Reports.web Pure Python プログラマーズガイド／APIリファレンス

Reports.web Pure Pythonは、ブラウザー内のWASMを使わず、Pythonだけで帳票を読み込み、データを差し込み、表示・保存・印刷するための帳票エンジンです。デスクトップアプリケーションにも、FastAPIなどのサーバー側Webアプリケーションにも組み込めます。

このガイドでは、次の処理を実装できるところまでを扱います。

- PREPD／PREPDJの帳票定義を読み、業務データを設定する
- 帳票全体を1つのPDFへ出力する
- SVG／PNGをページ単位で出力する
- 完成した帳票をPREPE／PREPEJとして保存し、再表示・再印刷する
- デスクトップでプレビュー・印刷する
- Web APIからPDF、ページ単位のSVG、PREPEJを配信する

画面の使い方は、Pure Python デザイナー操作ガイドとPure Python プレビュアー操作ガイドを参照してください。

目次

1. 最初に理解する3つのデータ 4

2. 開発を始める 4

2.1 動作環境とインストール 4

2.2 主な公開API 5

3. 最初の帳票プログラム 5

4. 印刷データを設定する 6

4.1 固定項目と明細 6

4.2 DBの検索結果を対応付ける 6

4.3 画像を差し替える 6

5. PDF、SVG、PNGを出力する 7

5.1 PDFは帳票全体を1ファイルへ出力する 7

5.2 SVGはページ単位 7

5.3 PNGもページ単位 7

5.4 ページ数を先に調べる 8

6. PREPEJへ保存し、もう一度出力する 8

7. デスクトップアプリケーションで使う 9

7.1 ページ画像を独自画面へ表示する 9

7.2 プリンターへ出力する 9

7.3 デスクトップの操作UIを外側から隠す 9

8. Webアプリケーションで使う 10

8.1 付属Webサンプル 10

8.2 アプリ側からプレビュアーを操作する 11

9. 長い処理を中止し、進捗を受け取る 13

10. キャッシュを利用する 14

11. モデルAPIリファレンス 14

11.1 ReportDocument 14

11.2 ReportObject 14

11.3 RenderOptions 15

12. 読み書きAPIリファレンス 15

13. エラー処理と安全性 16

14. 目的別の早見表 16

15. 付属サンプルを実行する 17

16. ライセンス 17

1. 最初に理解する3つのデータ

帳票は「レイアウト」と「値」を分けて扱います。

データ	Pythonでの表現	内容
帳票定義	`ReportDocument`	用紙、文字、罫線、画像、バーコード、繰返しなどのレイアウト
印刷時の値	`dict[str, Any]`	顧客名、日付、金額、商品明細など、帳票へ差し込む値
完成した印刷データ	`PrintDataPackage`	帳票定義と値をページ単位で保持し、後から再表示・再印刷できるデータ

拡張子	内容
`.prepdj`	JSON形式の帳票定義
`.prepd`	従来形式の帳票定義
`.prepej`	JSON形式の完成した印刷データ
`.prepe`	従来形式の完成した印刷データ

基本の流れは、`load_report()`で定義を読み、Pythonの辞書へ値を入れ、`Renderer`で出力する、という3段階です。

2. 開発を始める

2.1 動作環境とインストール

- Python 3.10以降（製品の検証環境はPython 3.13）
- PDF、SVG、PNGをサーバーで生成するだけなら、画面を開く必要はありません
- デザイナー、プレビュアー、プリンター出力にはPySide6を使用します

製品ルートで仮想環境を作り、開発モードでインストールする例です。

```
cd C:\Pao\Pao.Reports.Python
py -3.13 -m venv .venv313
.\.venv313\Scripts\python.exe -m pip install -e ".[test]"
```

