Reports Web

いつもの言語で、帳票も印刷も。

利用ガイド／リリース準備版

デザインに業務データを入れて、帳票を作成・表示・印刷する。さらに、帳票をファイルで人に渡し、JSONでシステムにつなぐ。Reports Webは、Webとパソコンの両方で使うための、デザイナー・エンジン・プレビュアーが揃う開発者向け帳票ツールです。

25年にわたりReports.NETで育ててきた帳票づくりを、Webへ。既存の業務を全部作り替えるのではなく、帳票の出口を広げるための道具です。

本書はフルセットの使い方を説明します。Web用の部品と、WPF・Java・Pythonでそれぞれ開発したパソコン用の帳票デザイナー・プレビュアーを組み合わせ、仕事に合った使い方を選びましょう。配布ファイルと動作環境の詳細は、各パッケージの案内をご確認ください。

本書の読み方：最初から全部覚えなくても大丈夫です。まず第1〜3章で「一枚出せた」を体験しましょう。文字を拡大して確かめる、PDFを手元のブラウザで作る。その先でロゴを入れたくなったら画像の章へ、自分のデータを使いたくなったら言語別の章へ。知りたい順番で読み進めてください。HTML版とPDF版は同じ原稿から生成しています。PDFの目次はクリックして移動できます。

目次

1. 帳票の入口は、ひとつでなくていい

この製品を、ひとつにまとめた理由

Webを基本に。つくる場所も、使う言語も、自由に。

Reports.Webの基本は、**いつもの開発言語から、共通のWebAssembly（WASM）帳票エンジンを使うこと**。各言語の小さな橋渡しライブラリ〈ラッパー〉を通して、Webアプリに帳票機能を組み込む構成です。

帳票は、ブラウザ上でデザインできます。さらに、パソコンでじっくり帳票を作りたい方のために、WPF・Java・Pythonでそれぞれ開発した帳票デザイナーも用意しています。

パソコン用のデザイナーを複数用意しているのは、**Windows・Mac・Linuxなど、お使いの環境に合わせて選べるようにするため**です。例えばWindowsではWPF版、MacではPython版やJava版を利用できます。**ブラウザでもパソコンでも、帳票デザインは共通形式なので、相互に引き継いで使えます。**

また、各版には**帳票エンジンとプレビュアー**も用意しています。出来上がった印刷データも共通形式なので、作成する環境と閲覧する環境を分けて使えます。

WPF版を支える.NETの帳票エンジンと、Java・Pythonでそれぞれ実装した帳票エンジンは、パソコン用アプリにも組み込めます。プログラムからエンジンを呼び出し、帳票デザインと業務データを渡すことで、ローカルで帳票を作成・表示・印刷するアプリを開発できます。

さらに、これらの帳票エンジンをサーバー側で使えば、**Webアプリから帳票を生成し、ブラウザに届ける構成**にも広がります。

Reports.Webの基本は、各言語から共通のWASMエンジンを使う方式です。それに加えて、**WASMを使わず、.NET・Pure Java・Pure Pythonのエンジンで帳票を生成する方式も選べます。**

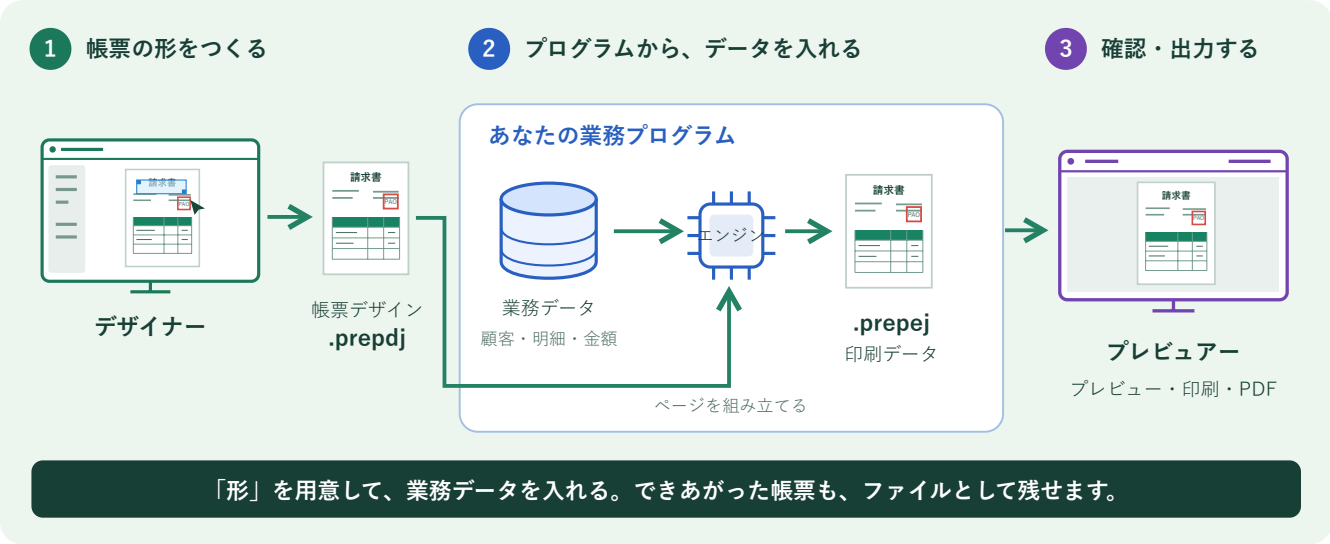
「Javaで構築する業務システムだから、帳票生成も、そのライブラリもJavaだけで揃えたい」。そんな場合にも応えられるよう、パソコン用の道具からサーバー側の帳票生成まで、一つの製品にまとめています。

Webから始まり、パソコンへ。同じ言語で完結する開発へ。
そのつながりを、ここから図でご紹介します。

まずは、帳票ができるまで

デザインに業務データを入れて、一枚の帳票をつくります。

Reports.Webは、**デザイナー・帳票エンジン・プレビュアーが揃う、開発者向け帳票ツール**です。
請求書や見積書などを、自分の業務アプリで作成・表示・印刷するために使います。



