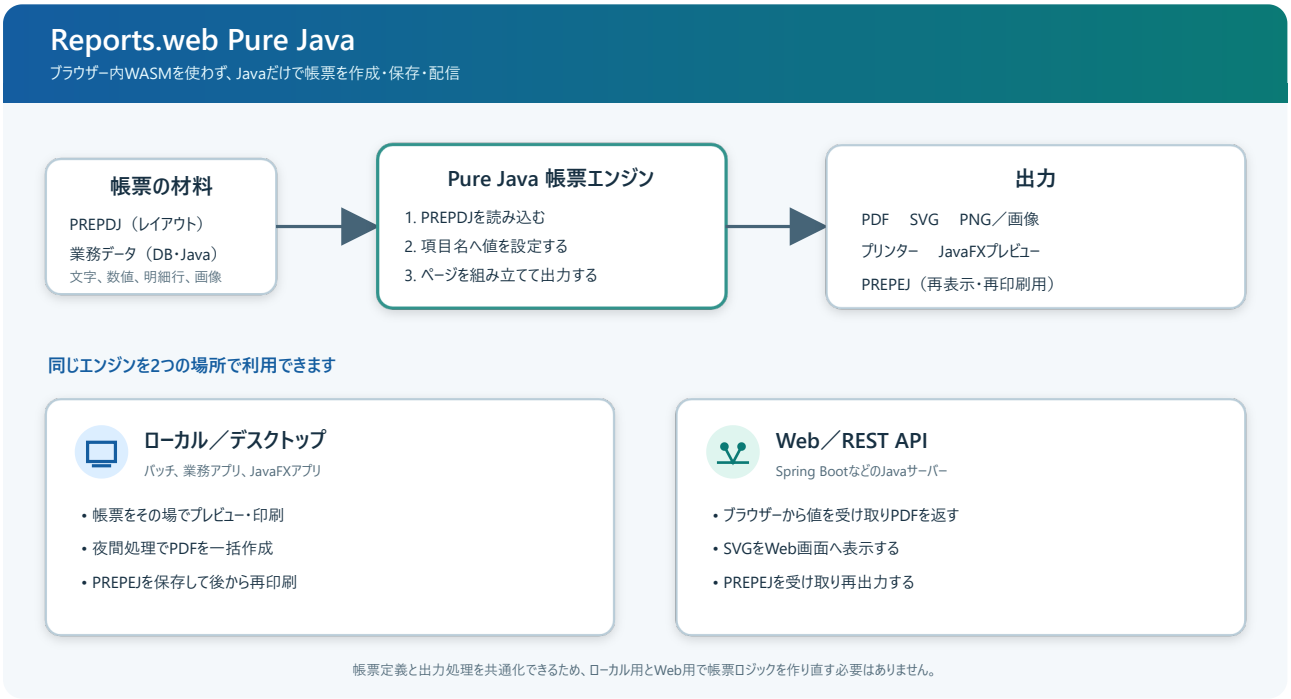
Reports.web Pure Java プログラマーズガイド／APIリファレンス

Reports.web Pure Javaは、ブラウザー内のWASMを使わず、Javaだけで帳票を作成・表示・保存・印刷するための帳票エンジンです。

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。

- JavaからPDF・ページ単位のSVG／PNGを作る
- JavaFXなどのデスクトップアプリケーションでプレビュー・印刷する
- Spring BootなどのWebアプリケーションから帳票を配信する
- 完成した帳票をPREPEJへ保存し、後から再表示・再印刷する
- Pure Javaエンジンの主要なクラス、メソッド、プロパティを調べる



目次

1. 最初に理解する3つのデータ

帳票を作るときは、「レイアウト」と「値」を分けて考えます。

データ	Javaの型	内容
帳票定義	`ReportDocument`	用紙、文字枠、罫線、画像、バーコード、明細の繰返しなどのレイアウト
印刷データ	`ReportData`	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	`PreviewJob`	帳票定義と印刷データをページ単位でまとめた、再表示・再印刷可能なデータ

それぞれを保存するファイルは次のとおりです。

拡張子	内容	主な用途
`.prepdj`	JSON形式の帳票定義	デザイナーで作成・編集し、Javaから読み込む
`.prepd`	従来形式の帳票定義	既存帳票の読み込み、移行
`.prepej`	帳票定義と値を含む完成した印刷データ	保存、転送、プレビュー、再印刷
`.prepe`	従来形式の完成した印刷データ	既存データの読み込み