配布ZIPでは、同梱READMEに記載されたPythonと依存パッケージを使用してください。

2.2 主な公開API

通常はpao_reportsから次の名前をインポートします。

API	役割
load_report() / save_report()	帳票定義ファイルの読み書き
loads_report() / dumps_report()	文字列と帳票定義の相互変換
ReportDocument / ReportObject	帳票と配置オブジェクトのモデル
ObjectType / BarcodeKind	オブジェクト種別とバーコード種別
Renderer / RenderOptions	PDF、SVG、PNG、印刷の出力
page_count()	帳票の物理ページ数を取得
save_print_data() / load_print_data()	PREPE／PREPEJの読み書き
RenderControl / RenderCancelled	進捗通知と処理中止
PreviewCache	ページ画像とSVGの再利用

3. 最初の帳票プログラム

次の例は、帳票定義へ値を差し込み、帳票全体を1つのPDFへ保存します。

```
from pathlib import Path

from pao_reports import Renderer, load_report

definition = load_report(Path("reports/invoice.prepdj"))
values = {
    "顧客名": "株式会社サンプル",
    "請求日": "2026年9月14日",
    "合計金額": "120,000",
}

output = Path("output/invoice.pdf")
output.parent.mkdir(parents=True, exist_ok=True)
Renderer().render_pdf(definition, values, output)
```

辞書のキーには、デザイナーでオブジェクトのデータ項目に設定した名前を指定します。
ReportObject.bindingとキーが一致すると、その値が表示されます。

4. 印刷データを設定する

4.1 固定項目と明細

顧客名や発行日のような値は、そのまま設定します。繰返し明細はリストで渡します。

```
values = {
    "伝票番号": "INV-2026-0012",
    "顧客名": "株式会社サンプル",
    "商品名": ["商品A", "商品B"],
    "数量": [2, 1],
    "単価": ["50,000", "20,000"],
}
```

同じ添字の値が同じ明細行に対応します。帳票定義側の`repeat`、`interval_y`などに従って繰り返され、用紙を越えた分は次ページになります。

日付、金額、単位など、業務上の表示形式を固定したい場合は、Python側で文字列へ整形してから渡すと意図が明確になります。

4.2 DBの検索結果を対応付ける

```
rows = connection.execute(
    "select product_name, quantity, unit_price from invoice_line "
    "where invoice_id = ? order by line_no",
    (invoice_id,),
).fetchall()

values = {
    "商品名": [row[0] for row in rows],
    "数量": [row[1] for row in rows],
    "単価": [f"{row[2]:,.0f}" for row in rows],
}
```

DBの列名を帳票へ暗黙に渡すのではなく、この箇所で帳票項目名へ明示的に対応付けると、DB変更の影響を限定できます。

4.3 画像を差し替える

画像オブジェクトに`binding`がある場合、そのキーへ画像ファイルのパスを渡せます。

```
values["社印画像"] = str(Path("images/company-seal.png").resolve())
```

Web APIの利用者から任意のパスやURLを受け取って、そのまま画像として開かないでください。アプリケーション管理下の画像へ対応付ける設計にします。

5. PDF、SVG、PNGを出力する

5.1 PDFは帳票全体を1ファイルへ出力する

```
from pao_reports import Renderer, RenderOptions

renderer = Renderer(RenderOptions(dpi=150))
renderer.render_pdf(definition, values, Path("output/report.pdf"))
```

`render_pdf()`は、繰返しで生じたページを含む帳票全体を、1つの複数ページPDFへ出力します。`path`を省略するとPDFの`bytes`を返します。

5.2 SVGはページ単位

```
renderer.render_svg_page(
    definition, 0, values, Path("output/page-1.svg")
)
```

ページ番号は0から始まります。全ページをページ別ファイルへ保存する場合は、出力ディレクトリを指定します。

```
paths = renderer.render_svg_pages(
    definition, values, Path("output/svg-pages")
)
```

ファイル名は`page-1.svg`、`page-2.svg`のようになります。`render_svg()`も`RenderOptions.page`で指定した1ページだけを返すAPIです。SVGが帳票全体を1ファイルへまとめるAPIではない点に注意してください。

5.3 PNGもページ単位

```
images = renderer.render_png_pages(
    definition, values, Path("output/png-pages")
)
```

PNGは`page-1.png`、`page-2.png`のようにページ単位で保存され、戻り値はPillowの画像オブジェクトのリストです。1ページだけなら`RenderOptions(page=0)`と`render_png()`を使用できます。