図を大きく開く ↗

1. **形を決める**

   デザイナーで文字・罫線・画像を配置し、帳票デザイン `.prepdj` を保存します。

2. **データを入れる**

   あなたのプログラムがエンジンを呼び、デザインに顧客名・明細・金額を当てはめます。DBの読み取りや業務計算はプログラム側が担当します。

3. **帳票として使う**

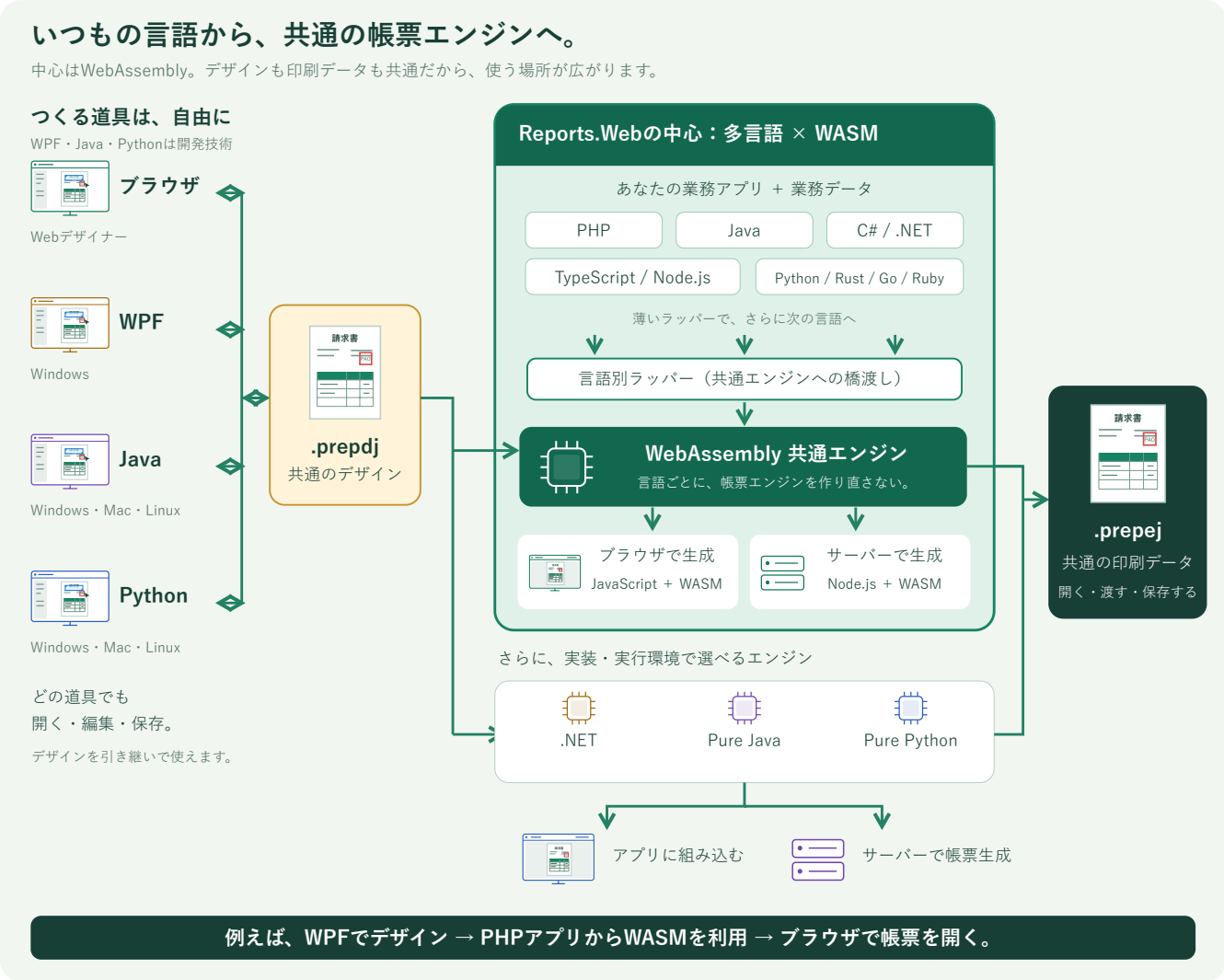
   ページになった印刷データ `.prepej` をプレビュアーで確認・印刷。PDFにして渡すこともできます。

「流し込むデータ」は業務データ。「出来上がった帳票」が印刷データ（プリントドキュメント）です。デザインファイルそのものを業務データで上書きする、という意味ではありません。

ここから、使い方が広がります

デザインも、出来上がった帳票も。
作った環境に、閉じ込めません。

ブラウザで使う道具も、パソコンで使う道具も、別々の島にしない。
共通の帳票デザインと印刷データが、使う道具・言語・場所をつなぎます。



図を大きく開く ↗

中心にあるのは、多言語で使えるWebAssemblyエンジン。

PHP・Java・C#／.NET・TypeScript／Node.jsなど、いつもの業務アプリから**言語別ラッパー（橋渡しのライブラリ）**を通して、共通の帳票エンジンを利用します。帳票生成はブラウザ内でも、Node.jsを使うサーバー側でも。言語ごとに別の帳票エンジンを用意する必要はありません。

Python、Rust、Go、Rubyへも。ラッパーを追加して対応言語を広げる構成です。公開中のWASMサンプルはPHP・Java・C#／.NET・TypeScript／Node.js・Rust・Go・Rubyの7種類です。

さらに、.NET・Pure Java・Pure Pythonで実装した帳票エンジンも選べます。パソコンで使うデザイナーやプレビュアーを支えるエンジンを、アプリにも活用できます。Pure Java／Pure Pythonなら、サーバーの帳票生成までその言語で揃えられます。

デザイン担当者と、開発者の環境を分けられます。

例えば、WPFで開発したWindows用の帳票デザイナーで作った請求書を、Javaのサーバーアプリで使う。ブラウザで作ったデザインを、パソコンのJava／Pythonアプリで使う。デザインを使う場所に合わせて、描き直すための別形式を用意する必要はありません。

