

Reports.web PHP 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、PHP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。

- PHP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
- PHPラッパーの主要APIを調べる

いつもの言語でデータを作り、描画は共通WASMへ。

薄いラッパーは描画処理を持たず、持ち運べる印刷データ PREPEJ を組み立てます。



PHP・Java・TypeScript / Node.js・C# / .NET・Rust・Go・Ruby ― 同じPREPEJで同じ結果へ

目次

1. 最初に知っておくこと	3
1.1 「WASM言語ラッパー」とは	3
1.2 帳票ができるまで	3
1.3 二つのファイル	3
2. サンプルを動かす	3
3. 最初の一枚を作る	4
4. PrintData APIリファレンス	5
4.1 値とインデックス	6
4.2 ページごとに定義を切り替える	7
5. 明細と改ページ	8
6. 画像・バーコード・動的属性	8
7. ブラウザへ渡して表示する	9
8. PDFを作る二つの場所	9
8.1 ブラウザで作る	9
8.2 サーバーで作る	9
9. PREPEJを保存・再利用する	10
10. Web APIとデータベース	10
11. テストとリリース確認	10
12. よくある問題	11
13. 配置・更新・ライセンス	11

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. PHP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ーア、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 **PHP 8.3** です。同梱のDocker構成では必要な言語環境とDBをコンテナ内に用意するため、まずPC側で準備するのはDockerです。Windows／MacではDocker Desktopを起動し、LinuxではDocker EngineとComposeプラグインが使える状態にしてください。

ZIPを全体ごと展開してください。共通RuntimeとDB初期化データも使うため、番号フォルダーだけを取り出さないでください。初回のイメージ取得とビルドにはインターネット接続が必要です。

短いコードから学ぶ場合は `WebAssembly/Samples/php/01`. あっという間に帳票出力/`README.html` から始めます。ここでは全帳票を確認できる02の起動を説明します。

1. VS CodeやCursor、または使い慣れたエディターで `WebAssembly/Samples/php/02`. 全帳票をWebでプレビューを開きます。
2. そのフォルダーでターミナルを開き、次を実行します。開発用のReports Webリポジトリをビルドする必要はありません。

```
docker compose up -d --build --wait
docker compose ps
```

3. 起動が成功してから、ブラウザで `http://127.0.0.1:19201/` を開きます。URLを開くだけではアプリは起動しません。
4. 終了するときは、同じフォルダーで `docker compose down` を実行します。このサンプルのコンテナとネットワークが削除されます。DBデータの初期化やイメージの片付けは、そのフォルダーのREADMEに手順があります。

02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。請求書などを選び、ソースと出力結果を見比べてください。別のアプリから帳票APIを呼ぶ使い方は、続く03のREADMEに説明しています。

3. 最初の一枚を作る

01のサンプルは、印刷データの形を見やすくするためにPHPの配列を直接組み立てています。業務アプリでは、同じ印刷データを `PrintData` クラスで作れます。まず、動いた01の画面を使って、この違いを試みましょう。新しいWebサーバーを作る必要はありません。

1. 01のREADMEに沿ってサンプルを起動します。
2. ZIP内の `WebAssembly/Samples/php/02`. 全帳票をWebでプレビュー/`src/PrintData.php` を、01の `quick-start.php` と同じフォルダーへコピーします。02を起動する必要はありません。
3. 元へ戻せるよう `quick-start.php` のコピーを取り、内容を次のコードに置き換えます。
`report.prepdj`、`router.php`、`index.html` はそのまま使います。

```
01. あっという間に帳票出力/
├ quick-start.php ← 次のコードに変更
├ PrintData.php   ← 02のsrcからコピー
├ report.prepdj   ← 同梱の定義をそのまま使う
├ router.php
└ index.html
```

```

<?php
declare(strict_types=1);
use Pao\Reports\Web\PrintData;

require_once __DIR__ . '/PrintData.php';

function createPrintData(): array
{
    $definition = json_decode(
        file_get_contents(__DIR__ . '/report.prepdj'),
        true, 512, JSON_THROW_ON_ERROR
    );

    return (new PrintData())
        ->setDefinition($definition)
        ->pageStart()
        ->setValue('Message', 'PrintDataで作った最初の帳票')
        ->pageEnd()
        ->toArray();
}

```

保存して `http://127.0.0.1:19200/` を再読み込みします。「PrintDataで作った最初の帳票」と表示されたら成功です。上部の「PDFを保存」でも同じ文字を確認できます。通常起動では、PHPの編集後にDockerを再起動する必要はありません。

`Message`は同梱の`report.prepdj`にある動的テキストの**オブジェクト名**です。見た目の文字列ではなく、この名前で値の入れ先を指定します。ご自分のデザインを使う場合は、定義ファイルとオブジェクト名の両方を合わせてください。

`use`はクラス名を短く書くための宣言で、ファイルを読み込む処理ではありません。この例では`require_once`で実装を読み込みます。`router.php`が`createPrintData()`の戻り値をJSONにしてブラウザへ返すため、ここでは`header()`や`echo`を書きません。自作のAPIでJSONを直接返す場合は、同じクラスの`toJson()`を使用します。