5.4 ページ数を先に調べる

```
from pao_reports import page_count

count = page_count(definition)
for page_index in range(count):
    renderer.render_svg_page(
        definition,
        page_index,
        values,
        Path(f"output/page-{page_index + 1}.svg"),
    )
```

6. PREPEJへ保存し、もう一度出力する

PREPEJは帳票定義と値を含む自己完結した印刷データです。元のPREPDJやDBがなくても、保存した時点の帳票を再表示できます。

```
from pao_reports import load_print_data, save_print_data

saved = save_print_data(
    definition, values, Path("output/invoice.prepej")
)

package = load_print_data(saved)
page = package.pages[0]
if page.definition is None:
    raise ValueError("帳票定義を含む印刷データが必要です")

Renderer().render_pdf(
    page.definition, page.values, Path("output/invoice-again.pdf")
)
```

`load_print_data()`は`PrintDataPackage`を返し、その`pages`に`PrintedPage`が入ります。複数の`PrintedPage`を持つファイルは、各ページの`definition`と`values`を組にしたまま扱ってください。付属プレビュアーもこの組を保って再表示します。

`.prepe`を指定すれば従来XML形式、`.prepej`を指定すればJSON形式で保存します。

7. デスクトップアプリケーションで使う

7.1 ページ画像を独自画面へ表示する

```
from pao_reports import QtPreviewSurfaceRenderer

surface = QtPreviewSurfaceRenderer().render_page(
    definition, values, 0, dpi=96, background="#FFFFFF"
)
qimage = surface.image
```

`QtPreviewSurfaceRenderer.render_page()`は`QtPageSurface`を返します。`image`は`QImage`、`width`と`height`は生成画像のピクセル寸法です。用紙寸法は元の`ReportDocument.width_mm`と`height_mm`を参照します。Qtを使わない画面では、`Renderer.render_png()`が返すPillow画像を利用できます。

付属プレビュアーの開く、保存、検索、ページ移動、表示倍率についてはプレビュアー操作ガイドを参照してください。

7.2 プリンターへ出力する

```
from PySide6.QtGui import QGuiApplication
from pao_reports import Renderer

app = QGuiApplication.instance() or QGuiApplication([])
Renderer().print_document(definition, values)
```

印刷はPySide6のQtPrintSupportを使用し、呼出側がメインスレッド上の`QGuiApplication`を所有している必要があります。プリンター名を指定する場合は`printer_name=`を使います。独自の`QPrinter`を渡す場合は`printer=`を使います。

```
Renderer().print_document(
    definition, values, printer_name="Office Printer"
)
```

`printer=`と`printer_name=`を同時には指定できません。物理印刷の可否はOS、ドライバー、プリンター設定にも依存するため、実機で確認してください。

7.3 デスクトップの操作UIを外側から隠す

02／04 Clientに同梱した `PreviewWindow` を利用する場合は、QtのUIスレッドから次のように操作できます。`preview` は作成済みの `PreviewWindow` です。

```
preview.set_preview_chrome_visible(False)  # 帳票だけにする
preview.set_preview_chrome_visible(True)   # 全ツールバー・メニュー・ステータス・ページ一覧を表示
```

通常の初期表示は変わりません。設定ボタンも隠れるので、再表示はアプリ側のボタンから行います。これは同梱サンプルWindowの公開メソッドであり、エンジンSDKの独立したUIコントロールではありません。OSのウィンドウ枠は残ります。既存の `set_toolbar_visible` は補助ツールバーだけを切り替えるため、帳票だけにする場合は上のメソッドを使用してください。Web用APIとは分けて考えます。

8. Webアプリケーションで使う

Pure Python版は、サーバーで帳票を生成し、PDFまたはページ単位のSVG／PNGをHTTPレスポンスとして返します。ブラウザーの中でPythonエンジンを実行する方式ではありません。