作成するアプリと、閲覧する環境も分けられます。

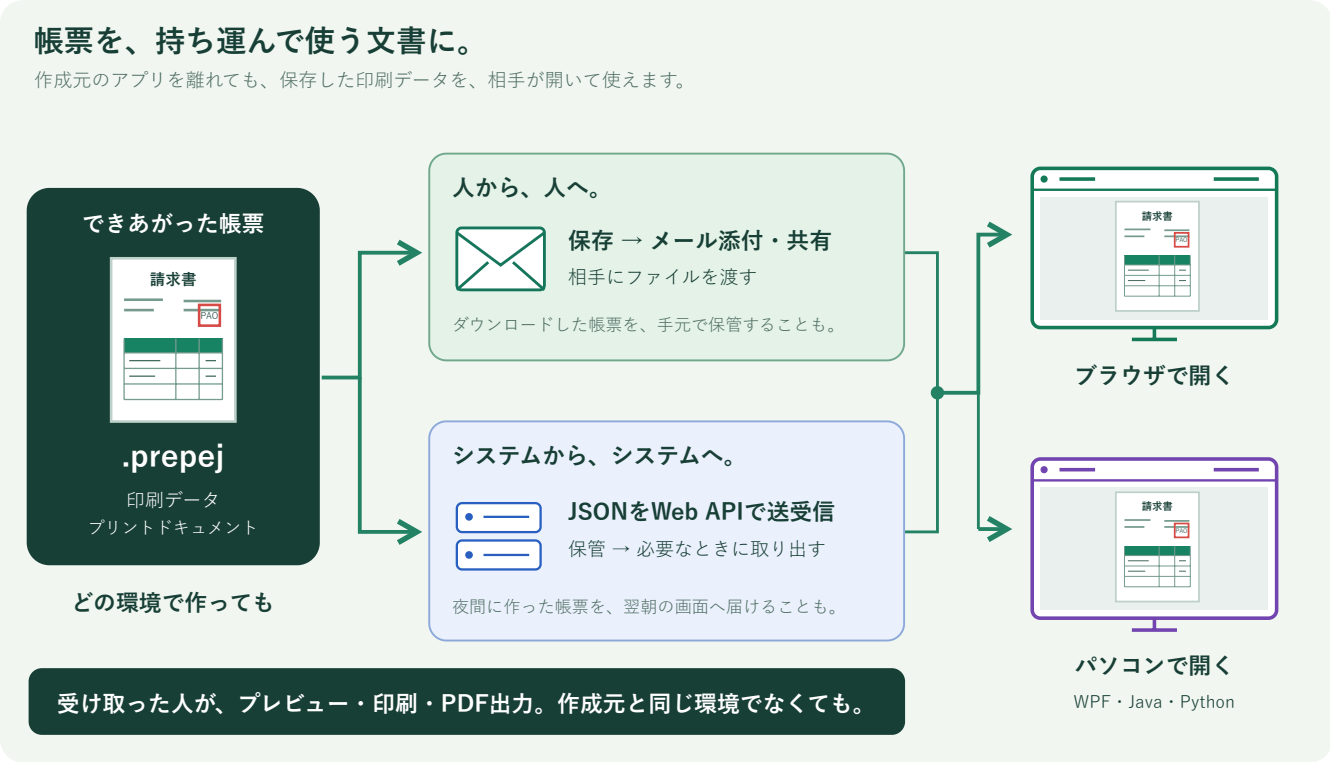
例えば、Javaサーバーが作った印刷データをブラウザで開く。Pythonで生成した印刷データを、パソコンのプレビュアーで確認する。共通の `.prepej` を受け渡すことで、作成元と利用先を組み合わせられます。

共通形式でも、使用する機能の対応範囲・画像などの資源・フォントは受け渡し先で確認します。全環境の表示や改行が無条件に同一になる、という保証ではありません。

印刷データも、持ち運べる文書です

保存して、送る。
受け取った人が、開いて使う。

Excelファイルを保存して人に渡すように、**帳票そのものを一つの文書として扱う。**
その役割を担うのが、印刷データ `.prepej`（プリントドキュメント）です。



図を大きく開く ↗

人に渡す：ファイルとして保存・送付

1. プレビュアーから印刷データを保存。
2. メールに添付する、共有フォルダーに置く。
3. 相手が「開く」やドラッグ＆ドロップで読み込む。
4. 帳票を確認し、印刷・PDF出力する。

保存した帳票は、あとから自分で開き直すこともできます。

システムにつなぐ：JSONをAPIで送受信

`.prepej` の中身はJSONです。業務アプリのWeb APIで送受信・保管し、必要なときに取り出してプレビュアーへ渡せます。

例えば、夜間のバッチで帳票をまとめて生成し、翌朝ログインした人の画面へ届ける。別のサーバーに保管して、後日取り出す。通信・保管・認証は組み込む業務システムが担当します。

Excelと同じなのは「ファイルを保存して受け渡す」という扱い方です。Excel形式ではなく、表計算や任意のセル編集を行うものでもありません。外部参照の画像やフォントなど、表示に必要な資源の準備も必要です。

まずは請求書で、保存して開き直してみる ↗

帳票づくりに必要なものを、一式に

帳票の機能

載せたいものがある。作りたい書式がある。
文字や罫線から画像・バーコードまで、業務の一枚を組み立てられます。

文字・罫線・図形

文字、直線・円弧、四角形・角丸四角形、円・楕円。見出しや囲み、明細欄など、帳票の基本となる部品を組み合わせてレイアウトできます。

画像・ロゴ・印影

会社ロゴ、商品写真、角印などを帳票に配置。固定の画像だけでなく、業務データに合わせてプログラムから画像を差し替えられます。

バーコード・二次元コード

商品や伝票の番号を、読み取り用のコードとして帳票へ。JAN、CODE128、CODE39、NW-7、ITF、QRコード、DataMatrix、PDF417、郵便カスタマバーコードを扱えます。

定型帳票・明細の繰り返し

請求書・見積書・納品書などの決まった書式から、件数に応じた明細の出力まで。文字と罫線を配置し、業務プログラムでデータを流し込みます。

複数ページ・帳票の組み合わせ

表紙と明細、ページをまたぐ一覧、異なる帳票をまとめた出力。業務プログラムでページの区切りや出力順を制御できます。

プレビュー・印刷・PDF・保存

SVGのプレビューで確認して印刷、PDFにして配布。印刷データ自体も保存し、あとから開き直して確認・再出力できます。

細部まで作り込めるデザイナー。位置・サイズ・回転などの細かな調整から、元に戻す・やり直しまで、帳票を作り込むための編集機能を備えています。

各エンジン・出力形式の対応範囲、フォント・画像などの条件はマニュアルでご確認ください。装飾文字オブジェクトは対象外です。

道具が多いのは、選べるようにするため

使うパソコンも、開発する言語も。
仕事に合わせて選べます。

クロスプラットフォームと、同じ言語での開発。二つの自由があります。

デザイン・確認するパソコンを選ぶ

WindowsではWPFで開発した帳票デザイナーとプレビュアーを利用できます。Mac・Linuxを含む環境では、Java・Pythonでそれぞれ開発した帳票デザイナーとプレビュアーを選べます。ブラウザ版のデザイナーも利用できます。

パソコンでじっくり作る担当者と、ブラウザで確認する利用者がいても、共通ファイルでつながります。各対応OSで必要なランタイムを用意して使います。

業務処理と帳票生成を同じ言語に揃える

Pure Java／Pure Pythonは、それぞれJava／Pythonで開発した帳票デザイナー・帳票エンジン・プレビュアーが揃う構成です。パソコンの帳票アプリだけでなく、サーバー側のWebアプリにも同じ言語のエンジンを組み込めます。

