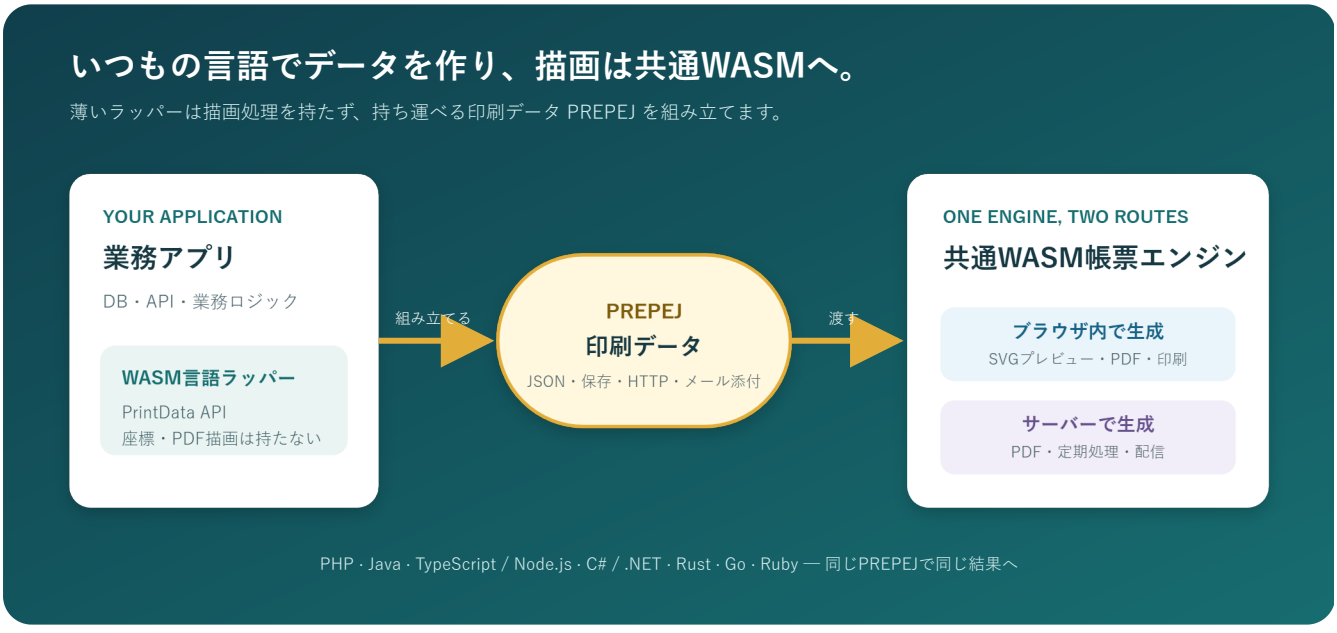
# Reports.web TypeScript／Node.js WASM言語ラッパー プログラマーズガイド／APIリファレンス

Reports.webのWASM言語ラッパーは、TypeScript／Node.jsの業務データから、共通の印刷データPREPEJを組み立てるための小さなライブラリです。帳票の座標計算やPDF描画を各言語へ重複実装せず、ブラウザまたはサーバーの共通WASM帳票エンジンへ任せます。

この一冊では、初めて利用する方が次のプログラムを作れるようになることを目標にしています。

- TypeScript／Node.jsから帳票定義PREPDJへ業務データを流し込む
- 完成した印刷データPREPEJをJSON、ファイル、HTTPで受け渡す
- ブラウザでSVGプレビュー・印刷・PDF生成を行う
- サーバー側の共通WASMエンジンでPDFを生成・配信する
- TypeScript／Node.jsラッパーの主要APIを調べる



# 目次

# 1. 最初に知っておくこと

## 1.1 「WASM言語ラッパー」とは

本書では、各言語でPREPEJを組み立てる補助ライブラリを**WASM言語ラッパー**と呼びます。実装の中心は`PrintData`です。ラッパー自身はWASMを直接ロードせず、SVGやPDFも描きません。出力形式を共通JSONに揃えることで、業務処理と描画エンジンを疎結合にします。