FastAPIでPDFを返す最小例です。

```
from fastapi import FastAPI, Response
from pao_reports import Renderer, load_report

app = FastAPI()
definition = load_report("reports/invoice.prepdj")

@app.get("/api/invoice.pdf")
def invoice_pdf() -> Response:
    values = {"顧客名": "株式会社サンプル"}
    body = Renderer().render_pdf(definition, values)
    return Response(body, media_type="application/pdf")
```

ページ単位のSVGを返す例です。

```
from fastapi import HTTPException
from pao_reports import Renderer, page_count

@app.get("/api/invoice/{page}.svg")
def invoice_svg(page: int) -> Response:
    if page < 0 or page >= page_count(definition):
        raise HTTPException(404, "ページがありません")
    body = Renderer().render_svg_page(definition, page, {})
    return Response(body, media_type="image/svg+xml")
```

公開APIでは、認証・認可、入力文字数、明細件数、要求サイズ、生成時間、同時実行数、最大ページ数を制限してください。

8.1 付属Webサンプル

```
.\.venv313\Scripts\pao-reports-web.exe
```

既定では`http://127.0.0.1:8765/`で起動します。主なAPIは次のとおりです。

API	内容
`GET /api/samples`	公開サンプル一覧
`GET /api/reports/{id}/svg?page=0`	指定ページのSVG
`GET /api/reports/{id}/pdf`	帳票全体のPDF
`GET /api/reports/{id}/data`	自己完結PREPEJ
`POST /api/print-data/svg?page=0`	PREPEJから指定ページのSVGを生成
`POST /api/print-data/pdf`	PREPEJから帳票全体のPDFを生成
`GET /health`	稼働状態とエンジン識別子

SVG・PDF応答の`X-Reports-Engine: PurePython`で、Pure Pythonエンジンが生成したことを確認できます。

8.2 アプリ側からプレビュアーを操作する

帳票だけを業務画面に置き、PDFや検索は自分のアプリのボタンから操作したい場合に使うインターフェースです。**ツールバーを隠しても、プレビュアーの機能は停止しません。** 設定ボタンまで隠せるため、再表示用のボタンはプレビュアーの外側に置きます。

03「全帳票をWebでプレビュー」と公開デモは、この使い方の実例です。最初はプレビューだけを表示し、帳票選択の横の「ツールバーを表示」を押すとPDF・検索・設定などを表示します。もう一度押すと、設定ボタンとステータスも含めて非表示になります。

表示・非表示を切り替えるコード

以下は03サンプルのプレビュアーを読み込んだ後に置く、アプリ側のコードです。サンプルの `native-previewer.js`、`native-previewer.css`、画面の初期化部分を一緒に利用してください。03を編集する場合は、既存の同じボタン・切り替え処理を置き換え、二重に追加しないでください。

```
<!-- プレビュアーの外側に置くアプリ側のボタン -->
<button type="button" id="togglePreviewToolbar"
        aria-controls="nativePreviewer" aria-expanded="false">
  ツールバーを表示
</button>
```

```
// native-previewer.js の読み込み後に実行します。
const previewer = window.ReportsWebNative;
const button = document.getElementById("togglePreviewToolbar");

function updateButton() {
  const visible = previewer.getState().chrome.toolbar;
  button.textContent = visible
    ? "ツールバーを非表示" : "ツールバーを表示";
  button.setAttribute("aria-expanded", String(visible));
}

button.addEventListener("click", () => {
  if (previewer.getState().chrome.toolbar) previewer.hideAll();
  else previewer.showAll();
});
const unsubscribe = previewer.subscribe("chrome-changed", updateButton);

previewer.hideAll(); // 初期画面は帳票だけ
updateButton();
// アプリ側の画面を破棄する場合は unsubscribe() で購読を解除します。
```

03サンプルでは、プレビュアーを初期化する前の `ReportsNativeConfig` に `externalChromeControl: true` を指定しています。この設定では、内部設定から全非表示にした場合も、設定を戻すための小さなボタンをプレビュアー内に残しません。必ず上のような外部ボタンを用意してください。`chrome-changed` を購読すると、内部設定から切り替えた場合も外部ボタンの文字が追従します。