「Javaで作る業務システムだから帳票生成までJavaで揃えたい」。その選択肢も、一つの製品の中に含めています。

Webアプリの帳票生成は、2系統から。

① WASM系統：共通エンジンを使う

PHP・Java・C#／.NET・TypeScript／Node.jsなどから、補助ライブラリを通して共通のWASMエンジンを使います。

- **ブラウザ内で生成：**帳票の描画を利用者のパソコンへ分担。
- **サーバーで生成：**Node.js上でWASMを使い、SVG・PDFを返す。

② 言語別エンジン系統：WASMを使わず生成する

.NET・Java・Pythonの業務アプリから、それぞれの言語で実装した帳票エンジンを直接呼び出します。サーバーでSVG・PDFを生成し、ブラウザへ届けます。

この帳票生成にWASM・Node.jsは不要です。ブラウザの画面操作にはHTML・JavaScriptを使います。「JavaからWASMを使う構成」と「Pure Java」は別の方式です。

違いは、3つの実デモで確かめられます。

同じ帳票を、WASMを使わない3つの構成で動かせます。まずはPure Javaから。お使いの技術に合う入口も、そのまま開けます。

Pure Java	Pure Python	.NET
Javaの帳票クラスを直接使う	Pythonの帳票モジュールを直接使う	.NETの帳票ライブラリを直接使う

一つずつが開発セット。それを、オールインワンに。必要な道具とエンジンを選んで使います。すべてを同時に導入・起動する必要はありません。

ここではフルセットの製品構成と使い方を説明しています。各版・サンプルの配布状況は体験版ダウンロードとマニュアルでご確認ください。OSにJava・Python・.NETが必ず標準搭載されている、という意味ではありません。

2. まず、請求書を開いてみよう

いつもの操作で、言語の違いを試す

製品サイトの「デモ」からサンプルを開きます。上段の7言語は共通WASMエンジンを使うデモ、下段のPure Java・Pure Python・.NETは各言語の帳票エンジンを直接使うデモです。公開状況と入口はデモ一覧で確認してください。

1. 初期表示の**請求書**で、罫線・金額・印影を確認します。
2. 帳票の選択欄で、見積書や郵便番号一覧に切り替えます。選ぶと自動でプレビューされます。
3. 下段のプレビュアーでページを送り、倍率を変えます。
4. 上段のPDF生成場所を「ブラウザ（フロントエンド）」にして「PDFを生成」を押します。
5. 同じ帳票で「サーバー（バックエンド）」を選び、出力する場所だけを変えてみます。

サンプル	試してほしいところ
簡単なサンプル	文字・画像を載せる最初の1枚
10のサンプル	繰り返しの中の特定位置だけを変更する処理
郵便番号一覧	多数ページの移動と繰り返し明細
見積書	表紙と明細、ロゴ・印影
請求書	数値の右寄せ、罫線、明細とページ制御
商品大小分類	分類ごとのデータと表の組み立て
名刺	小さな用紙上のレイアウト
デザイン機能見本	基本オブジェクトの表現

上段と下段は、役割が違います

上段は**サンプルアプリ**です。帳票選択、印刷データの保存・読込、PDF生成場所の選択を担当します。下段は**共通プレビュアー**です。表示・検索・印刷を担当します。PDFの入口が二つあるのは、生成場所を比較するためです。

3. 表示する、探す、印刷する

必要なページへ、必要な文字へ

ページ番号の入力と前後移動を使えば、長い帳票でも先頭から順にスクロールする必要はありません。倍率を変えるときは「ページ全体」と数値指定を使い分けてください。

検索欄に文字を入れて検索すると、該当ページへ移動し、該当文字を強調表示します。次の候補は検索操作を繰り返し、前の候補へ戻る操作と解除も利用できます。解除は印刷データの文字を消す操作ではありません。

印刷とPDFの違い

「開く」の隣の「印刷データを保存」で、表示中の帳票の全ページを `.prepej` に保存できます。PDF表示中でも保存対象はPDFではなく共通印刷データです。検索の色やズーム倍率は保存されません。後から「開く」で選ぶか、ファイルをプレビュアーへドロップして再び確認しましょう。

歯車の「表示設定」では、ツールバーとステータスバー、ページ移動、倍率、単一・連続・見開き、検索、言語切替などを、まとめて、または一つずつ表示・非表示にできます。「すべて非表示」にすると帳票の表示面だけになり、業務アプリ側の画面を広く使えます。`Ctrl + Shift + T` でツールバーを戻せます。情報ボタンから開くオンラインヘルプでは、プレビュアーの基本操作を確認できます。

組み込むアプリからも、ページ移動、倍率、表示形式、検索、SVG／PDFの切り替え、印刷、保存、表示項目の設定を公開インターフェース経由で操作できます。ツールバーを表示しない構成でも、利用者向けアプリに用意したボタンから同じ操作を呼び出せます。

操作	動作
プレビュアーの印刷	全ページのSVGをブラウザの印刷機能へ渡す
プレビュアーのPDF表示	ブラウザでPDFを作り、同じ表示領域を切り替える
上段のPDF生成	ブラウザかサーバーを選び、通常は別タブに表示する
上段のダウンロード指定	生成したPDFをファイルとして保存する

印刷はPDFをいったん作って印刷する経路ではありません。印刷プレビューに使うSVGから印刷します。実際の印刷画面はブラウザとOSが表示するため、用紙・余白・倍率・プリンターの設定も確認してください。プリンターを無断で動かすサイレント印刷ではありません。