これはPure Java／Pure Python／WPF・.NETの言語別エンジンとは異なる方式です。Pure系は各言語の帳票エンジンを直接呼びます。WASM方式は7言語が同じWASMエンジンとブラウザUIを共有します。

## 1.2 帳票ができるまで

1. デザイナーで帳票定義 `.prepdj` を作ります。
2. TypeScript／Node.jsの`PrintData`へPREPDJを設定します。
3. DBやAPIから取得した値を、デザイン上のオブジェクト名へ設定します。
4. 1ページずつ確定し、`.prepej`を作ります。
5. PREPEJをブラウザのプレビュアー、またはサーバーWASMへ渡します。
6. 共通エンジンがSVGプレビュー、PDF、印刷を担当します。

## 1.3 二つのファイル

| 拡張子 | 呼び名 | 役割 |
|---|---|---|
| `.prepdj` | 帳票定義 | 用紙、文字、罫線、画像、バーコード、繰返しなどのデザイン |
| `.prepej` | 印刷データ（Print Document） | 帳票定義とページごとの値・属性をまとめた、再表示・再印刷可能な文書 |

PREPEJは単なる一時レスポンスではありません。ファイル保存、再読込、メール添付、REST API、サーバー間転送、後日の再印刷に同じ文書を使えます。

# 2. サンプルを動かす

基準環境は **Node.js 22+** です。リポジトリ内の実装とサンプルを同じ版で使ってください。

```
cd C:\Pao\Pao.Reports.Web\samples\node
npm ci
npm run build
npm start
```

ブラウザで `http://127.0.0.1:8092/demo/reports.web/samples/node/` を開きます。

収録サンプルでは、単純帳票、10の倍数、郵便番号一覧、見積書、請求書、商品一覧、名刺、デザイン機能見本を確認できます。最初は請求書を開き、ブラウザ生成とサーバー生成のPDFが同じページになることを確かめてください。

## 3. 最初の一枚を作る

```
import { readFile } from "node:fs/promises";
import { PrintData } from "./PrintData.js";

const definition = JSON.parse(await readFile("invoice.prepdj", "utf8"));
const print = new PrintData()
  .setDefinition(definition)
  .pageStart()
  .setValue("請求番号", "INV-2026-001")
  .setValue("お客様名", "パオ商事株式会社")
  .pageEnd();

response.writeHead(200, { "Content-Type": "application/json; charset=utf-8" });
response.end(print.toJson());
```

呼出順は **定義を設定 → ページを開始 → 値を設定 → ページを確定 → JSONを取得** です。未完了のページがある状態ではPREPEJを取得できません。この制約により、途中までの帳票を誤って配信することを防ぎます。

## 4. PrintData APIリファレンス

実装ファイルは `samples/node/src/PrintData.ts` です。次のAPI名は実装に合わせています。

| API | 用途 |
| --- | --- |
| `setDefinition` | 帳票定義（PREPDJ）を設定します。設定値は複製され、呼び出し元の変更から守られます。 |
| `pageStart` | 新しい印刷ページを開始します。別の定義を渡すと、表紙から明細などへ様式を切り替えられます。 |
| `setValue` | 名前で指定した動的オブジェクトへ値を設定します。indexは0から始まる繰返し位置です。 |
| `setRepeatedValue` | 縦横二方向に繰り返すオブジェクトへ、X・Y位置を指定して値を設定します。 |
| `setTypedValue` | DynamicText、DynamicImage、DynamicBarcodeなど、値の種類を明示して設定します。 |
| `setImage` | 画像URLまたはdata URIを動的画像へ設定します。 |
| `setBarcode` | バーコード／二次元コードへ渡す文字列を設定します。 |
| `changeAttributes` | 位置、幅、色、フォント、配置などをページ単位で動的に変更します。 |
| `changeRepeatedAttributes` | 繰返し位置を含めて動的属性を変更します。 |

| API | 用途 |
| --- | --- |
| `pageEnd` | 現在のページを確定し、印刷データへ追加します。 |
| `toObject / toJson` | 完成したPREPEJをオブジェクトまたはJSON文字列として取得します。 |
| `save` | 完成したPREPEJを一時ファイル経由で安全に保存します。 |