通常の処理は次の3段階です。

1. `ReportDefinitionIO` でPREPDJを読み込む
2. `ReportData` に項目値と明細行を設定する
3. `PdfExporter`、`SvgExporter`、`ReportRenderer`、`PrinterOutput`のいずれかで出力する

PREPEJは、出力後の帳票を保存・受け渡しするときに使用します。PDFを1枚作るだけなら、最初からPREPEJを作る必要はありません。

2. 開発を始める

2.1 動作環境

- JDK 21
- デスクトップのデザイナー／プレビュアーを使う場合はJavaFX 21以上
- Webサーバーだけで PDFやSVGを作る場合、JavaFXは不要

製品に含まれるエンジンJARと、その依存ライブラリをアプリケーションのクラスパスへ追加してください。製品ソースをGradleのマルチプロジェクトとして利用する場合は、アプリケーション側から `engine` プロジェクトを参照します。

```
dependencies {
    implementation(project(":engine"))
}
```

エンジンはPDF出力とJSON/XML処理のため、PDFBoxおよびJacksonを使用します。配布物をJAR単位で組み込む場合は、製品に同梱された依存JARも一緒に配置してください。

2.2 パッケージ

通常の帳票出力で使用するパッケージは次の3つです。

パッケージ	役割
`com.pao.reports.engine.model`	帳票定義、印刷データ、印刷ジョブのモデル
`com.pao.reports.engine.io`	PREPDJ、PREPEJ、値データの読み書き
`com.pao.reports.engine.render`	PDF、SVG、画像、プリンターへの出力

3. 最初の帳票プログラム

次のプログラムは、`invoice.prepdj` を読み、3つの値を設定して `invoice.pdf` を作ります。

```
import com.pao.reports.engine.io.ReportDefinitionIO;
import com.pao.reports.engine.model.ReportData;
import com.pao.reports.engine.model.ReportDocument;
import com.pao.reports.engine.render.PdfExporter;

import java.nio.file.Files;
import java.nio.file.Path;

public class CreateInvoicePdf {
    public static void main(String[] args) throws Exception {
        Path definitionFile = Path.of("reports", "invoice.prepdj");
        Path outputFile = Path.of("output", "invoice.pdf");
        Files.createDirectories(outputFile.toAbsolutePath().getParent());

        // 帳票のレイアウトを読み込む
        ReportDocument definition =
                new ReportDefinitionIO().read(definitionFile);

        // 帳票へ差し込む値を設定する
        ReportData data = new ReportData();
        data.values.put("顧客名", "株式会社サンプル");
        data.values.put("請求日", "2026年9月14日");
        data.values.put("合計金額", 120000);

        // PDFを作る
        new PdfExporter().export(definition, data, outputFile);
    }
}
```

`data.values` のキーには、デザイナーで文字オブジェクトの「連結する項目」に設定した名前を指定します。名前が一致したオブジェクトへ値が表示されます。

4. 印刷データを設定する

4.1 1帳票に1つの値を設定する

顧客名、伝票番号、発行日、合計金額のような値は `values` に設定します。

```
ReportData data = new ReportData();
data.values.put("伝票番号", "INV-2026-0012");
data.values.put("顧客名", "株式会社サンプル");
data.values.put("発行日", LocalDate.of(2026, 9, 14));
data.values.put("合計金額", 120000);
```

値には文字列だけでなく、数値、真偽値、日付なども渡せます。画面へ表示される形式を固定したい場合は、業務プログラム側で文字列へ整形してから設定してください。

```
NumberFormat money = NumberFormat.getNumberInstance(Locale.JAPAN);
data.values.put("合計金額", money.format(120000) + " 円");
```

4.2 明細行を設定する

商品一覧のように同じレイアウトを繰り返す値は、1行を `Map<String, Object>` として `rows` へ追加します。

```
Map<String, Object> row1 = new LinkedHashMap<>();
row1.put("商品名", "商品A");
row1.put("数量", 2);
row1.put("単価", 50000);
data.rows.add(row1);

Map<String, Object> row2 = new LinkedHashMap<>();
row2.put("商品名", "商品B");
row2.put("数量", 1);
row2.put("単価", 20000);
data.rows.add(row2);
```