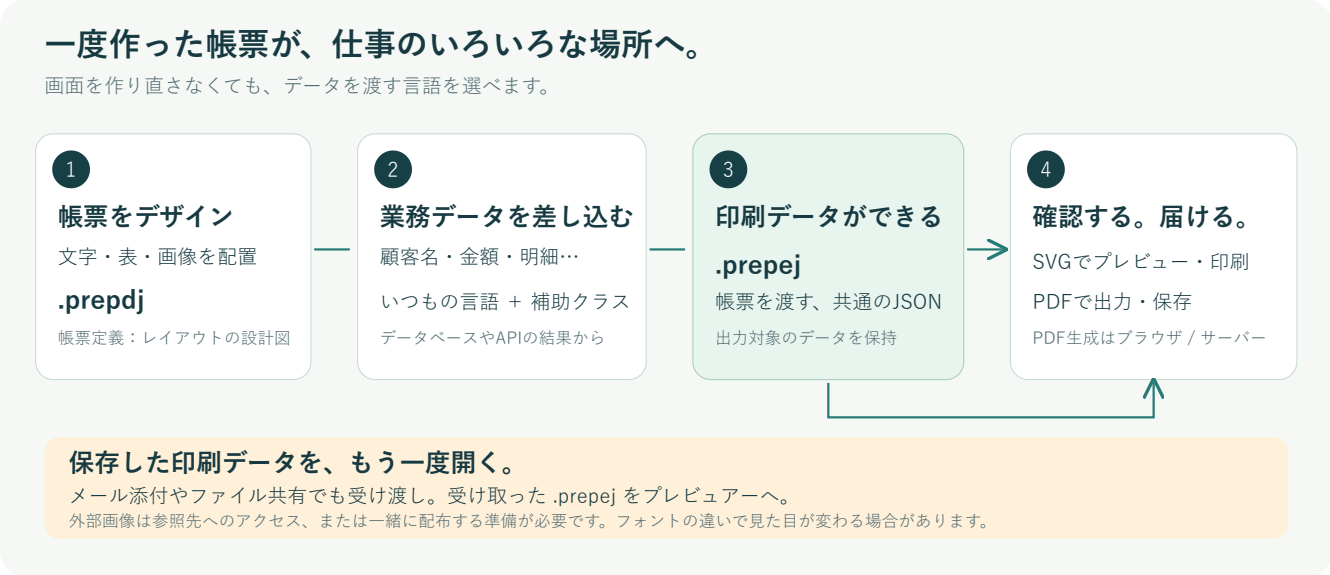
SVGとPDFは、文字や線をベクトルで扱えます。PDFの座標に使うポイントは「解像度が72dpiに固定」という意味ではありません。画像の鮮明さは元画像の画素数、文字の見た目はフォントにも左右されます。

困ったとき：PDFの別タブが開かない場合はポップアップ制限を確認します。印刷時に意図しない縮小やヘッダーが付く場合は、ブラウザの印刷設定を確認してください。

4. 帳票をファイルとして渡してみよう

設計図と、印刷データ

ファイル	内容
`.prepdj`	帳票定義。文字・画像・罫線などの配置を決める設計図
`.prepej`	印刷データ（Print Document）。定義・値・ページ別変更をまとめたデータ



「印刷データを保存」で保存した`.prepej`を、「印刷データを開く」またはドラッグ＆ドロップで読み込みます。サンプルではファイル名を選択欄に追加し、同じ名前を再び開いた場合は、その内容を更新します。ディスク上の元ファイルを勝手に上書きする操作ではありません。

別言語のサンプルへ渡す、業務APIで送る、保管した帳票を後から読む。PDFだけでなく、再表示できる印刷データを持てるのが、この仕組みの面白さです。

1ファイルで持ち運べる条件

外部画像URLを使う帳票は、その画像へアクセスできる必要があります。画像も含めて渡したい場合は、許可した画像を埋め込みます。URL付きのファイルを保存しただけで、外部画像が必ず同梱されるわけではありません。

印刷データには顧客名や金額が含まれます。メール添付や共有場所へ保存するときは、通常の業務文書と同じように取扱範囲を決めてください。JSON形式であること自体は暗号化を意味しません。

5. デザインを変えて、自分の帳票へ

名前を付けると、プログラムとつながる

Webデザイナーで帳票定義を開き、文字や画像を配置します。業務プログラムから変更するオブジェクトには、たとえば`txtTotal`のような名前を付けます。業務側はこの名前を指定して値を入れます。

1. 定義を開くか、新しい帳票を用意します。
2. ツールボックスからオブジェクトを選び、キャンバス上へ配置します。
3. 位置・大きさ・文字・フォント・配置をプロパティで整えます。
4. 帳票定義を`.prepdj`として保存します。
5. サンプルの業務処理から、その定義と名前を使って値を設定します。

サンプルの「デザイナーで開く」は、帳票の設計を見るための補助入口です。保存した定義が自動で業務サーバー上の原本へ反映されるわけではありません。自分のアプリで採用する定義は、管理された配置手順で更新します。

単位を取り違えない

座標は定義の`CoordinateUnit`に従います。.NET由来のmm定義はmmのまま扱い、Webで作るpx定義は96px＝25.4mmとして扱います。フォントサイズのptとは別です。定義の数値だけをコピーして別単位の位置へ代入しないでください。

既存XML定義を使う場合は、対応する.NETデザイナーで読み、JSONとして保存する経路を基本とします。旧XMLの印刷データを、Webがすべて直接読めるという意味ではありません。

デスクトップでも作りたい方へ

WPF・Java・Pythonでそれぞれ開発したパソコン用の帳票デザイナーとプレビュアーを使い、パソコンで設計・確認することもできます。WPFはWindows向け、Java・Pythonは対応する実行環境を用意して利用します。デザインを担当する人はパソコンでレイアウトを作り、開発担当者はその定義をWebアプリへ組み込む、といった分担にも向いています。

6. ロゴと印影が入ると、ぐっと自分の帳票になる

固定画像と差し込み画像

会社ロゴのようにデザイン時に決まるものは**固定画像**。商品写真や社員写真のように出力ごとに変わるものは**差し込み画像**です。どちらも画像を扱いますが、いつ値を決めるかが違います。

使い方	選ぶ方法
会社ロゴを繰り返し使う	許可済み画像URLを参照してキャッシュを活用
商品ごとに写真を変える	業務処理から画像値を指定
ファイルだけで持ち運ぶ	許可した画像をBase64等で埋め込む
閉じた環境で使う	同じ環境の画像配信先、または埋込画像を使う

PHP・Javaなどのサンプルは、登録済みのロゴと印影を安全な画像APIで返します。サーバーでPDFを作る場合は、業務アプリが許可済み画像を読み、画像バイトを印刷データに入れて描画サービスへ渡します。

外部画像を「何でも読む」にしない

ブラウザは画像URLを表示できますが、サーバー描画サービスは利用者が指定した任意URLやサーバー内パスを読みません。URLをそのままサーバーに渡して読み込ませる方式は、内部ネットワークへの不正アクセスにつながるためです。

他言語で保存した印刷データを扱う場合、サンプルの`REPORTS_TRUSTED_ASSET_BASES`は、登録済み画像APIの正確な接頭辞とファイル名を照合します。ドメイン全体を無条件に信頼する設定ではありません。自社画像を追加するときは許可リストと対応ファイルも整備します。