### 4.1 値とインデックス

値には文字列、数値、真偽値、nullの単純値を渡します。配列や任意オブジェクトを値として渡さず、帳票オブジェクトごとに明示的に対応付けてください。繰返しのindex、indexX、indexYは0から始まり、負数はエラーです。

`drawing=false`は、その繰返し位置を描画しない場合に使います。同じ名前・同じ位置への設定はサンプルの規約に従い、意図しない重複を作らないようにします。

### 4.2 ページごとに定義を切り替える

表紙と本文、途中から異なる一覧形式などでは、ページ開始時に別のPREPDJを渡します。座標をプログラムへ直接書くのではなく、デザイナーで完成させた定義をページへ結び付けます。定義を変更する前に現在のページを必ず確定してください。

## 5. 明細と改ページ

```
const print = new PrintData().setDefinition(definition);
for (let offset = 0, page = 1; offset < details.length; offset += 20, page++) {
  print.pageStart().setValue("ページ", page);
  details.slice(offset, offset + 20).forEach((d, row) =>
    print.setValue("品番", d.code, row)
         .setValue("品名", d.name, row)
         .setValue("金額", d.amount, row));
  print.pageEnd();
}
```

1ページの行数は帳票定義と業務仕様から決めます。最終ページの空行を描画するか、明細ごとに罫線を出すかもプログラム側で明示します。ページ番号はPREPEJのページ順と、帳票へ表示する業務上のページ番号を分けて考えます。

## 6. 画像・バーコード・動的属性

```
print.setImage("角印", stampDataUri)
  .setBarcode("商品QR", "https://example.jp/items/123")
  .changeAttributes("合計金額", {
    bold: true, fontSize: 16, horizontalAlignment: "Right"
  });
```

画像は閲覧先でも取得できる管理下のURL、または許容サイズのdata URIにします。サーバーPDF生成では、任意の外部URLをそのまま取得するとSSRFにつながります。固定画像は許可リストまたはアプリ管理下のファイルから解決してください。

動的属性では、名前・型・値を無検証のリクエストから直接渡さないでください。変更できる属性をアプリ側で限定し、数値範囲、色、配置、フォント名を検証します。

## 7. ブラウザへ渡して表示する

TypeScript／Node.js APIはPREPEJを`application/json; charset=utf-8`で返します。ブラウザ側は共通のReports Web APIへ渡します。

```
<iframe id="reportsViewer" src="/reports-web/previewer.html"></iframe>
<script>
const viewer = document.querySelector('#reportsViewer');
viewer.addEventListener('load', async () => {
  const response = await fetch('/api/invoices/123/print-data');
  if (!response.ok) throw new Error(`HTTP ${response.status}`);
  await viewer.contentWindow.ReportsWeb.setPrintData(await response.json());
});
</script>
```

親画面とプレビュアーは同一オリジンに置くのが最も簡単です。別オリジンで`postMessage`を使う場合は、送信元と送信先のoriginを固定し、`*`を本番環境で使わないでください。

プレビュアー側では`setZoom("auto")`、ページ移動、検索、印刷、`exportPdf()`、`savePrintData()`を利用できます。独自画面へ表示面だけを組み込む場合は`hideAll()`でツール群を隠し、同じ機能を外側のボタンから呼び出します。

## 8. PDFを作る二つの場所

### 8.1 ブラウザで作る

PREPEJを利用者のブラウザへ渡し、ブラウザ内のWASMがPDFを生成します。サーバーのPDF描画負荷を抑え、プレビューからそのまま保存・印刷する操作に向きます。機密データをブラウザへ渡してよいかは、業務の認可設計で判断します。