元の01へ戻すときは、退避した`quick-start.php`に戻します。追加した`PrintData.php`は元のコードから参照されません。

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

配布ZIP内の実装ファイルは `WebAssembly/Samples/php/02.全帳票をWebでプレビュー/src/PrintData.php` です。各番号のサンプルにも必要なソースを置いています。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。
drawObject	値を変更せず、指定したオブジェクト／繰返し位置を描画対象にします。
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。
pageEnd	現在のページを確定し、印刷データへ追加します。
toArray / 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

引数と戻り値

`new PrintData()`は引数なしです。以下の`self`は同じPrintDataインスタンスを返す意味で、`->`で呼び出しをつなげられます。`$value`は文字列・整数・小数・真偽値・null、`$values`は属性名をキーにしたPHP配列です。

```

setDefinition(array $definition): self
pageStart(?array $definition = null): self
setValue(string $name, $value, int $index = 0, bool $drawing = true): self
setRepeatedValue(string $name, $value, int $indexX, int $indexY,
    bool $drawing = true): self
setTypedValue(string $name, string $type, $value, int $index = 0,
    bool $drawing = true): self
setImage(string $name, string $source, int $index = 0,
    bool $drawing = true): self
setBarcode(string $name, string $code, int $index = 0,
    bool $drawing = true): self
drawObject(string $name, int $index = 0, string $type = "DynamicText"): self
changeAttributes(string $name, array $values, int $index = 0): self
changeRepeatedAttributes(string $name, array $values, int $index = 0,
    ?int $indexX = null, ?int $indexY = null,
    string $type = "DynamicText"): self
pageEnd(): self
toArray(): array
toJson(bool $pretty = true): string
save(string $path): void

```

これは引数を調べる一覧で、実行用コードではありません。`toArray()`はPHP配列、`toJson()`はUTF-8 JSON文字列です。`save()`はPHPが動いている**サーバー上のファイル**を保存します。利用者のPCへのダウンロードは、HTTP応答をブラウザ側で保存する別の操作です。保存先のフォルダーは先に用意し、PHP実行ユーザーへ書込権限を与えてください。

呼出順の違反は`LogicException`、空の名前や負のインデックス等は`InvalidArgumentException`、保存失敗は`RuntimeException`として通知されます。HTTP APIでは例外をログへ記録し、利用者へは内部のパス等を含めないエラー応答を返します。

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。値設定はページ内のリストへ順に追加されます。同じ名前・同じ位置へ何度も設定する代わりに、アプリ側で値を確定してから一度設定すると分かりやすくなります。`changeRepeatedAttributes()`で二方向の位置を指定するときは、`indexX`と`indexY`の両方を指定します。

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

以下は明細処理の考え方を示す抜粋です。01のMessageだけの定義ではなく、ページ・品番・品名・金額という名前の動的テキストを用意した帳票定義と、code・name・amountの列を持つ\$details配列が必要です。そのまま試せる複数ページの実装は02のsrc/SampleCatalog.phpと帳票定義を併せてご覧ください。

```
$print = (new PrintData())->setDefinition($definition);
foreach (array_chunk($details, 20) as $pageNo => $rows) {
    $print->pageStart()->setValue('ページ', $pageNo + 1);
    foreach ($rows as $rowNo => $row) {
        $print->setValue('品番', $row['code'], $rowNo)
            ->setValue('品名', $row['name'], $rowNo)
            ->setValue('金額', number_format($row['amount']), $rowNo);
    }
    $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

次はpageStart()とpageEnd()の間で呼ぶ抜粋です。定義に角印（動的画像）、商品QR（動的バーコード）、合計金額（動的テキスト）を用意し、\$stampDataUriには画像のdata URIを設定しておきます。01の定義へそのまま追加しても、対応するオブジェクトがなければ表示されません。

```
$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

PHP 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。次は01の配信URLに合わせた表示部分の例です。自作アプリへ組み込む場合は、プレビューアと印刷データのURLを、そのアプリの配置先へ変更してください。

```
<iframe id="reportsViewer" src="/reports.web/preview/?empty=1"></iframe>
<script>
const viewer = document.querySelector('#reportsViewer');
viewer.addEventListener('load', async () => {
  const response = await fetch('/print-data');
  if (!response.ok) throw new Error(`HTTP ${response.status}`);
  await viewer.contentWindow.ReportsWeb.setPrintData(await response.json());
});
</script>
```

親画面とプレビューア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。

プレビューア側では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

PHPから共通Node/WASMサービスへHTTPでPREPEJを送ります。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プレビューアの「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に残します。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

配布サンプルの確認は、各READMEの起動・ソース変更・PDF保存・終了の順で行えます。01では第3章のコードを試し、02では明細と改ページ、03では別アプリとの受け渡しを確認してください。開発用リポジトリだけにあるテストファイルを、配布ZIPに存在する前提では扱いません。

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、プレビューア、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、プレビューアー、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。

現行の製品版への切替は、購入者のメールアドレスを含む電子署名付きの **reports-web.license** を、実際に帳票を出力する側へ配置して行います。購入者ごとにWASMを差し替える手順ではありません。ブラウザとサーバーでは配置先が異なるため、製品版を使い始めるに従ってください。ライセンスファイル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。