PDFへ入れるGIFは先頭フレームを画像として扱い、アニメーションにはなりません。印影の透明部分は透過に対応した画像を使用してください。

7. フォントは、小さな案内所で整理する

同じ名前が、どのパソコンにもあるとは限らない

Windowsで選んだフォントが、MacやLinux、ブラウザにも必ずあるわけではありません。そこでReports Webは`font-map.json`を使い、帳票のフォント名を配布可能な標準フォントへ対応付けます。

1. 帳票定義と印刷データに登場するフォントを集めます。
2. 対応表で使用するフォントを決めます。
3. 必要なフォント資源を読み込みます。
4. 読み込んだ資源を再利用し、描画に使います。

フォント本体はWASMへすべて埋め込むのではなく、製品の静的資源として一緒に配置します。お客様が外部のフォント登録サービスへ登録する必要はありません。必要なファイルを自社の配信先へ置く方式です。

対応表がなくても、既定の選択へ

`font-map.json`が見つからない場合は、内蔵の最小対応表へフォールバックします。未知のフォント名も既定のNoto Sans CJK JPへ解決します。ただし、フォント本体まで無くてよいという意味ではありません。配布資源を欠かさず配置してください。

製品標準の対応付けはSVGとPDFの整合を重視しています。それでも元のWindowsフォントと代替フォントでは、字形や文字幅に差が残ることがあります。元フォントの完全再現や、OSをまたぐ全フォントの同一表示を保証するものではありません。

独自フォントを使う場合は、配信・PDFへの埋込を許可するライセンスを確認します。ファイルを持っていることと、利用者へ配れることは同じではありません。

8. 好きな言語で、印刷データを作る

補助クラスは、共通の荷物を作る道具

PHPやJavaの補助クラスがブラウザのJavaScriptを直接呼ぶわけではありません。サーバー側でJSONを作り、HTTPでブラウザへ返すか、内部描画サービスへ送ります。

共通の作業は「定義を設定 → ページを開始 → 名前で値を入れる → ページを確定 → JSON化」です。以下の例はいずれも`txtTotal`というオブジェクトを持つ定義が必要です。サンプルソースを参照する例であり、公開パッケージをインストール済みと仮定したものではありません。

PHP

```
require_once __DIR__ . '/src/PrintData.php';
use Pao\Reports\Web\PrintData;

$definition = json_decode(
    file_get_contents('invoice.prepdj'),
    true, 512, JSON_THROW_ON_ERROR
);
$document = (new PrintData())->setDefinition($definition);
$document->pageStart();
$document->setValue('txtTotal', '12,345');
$document->pageEnd();
$json = $document->toJson();
```

`samples/php/src/SampleCatalog.php`が帳票固有の処理、`PrintData.php`が補助クラスです。金額計算や顧客検索を自社の処理へ置き換えるところから始められます。

9. Java・TypeScript・C#・Rust・Go・Rubyでも、同じ流れ

Java／Spring Boot

```
var mapper = new ObjectMapper();
var definition = mapper.readTree(
    Path.of("invoice.prepdj").toFile());
var document = new PrintData().setDefinition(definition);
document.pageStart();
document.setValue("txtTotal", "12,345");
document.pageEnd();
String json = document.toJson();
```

`jp.pao.reports.web.PrintData`とJackson、`java.nio.file.Path`を使用します。`samples/java/report-data`はSpring Bootから独立した補助モジュールです。`sample-app`がDB・HTTPを担当します。これは独立製品Reports.Javaのエンジンを呼ぶ方式ではありません。

TypeScript／Node.js

```
import { readFile } from 'node:fs/promises';
import { PrintData } from './PrintData.js';
const definition = JSON.parse(
  await readFile('invoice.prepdj', 'utf8'));
const document = new PrintData().setDefinition(definition);
document.pageStart();
document.setValue('txtTotal', '12,345');
document.pageEnd();
const json = document.toJson();
```

サンプルの`src`内で使う例です。TypeScriptはJavaScriptへビルドします。Javaへ変換するわけではありません。業務アプリのNode.jsと、共通描画サービスのNode.jsは役割の異なるプロセスです。

Rust／Axum

```
use pao_reports_web::PrintData;
use serde_json::Value;

let definition: Value = serde_json::from_slice(
    &std::fs::read("invoice.prepdj")?)?;
let mut document = PrintData::new();
document.set_definition(definition)?;
document.page_start()?;
document.set_value("txtTotal", "12,345")?;
document.page_end()?;
let json = document.to_json(true)?;
```

`PrintData`はWebフレームワークに依存せず、帳票項目へ値を渡して共通の印刷データを作る薄いライブラリです。`samples/rust`のAxumアプリがHTTP処理とPostgreSQLを担当し、描画は他言語版と同じWASMエンジンへ渡します。これはPure Java／Pure Pythonとは異なる、共通WASMを使う方式です。

Go

```
definition, _ := os.ReadFile("invoice.prepdj")
document, _ := printdata.New(definition)
document.PageStart()
document.SetValue("txtTotal", "12,345")
document.PageEnd()
jsonBytes, _ := document.JSON(true)
```

`samples/go/printdata`はHTTPフレームワークに依存しない補助ライブラリです。Go標準のHTTPサーバーを使うサンプルから、共通WASMエンジンへ印刷データを渡します。

Ruby

```
definition = JSON.parse(File.read('invoice.prepdj', encoding: 'UTF-8'))
document = PaoReportsWeb::PrintData.new(definition)
document.page_start
document.set_value('txtTotal', '12,345')
document.page_end
json = document.to_json(pretty: true)
```

`samples/ruby/lib/pao_reports_web`はSinatraから独立した補助ライブラリです。Rubyで組み立てた印刷データを、他言語版と同じWASMエンジンへ渡します。

10. C#と、明細の細かな変更

C#／ASP.NET Core

```
using System.Text.Json.Nodes;
using Pao.Reports.Web;
var definition = JsonNode.Parse(
    File.ReadAllText("invoice.prepdj"))!.AsObject();
var document = new PrintData().SetDefinition(definition);
document.PageStart();
document.SetValue("txtTotal", "12,345");
document.PageEnd();
string json = document.ToJson();
```

`samples/dotnet/ReportData`を参照します。従来の`Pao.Reports.dll`を呼ぶ方式とは別で、ASP.NET Coreから共通Webエンジンへつなぐ入口です。

特定の行だけ変える

繰り返しの途中で一つだけ太字にしたい。合計だけ大きくしたい。単なる項目名と値の一覧だけでは、このような帳票を表せません。補助クラスでは繰り返し位置やページ別属性を指定します。

```
$document->pageStart();
$document->setValue('field5', '12,345', 1);
$document->changeAttributes(
    'field5', ['fontSize' => 16, 'bold' => true], 1
);
$document->pageEnd();
```