ツールバーがなくても、PDF・検索・ページ移動を使える

```
await previewer.showPdfPreview(); // サーバーへPDFを要求し、同じ表示領域へ表示
await previewer.showSvgPreview(); // 通常の印刷プレビューへ戻る
previewer.goToPage(2);            // ページ番号は1から
previewer.setZoom("auto");       // 表示領域に合わせた倍率
previewer.search("請求");         // 通常の印刷プレビュー内を検索
```

ここでの `showSvgPreview()` はAPI名であり、利用者に見せるボタン名は「印刷プレビュー」で構いません。PDF表示中の検索などはブラウザー内蔵PDFビューアーの機能と区別します。非表示のままPDF表示へ切り替えることもできます。プレビューを最初に開いただけではPDFを要求しません。

API	用途・注意
`hideAll()`	設定・ステータスを含む操作UIを一括非表示にする。開いている設定ダイアログも閉じる
`showAll()`	全コマンドを表示する。以前の個別表示設定を復元する操作ではない
`getState()`	現在のページ・倍率・表示設定などの状態を取得する

API	用途・注意
`subscribe("chrome-changed", handler)`	表示設定の変更を受け取る。戻り値は購読解除用関数
`setChrome({commands: {print: false}})`	印刷など特定のコマンドだけを隠す
`showPdfPreview()` / `showSvgPreview()`	PDF表示／通常プレビューへ切り替える。切り替え処理は `await` で待てるが、ブラウザー内蔵PDFビューアーの読込完了を保証するものではない
`print()` / `savePrintData()` / `openPrintData()`	印刷・PREPEJ保存・PREPEJを開く。自分のボタンのクリックから呼び出す

非表示の対象はReports.webの操作UIです。PDF表示中にブラウザー内蔵PDFビューアーが表示する独自のボタンは、このAPIの制御対象ではありません。

このインターフェースは、3つのNative Webサンプルで使うブラウザー側の `ReportsWebNative` です。サーバーの帳票エンジンのクラスや、デスクトップ用Windowとは別です。WASM版にも一括表示・非表示の機能がありますが、初期化方法やAPIの所属はWASM版のガイドを参照してください。

9. 長い処理を中止し、進捗を受け取る

```
from pao_reports import RenderCancelled, RenderControl, Renderer

def progress(done: int, total: int) -> None:
    print(f"{done}/{total}")

control = RenderControl(progress=progress)
renderer = Renderer(control=control)

try:
    renderer.render_pdf(definition, values, Path("output/report.pdf"))
except RenderCancelled:
    print("帳票生成を中止しました")
```

別スレッドなどから`control.cancel()`を呼ぶと、次のチェックポイントで`RenderCancelled`が送出されます。同じ`RenderControl`を再利用すると中止状態も引き継がれるため、処理単位で作成してください。

10. キャッシュを利用する

```
from pao_reports import PreviewCache, Renderer

cache = PreviewCache(limit=16)
renderer = Renderer(cache=cache)

image = renderer.render_png(definition, values)
same_image = renderer.render_png(definition, values)
cache.clear()
```

同じ帳票定義、データ、描画設定のプレビューを繰り返す場合に利用できます。変更可能なデータを複数リクエストで共有せず、業務要求ごとに辞書を新しく作ってください。

11. モデルAPIリファレンス

11.1 `ReportDocument`

主な属性は次のとおりです。寸法の単位はmmです。

属性	内容
`title`	帳票名
`width_mm`, `height_mm`	用紙の幅と高さ
`paper_size`	A4などの用紙名
`grid_mm`	デザイナーのグリッド間隔
`show_grid`, `snap_to_grid`	グリッド表示と吸着
`default_font_family`, `default_font_size_pt`	既定フォント
`objects`	`ReportObject`の一覧
`extensions`	互換性維持用の拡張情報
`to_dict()`	JSON化できる辞書へ変換

