined the building blocks that make up our report.


We will now complete the definition of the report itself – by telling it, which building block must be used on which page(s). This is done by two simple statements:

```
# -----  
# We have defined all required building blocks for the report.  
# Now define on which pages they should be used.  
# Report type "FrontMatterReport" supports this by two separate lists  
# of building blocks:  
# One for the first page and one for all other pages.  
# -----  
  
# Attribute 'page0' can contain a different set of stories than other pages  
report.page0 = [header, prolog, items] # page 1 shows 3 building blocks  
report.pages = [header, items] # other pages show the header and item details  
# Done!
```

To actually **generate** the PDF, **only one more statement** is required:

```
# This generates the report and saves it to the given path name.  
report.run("pymupdf-invoice.pdf")
```

Here is an example invoice produced by the above. Building blocks and their names are indicated in red.



header.html

### Standard Purchase Order

Artifex Software, Inc.

PO#	740462660
Revision	0
Buyer Name	McKie, Jorj X.
Email	jorj.x.mckie@outlook.de
Phone	123-456-7890
Order Date	18-AUG-2023
Rig/Whse	252

<b>Supplier</b> Artifex Software, Inc. 39 Mesa Street Suite 108A San Francisco, CA 94129 UNITED STATES	<b>Ship To Address</b> Paul Atreides (Muad'Dib) The Emperor's Palace Arrakeen Planet Arrakis (Dune)
<b>Supplier Contact Info</b> Artifex Software, Inc. 39 Mesa Street Suite 108A San Francisco, CA 94129 UNITED STATES	<b>Bill To Address</b> Jorj X. McKie, Saboteur Extraordinary Bureau of Sabotage Central City Planet Central-Central

Supplier No.	Payment Terms	FOB	Ship Via	Freight Terms
3008208	NET 60 DAYS			

PO Description: A MuPDF Commercial License as a gift for the Emperor's delight.

Line	H&P ID	Description	Part No.	Qty	UOM	Date	Unit Price	Total Price
1	88003146	Enter arbitrary data here, including line breaks as required. May contain address data, or comments of any kind.	X91-01410	12	Each	28-AUG-2023	\$11.48	\$137.76
2	88051507	RAGS, TURKISH, 25 LB BOX Project: 0252.0006 Task: Example Task Inventory Org: Organizational Unit	X70-00076	2	Each	28-AUG-2023	\$29.71	\$59.42

Page 1 of 3

And here follows a glimpse at page 2. Please note that the prolog is no longer included and the items are directly positioned underneath the header, as desired.



header.html

### Standard Purchase Order

Artifex Software, Inc.

PO#	740462660
Revision	0
Buyer Name	McKie, Jorj X.
Email	jorj.x.mckie@outlook.de
Phone	123-456-7890
Order Date	18-AUG-2023
Rig/Whse	252

Line	H&P ID	Description	Part No.	Qty	UOM	Date	Unit Price	Total Price
3	88055335	RAGS, TURKISH, 25 LB BOX Project: 0252.0007 Task: Example Task 2 Inventory Org: Organizational Unit 2	X70-32976	2	Each	28-AUG-2023	\$147.39	\$294.78
4	88076195	Lorem Ipsum is simply dummy text of the printing and typesetting industry.	A91-00006	1	Package	28-AUG-2023	\$130.08	\$130.08
5	88076305	There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.	A72-00015	1	Box	28-AUG-2023	\$489.88	\$489.88
6	88076350	If you are going to use a passage of Lorem Ipsum, you need to	A91-00008	1	Package	28-AUG-2023	\$151.33	\$151.33

## Conclusion

PyMuPDF's new reporting feature provides efficient ways to quickly compose reports based on predefined HTML building blocks – thus making creative use of PyMuPDF's Story class.

We are planning to extend the collection of both the report types and the building blocks in the near future.

Have a look at many more interesting articles on the [Artifex blogging](#) site. Other PyMuPDF resources are our excellent [documentation](#), the [#pymupdf](#) channel on Discord and the interactive, installation-free playground [pymupdf.io](#).