制御	PHP・Java・TypeScript・Rust・Go・Ruby	C#
繰り返し位置	setRepeatedValue	SetRepeatedValue
属性変更	changeAttributes	ChangeAttributes
画像指定	setImage	SetImage
バーコード値	setBarcode	SetBarcode

定義に存在する名前を指定し、繰り返し位置と単位を確認してください。作成途中の`PrintData`は要求ごとに作り、複数利用者で共有しません。金額の丸め方は業務側で決めます。

行の背景色を変えてみよう

請求書の見出しをオレンジに、交互の明細を青に、合計欄を黄色に。背景の四角形も、文字と同じように名前と繰り返し位置で指定できます。

```
$document->changeAttributes('LineRect', [
    'fillEnabled' => true,
    'fillStyle' => 'Solid',
    'background' => '#FFDAB9'
], 0);
$document->setValue('LineRect', '', 0);
```

これはページの組み立て中に行う例です。`LineRect`は定義に置いた背景用四角形の名前、`0`は最初の繰り返し位置です。値が空でも、図形をその位置に出力する要求になります。必要な行へ同じように指定してください。

ポイントは、色だけでなく **塗りつぶしを有効にし、種類をSolidにする** こと。定義が「塗りなし（None）」なら、色を指定しただけでは塗りません。Noneを勝手に消さないので、色を持ったまま背景をオフにする使い方もできます。Java・TypeScriptも同じ属性名、C#では`ChangeAttributes`を使います。ブラウザ生成でもサーバー生成でも、この指定は共通です。

11. 自分のアプリに組み込む

サンプルの見た目を真似する必要はありません

顧客検索画面に「帳票」を追加する。注文一覧で選んだ伝票だけ出す。必要なのは、業務側で対象の印刷データを作り、共通プレビュアーへ渡す入口です。

ブラウザ側の連携コードは各サンプルの画面ソースを出発点にしてください。帳票データを渡すタイミング、読み込み完了、PDF完了、エラーを画面へ通知する処理も含めて利用します。未確定のAPI名を推測して呼ぶより、同梱版のコードと一致させることが確実です。

サーバーPDFは内部サービスへ

業務サーバーが`POST /render/pdf`へ印刷データを送り、Node.js上のJavaScriptとWASMがPDFを作ります。Java・PHP・C#がWASMを直接実行する方式ではありません。

この入口を使えば、夜間に請求書を作る、別システムから受け取った印刷データをPDF化する、保管用のPDFを返すAPIを作る、といった応用ができます。メール送信・ジョブ管理・保存先の管理は業務アプリ側で実装します。製品にそれらの業務機能がすべて付属するという意味ではありません。

配置するもの

配置先	必要なもの
Web配信先	HTML・JS・CSS、WASM、画像・フォント・対応表
業務サーバー	選んだ言語のアプリと補助クラス、DB接続設定
内部描画サービス	Node.js、サーバー用JS、WASM、描画資源

ブラウザ生成だけの構成なら、サーバーPDFサービスは必須ではありません。Dockerを使うか通常プロセスにするかは運用方式の選択であり、公開URLにポート番号を出す必要はありません。

12. サンプルのソースを起動する

開発環境の入口

以下のパスは開発用ソース一式のルートからの相対パスです。配布準備版のため、ダウンロード提供の有無と最終構成は製品サイトで確認してください。DB・共通描画サービス・Web資源を先に用意する必要があります。

言語	開く場所	実行・デバッグの入口
PHP	samples/php	PHPのWeb公開先をpublicにし、DB接続を設定
Java	samples/java/pom.xml	Java 21・Maven。IDEでSampleApplicationを起動
Node.js	samples/node	npm ci、npm run build、npm start
C#	samples/dotnet/ReportsWebSamples.sln	SampleAppを起動プロジェクトにしてF5
Rust	samples/rust	Rust 1.94・Cargo。`cargo run --locked`
Go	samples/go	Go 1.25。`go run .`
Ruby	samples/ruby	Ruby 3.3・Bundler。`bundle exec puma config.ru`

Javaは先にMavenの`install`で親POM・補助モジュールをローカルへ登録します。Docker版のビルドは同梱Dockerfileを使用します。Node.jsは22以上を前提とするサンプルで、TypeScript変更後は再ビルドします。C#サンプルの対象は.NET 8で、実行には対応するASP.NET Coreランタイムが必要です。Rust版は`cargo test --locked`、Go版は`go test ./...`、Ruby版は`ruby test/print_data_test.rb`で補助ライブラリを確認します。各版は同梱Dockerfileでも起動できます。

必ず環境に合わせる設定

DB接続、帳票資源、共通Web資源、内部描画サービスのURL、画像APIの公開URLを確認します。コンテナ内のホスト名は、そのまま開発PCで解決するとは限りません。

共通設定には`REPORTS_ENGINE_URL`、`REPORTS_RESOURCE_ROOT`、`REPORTS_WEB_ROOT`があります。DBはJavaのSpring接続設定、Node.jsのPG系環境変数、C#の`REPORTS_CONNECTION_STRING`、Rustの`REPORTS_DB_URL`など、各サンプルに従います。DB認証情報はサーバー側に置き、ブラウザへ配布しないでください。

UIの処理はブラウザ開発者ツール、帳票の値・ページ数は各言語の`SampleCatalog`、サーバーPDFは内部描画サービスのログ、と順番に追うと見通しがよくなります。

13. 安心して使うための確認

試せることと、正式な保証を分ける

本書とサイトはリリース準備版です。装飾文字オブジェクトはReports Webの対象外です。既存Reports.NETのすべての表現、全OSのすべてのフォント、あらゆるプリンターの完全一致を約束するものではありません。

業務で採用する前には、自社の帳票定義・データ・用紙・ブラウザ・フォントを使い、プレビュー、PDF、実印刷を確認します。ファイルが生成できることと、業務上必要な位置・余白で印刷できることは別の確認です。

運用のチェックリスト

- HTTPSと認証を設定し、顧客情報を許可した利用者だけへ返す。
- DBと内部描画サービスを一般公開しない。
- 画像は許可リストを使い、任意URLや任意ファイルを読み込ませない。
- 入力サイズ、タイムアウト、同時生成数に上限を設ける。
- エラー時に接続情報や内部例外をそのまま画面へ出さない。
- 待機時だけでなく大量ページ出力中のメモリも測る。
- 定義と描画資源を版管理し、元へ戻せる状態で更新する。

サンプルの制限を外して無制限に処理を受け付けるのは避けてください。大きな画像や多数ページは、ブラウザでもサーバーでもメモリを使います。Dockerを使わなくても、起動中のアプリはメモリを消費します。