11.2 `ReportObject`

分類	主な属性
種別と名前	`type`, `name`, `binding`, `text`

分類	主な属性
位置と大きさ	x, y, width, height, rotation
繰返し	repeat, repeat_x, repeat_xy_both, interval_x, interval_y
文字	font_family, font_size_pt, bold, italic, underline, strikeout, text_vertical, text_elastic
配置	horizontal_alignment, vertical_alignment
色と枠	text_color, fill_color, fill_enabled, stroke_color, stroke_width, stroke_style
図形	fill_style, hatch_density, line_type, line_thickness, corner_radius
画像	image_path, image_data_base64, image_size_mode, image_alignment, image_reverse
バーコード	barcode_kind, barcode_draw_mode, barcode_text_visible, barcode_justify, qr_error_correction, qr_version
状態	visible, locked, extensions

`resolved_text(data, repeat_index)`は、`binding`に対応する値を取得します。通常は`Renderer`が内部で呼ぶため、業務コードから直接呼ぶ必要はありません。

11.3 `RenderOptions`

属性	既定値	内容
dpi	150	PNG描画などの解像度
background	#FFFFFF	ページ背景色
page	0	render_png()、render_svg()で出力する0始まりのページ

12. 読み書きAPIリファレンス

```
from pao_reports import dumps_report, loads_report, save_report

text = dumps_report(definition, format="json", indent=2)
copy = loads_report(text, format="json", source_name="invoice.prepdj")
save_report(copy, Path("output/invoice.prepdj"))
```

`load_report()`と`save_report()`は拡張子に応じてJSONまたは従来XMLを扱います。文字列APIでは`format="json"`または`format="prepd"`を明示できます。

入力ファイルは信頼できるものに限定し、公開アップロードではファイルサイズ、オブジェクト数、画像データ量を検証してください。

13. エラー処理と安全性

```
from json import JSONDecodeError
from pao_reports import RenderCancelled

try:
    definition = load_report(definition_path)
    Renderer().render_pdf(definition, values, output_path)
except FileNotFoundError:
    # 帳票定義や参照画像が見つからない
    ...
except (ValueError, JSONDecodeError):
    # 入力形式や値が不正
    ...
except RenderCancelled:
    # 利用者またはアプリケーションが中止した
    ...
except OSError:
    # 読み書き、保存先、OS資源に関する失敗
    ...
```

- 入力された任意のローカルパスやURLを画像として参照させないでください。
- 出力先をアプリケーション管理下に限定してください。
- 一時ファイルは`TemporaryDirectory`などで確実に破棄してください。
- Web APIでは内部例外をそのまま応答せず、要求IDとともにサーバーログへ記録してください。
- 帳票で使うフォントを本番環境にも配置し、文字幅、改行、PDF表示を確認してください。

14. 目的別の早見表

やりたいこと	参照章	主なAPI
最初のPDFを作る	3、5.1	`load_report`, `Renderer.render_pdf`
明細を差し込む	4.1	`dict`, `ReportObject.binding`
ページ別SVGを作る	5.2	`render_svg_page`, `render_svg_pages`
ページ別PNGを作る	5.3	`render_png`, `render_png_pages`
完成帳票を保存・再読込する	6	`save_print_data`, `load_print_data`
独自プレビューへ組み込む	7.1	`QtPreviewSurfaceRenderer`
プリンターへ出力する	7.2	`Renderer.print_document`

やりたいこと	参照章	主なAPI
Webから帳票を返す	8	`Renderer`, FastAPI `Response`
処理を中止する	9	`RenderControl`, `RenderCancelled`
モデル属性を確認する	11	`ReportDocument`, `ReportObject`

15. 付属サンプルを実行する

デスクトップサンプルとCLIを起動します。

```
cd C:\Pao\Pao.Reports.Python
$env:PYTHONPATH='engine;.'
.\.venv313\Scripts\python.exe -m samples --output sample-output
.\.venv313\Scripts\python.exe -c "from samples.gui import main; raise SystemExit(main())"
```

最初は`S01`のクイックスタートでPDF、SVG、PNGを確認し、次に請求書や見積書の定義と項目名へ置き換えてください。

16. ライセンス

本製品は商用ライセンスです。評価、開発用PC、サーバー配置、再配布、組み込み利用の条件は、製品に付属する使用許諾契約書を確認してください。