### 8.2 サーバーで作る

Node.jsから同一プロセスまたは共通Node/WASMサービスへPREPEJを渡します。 定期処理、メール添付、保存、監査、画面を開かない帳票生成に向きます。

```
POST /render/pdf
Content-Type: application/json

<PREPEJ JSON>
```

成功時は`application/pdf`を返します。入力・出力サイズ、同時実行数、処理時間を制限し、失敗時の途中PDFを保存・配信しないでください。ブラウザ生成とサーバー生成は同じPREPEJを入力にするため、業務データ組立てを二重実装する必要はありません。

## 9. PREPEJを保存・再利用する

`save`は完成したPREPEJをUTF-8 JSONとして保存します。保存前にページをすべて確定してください。ファイル名には`.prepej`を使い、アップロード時には拡張子だけで信用せず、Format、Version、Definition、Pages、サイズ、ページ数を検証します。

保存した文書は、Webプレビュアーの「開く」またはドラッグ＆ドロップで再表示できます。HTTPで受け渡す場合もファイルと同じJSONです。再印刷の証跡として保管する場合は、業務データの保持期間、暗号化、アクセス権、削除規則を定めます。

## 10. Web APIとデータベース

DB列と帳票オブジェクト名はコードで明示的に対応付けます。SQLはプレースホルダーを使い、利用者が参照できる伝票かを取得前に確認します。日付、通貨、端数処理は帳票へ渡す前に業務規則で確定します。

機密帳票のレスポンスには`Cache-Control: no-store`を設定します。内部例外、接続文字列、ファイルパスを利用者へ返さず、要求IDを安全なサーバーログへ残します。帳票定義や画像は可能なら公開ルート外へ置きます。

## 11. テストとリリース確認

ラッパー単体の基準コマンドは `npm test` です。

最低限、次を確認します。

- Formatが`Reports.net PrintData`、Versionが`1.0`である
- PREPDJを設定せずにページを開始するとエラーになる
- ページを閉じずに次のページを開始できない
- 日本語、改行、数値、真偽値、nullが壊れない
- 明細の先頭・最終行、改ページ前後に欠落や重複がない

- ページ途中の帳票定義切替が、完了済みページを変更しない
- 画像・角印、バーコード、動的属性がSVGとPDFへ反映される
- 保存したPREPEJを開き直すと同じページ・内容になる
- ブラウザ生成とサーバー生成のページ数・主要表示が一致する

ラッパー変更時はWASM描画エンジン全体を毎回再試験するのではなく、生成PREPEJが共通仕様と基準サンプルに一致することを中心に確認します。WASM、プレビュアー、フォント、画像処理を変更した場合は、別途描画回帰試験が必要です。

## 12. よくある問題

| 症状 | 確認すること |
| --- | --- |
| 帳票が空になる | PREPDJ上のオブジェクト名と`setValue`の名前、indexを確認します。 |
| 2ページ目がない | ページごとにpage start／page endが対になっているか確認します。 |
| 画像が×になる | URLの到達性、data URI、CORS、サーバー側許可リストを確認します。 |
| 日本語が欠ける | 配置したフォント、フォントマップ、WASMとWeb資源の版を確認します。 |
| WASMが読み込めない | `.wasm`のURL、`application/wasm`、キャッシュ、同一リリースのJSを確認します。 |
| PDFだけ失敗する | 入力サイズ、画像サイズ、タイムアウト、同時実行制限、サーバーログの要求IDを確認します。 |

## 13. 配置・更新・ライセンス

ラッパー、プレビュアー、JavaScript／TypeScript、CSS、WASMは同じReports.web配布版から取り出してください。HTMLだけ、またはラッパーだけを別バージョンへ差し替えると、PREPEJや公開APIの互換性を判断できなくなります。

試用版と製品版ではWASMが異なります。購入者用WASMを公開サンプルやソース管理へ混入させないでください。正式な動作環境、使用許諾、更新履歴、問い合わせ先は共通製品マニュアルを正本とします。開発ライセンスは開発用PC 1台につき1ライセンスです。