14. よくある疑問

PHPで作った帳票をJavaで開けますか？

共通形式の印刷データを使うことで、そのような利用ができます。ただし、外部画像へのアクセスとフォント資源も必要です。形式・対応機能・資源をセットで確認します。

PDFとSVGのファイル内容は同じですか？

異なる形式です。PDFは文書として保存・配布する用途、SVGはプレビュー・印刷の描画に使います。別経路のPDF同士でも、内部のバイト一致ではなく必要な表示結果を確認します。

Webなのに、なぜサーバーでJavaScriptを使うのですか？

Node.jsがブラウザの外でJavaScriptを動かします。共通WASMをサーバーで利用する窓口として使うことで、各言語から同じ描画処理へ依頼できます。

ロゴが消えたときは？

画像URLが利用可能か、認証やCORSが妨げていないか、サーバーPDFの場合は許可済み画像として解決されているかを確認します。保存したJSONの画像URLが古いサンプル先を指している場合もあります。

数値や罫線の位置が違うときは？

まず同じ定義・同じ印刷データかを確認し、mm／px、フォント、内側余白、右寄せ、繰り返し位置を順に調べます。帳票全体を拡大縮小して違いを隠す方法は避けてください。

次に何を試せばよいですか？

自社の帳票から一つ選び、デザインのオブジェクト名と業務データを結び付けてください。次に、その印刷データを保存してもう一度開いてみましょう。作る場所と読む場所が離れても、同じ帳票が届く。その小さな成功が、次の業務へ広げる入口になります。

Reports Web — 帳票づくりの経験を、次の場所へ。

15. 使用許諾

Reports Web（エンジン及びデザイナー）の使用について、Reports Webの使用者（以下「利用者様」と称します）と有限会社パオ・アット・オフィス（以下「弊社」と称します）は、以下の各項目についての内容に同意するものとします。

1. Reports Webの使用に関する使用許諾書

この使用許諾書は、利用者様がお使いのパソコンにおいて、Reports Webを使用する場合に同意しなければならない契約書です。

2. 使用許諾書の同意

利用者様がReports Webを使用する時点で、本使用許諾書に同意されたものとします。同意されない場合は、Reports Webを使用する事はできません。

3. 試用制限

Reports Webは、いつでもどこでも誰でもその機能を試していただくことが可能です。ただし、デザイナーでは一定回数以上の保存ができない、エンジンでは帳票に「SAMPLE」という文字が挿入されるという制限があります。

4. ライセンス（使用権）の購入

製品版を使用して開発を行う場合、1台の開発用コンピュータにつき1ライセンスの購入が必要です。お客様環境等、開発コンピュータでないマシンでの使用にはライセンスは不要です（ランタイムライセンスフリー）。

5. 著作権

Reports Webの著作権は、いかなる場合においても弊社に帰属いたします。

6. 免責

Reports Webの使用によって、直接的、あるいは、間接的に生じた、いかなる損害に対しても、弊社は補償賠償の責任を負わないものとします。

7. 禁止事項

Reports Web及びその複製物を第三者に譲渡・貸与する事は出来ません。Reports Webを開発ツールとして再販/再配布することを禁止します。なお、モジュールとしてアプリケーションソフトに組み込みを行い再販/再配布する場合は、開発ツールとしての再販/再配布には含まれません。

8. 保証の範囲

弊社はReports Webの仕様を予告無しに変更することがあります。その場合の利用者様に対する情報提供は、弊社WebSiteにて行う事とします。

9. 適用期間

本使用許諾条件は利用者様がReports Webを使用した日より有効です。利用者様が本使用許諾条件のいずれかの条項に違反した場合、又は、本許諾条件に同意出来ない場合は、利用者様はReports Webを一切使用出来ないものとします。

16. 価格・お支払い・製品のお届け

Reports Webを正式にご利用頂く場合は、ライセンスを購入して頂く必要があります。

必要なライセンス数と価格

Reports Webを使用して開発を行うパソコンの台数分のライセンスが必要です。

内容	税抜価格	税込価格
Reports Web フルセット・開発用PC1台分	100,000円	110,000円

- バグフィックス等のバージョンアップは原則として無償とさせて頂きます。
- 大幅な機能追加等によるバージョンアップの場合には別ライセンスとさせていただく場合がございます。
- 本価格はReports Webの使用権に対するものです。カスタマイズや保守等の費用は一切含まれておりません。
- 運用先のサーバーやエンドユーザーのPCでの使用は、開発用途でない限りランタイムライセンスフリーです。

お支払い方法

ご購入・入金連絡の窓口：https://www.pao.ac/reports.web/buy.html

銀行振込・郵便振替の場合は、110,000円×ライセンス数を下記口座へお振り込みください。振込手数料は利用者様負担でお願い致します。

銀行名	支店名（コード）	口座番号	名義
三菱UFJ銀行	新宿支店（341）	普通 3831891	ユ）パオアツトオフイス
PayPay銀行	すずめ支店（002）	普通 6461359	ユ）パオアツトオフイス

郵便口座番号	名義
00150-0-576845	有限会社 パオ・アット・オフィス

お支払いの通知と製品の送付

振り込みが完了した時点で、必ず弊社へ入金のご連絡をお願いいたします。購入ページの受付窓口、または `info@pao.ac` へ、製品名「Reports Web」、振込人名、振込日、金額、ご購入ライセンス数をお知らせください。

弊社では上記連絡を受けて入金確認を行い、購入者様のメールアドレスを組み込んだ製品版エンジンを含む提供物の入手方法を、電子メールでご案内します。Reports.NETのライセンスキーをデザイナーに入力する方式とは異なります。受信した案内と製品版ファイルは、必ずバックアップを取る等して大切に保管してください。

17. 書類発行・お問い合わせ

見積書・納品書・請求書・領収証

見積書/納品書/請求書/領収証の発行は可能でございます。本製品納品後のお支払いについても、ご購入窓口へご相談ください。

https://www.pao.ac/reports.web/buy.html

製品に関するお問い合わせ

有限会社 パオ・アット・オフィス

- 製品サイト：https://www.pao.ac/reports.web/
- メール：info@pao.ac

サポート掲示板は製品サイトからご案内します。ご質問にはReports Webのバージョン、ご利用の言語、OS、ブラウザ、操作手順をお書きください。

掲示板は公開されています。ライセンス情報、購入者様のメールアドレス、個人情報や機密情報を含む実データは投稿しないでください。購入・入金に関するご連絡は、公開掲示板ではなく購入窓口またはメールをご利用ください。