帳票定義側で繰返しが設定されたオブジェクトには、`rows` の各行が順番に割り当てられます。1ページに収まらない場合は、エンジンが必要なページ数を計算します。

4.3 DBの検索結果から明細を作る

```
try (PreparedStatement statement = connection.prepareStatement(
        "select product_name, quantity, unit_price from invoice_line where invoice_id = ?")) {
    statement.setLong(1, invoiceId);

    try (ResultSet result = statement.executeQuery()) {
        while (result.next()) {
            Map<String, Object> row = new LinkedHashMap<>();
            row.put("商品名", result.getString("product_name"));
            row.put("数量", result.getInt("quantity"));
            row.put("単価", result.getBigDecimal("unit_price"));
            data.rows.add(row);
        }
    }
}
```

SQL列名を帳票項目名と直接結び付けず、`row.put(...)` の箇所で明示的に対応付けると、DB変更の影響を帳票へ持ち込みにくくなります。

4.4 印刷時だけ位置や色を変更する

帳票定義を変更せず、今回の印刷だけオブジェクトの表示、位置、色などを変えることもできます。キーは `@オブジェクト名.プロパティ名` の形式です。

```
data.values.put("@合計金額.Bold", true);
data.values.put("@社内控え.Visible", false);
data.values.put("@タイトル.FontSize", 18);
```

明細行ごとに指定すれば、その行に対応する繰返しオブジェクトだけを変更できます。

```
Map<String, Object> row = new LinkedHashMap<>();
row.put("商品名", "特別値引き");
row.put("@商品名.Bold", true);
row.put("@商品枠.StrokeColor", "#CC0000");
data.rows.add(row);
```

指定できる主な動的プロパティは次のとおりです。先頭の `@` は省略できますが、通常の項目名と区別しやすいため付けることを推奨します。

分類	プロパティ名
位置・大きさ	`X`、`Y`、`Width`、`Height`、`Angle`／`Rotation`
表示・値	`Visible`、`Text`／`Value`
文字	`FontFamily`／`FontName`、`FontSize`、`Bold`、`HorizontalAlignment`、`VerticalAlignment`
塗り	`FillColor`、`FillEnabled`、`FillStyle`、`HatchDensity`
線・枠	`StrokeColor`、`StrokeWidth`、`StrokeStyle`、`DashPattern`、`LineType`、`TextBorderEnabled`

動的プロパティは一時的な出力変更です。元の `ReportDocument` やPREPDJは変更されません。

5. ローカルアプリケーションで出力する

5.1 PDFを作る

標準のPDF出力は次の1行です。

```
new PdfExporter().export(definition, data, Path.of("output", "invoice.pdf"));
```

この呼び出しは、互換性を重視した100 DPIのイメージPDFを作ります。

文字を検索できるベクターPDFを作る場合は、使用するフォントファイルを明示します。

```
List<Path> fonts = List.of(
        Path.of("fonts", "NotoSansJP-Regular.ttf"),
        Path.of("fonts", "NotoSansJP-Bold.ttf"));

PdfExporter.Result result = new PdfExporter().export(
        definition,
        data,
        Path.of("output", "invoice-vector.pdf"),
        PdfExporter.Options.vector(150, fonts),
        RenderControl.none());

result.diagnostics().forEach(System.out::println);
```

ベクターPDFで利用できない文字は、読めない文字へ置き換えず輪郭として出力され、`diagnostics()` に診断情報が返ります。その文字は表示できますが、PDF内検索の対象にはなりません。

5.2 SVGを作る

1ページだけをSVGファイルへ保存します。

```
new SvgExporter().export(
        definition, data, Path.of("output", "invoice.svg"));
```

複数ページをページ別のSVGへ保存します。

```
List<Path> files = new SvgExporter().exportPages(
        definition, data, Path.of("output", "svg"), "invoice");
```

SVG文字列をWebレスポンスへ直接返す場合は `svg(...)` を使います。

```
String svg = new SvgExporter().svg(definition, data, 0);
```

ページ番号は0から始まります。最初のページは `0` です。

5.3 BufferedImageを作る

```
BufferedImage image = new ReportRenderer().render(
        definition, data, RenderOptions.preview());

ImageIO.write(image, "png", Path.of("output", "invoice.png").toFile());
image.flush();
```

`render(...)` は先頭ページを返します。全ページが必要な場合は `renderPages(...)` を使用します。

```
List<BufferedImage> pages = new ReportRenderer().renderPages(
        definition, data, RenderOptions.preview());
```

大量ページを一度にメモリーへ置きたくない場合は、`pageCount(...)` でページ数を取得し、`renderPage(...)` を1ページずつ呼び出します。

```
ReportRenderer renderer = new ReportRenderer();
int count = renderer.pageCount(definition, data);

for (int page = 0; page < count; page++) {
    BufferedImage image = renderer.renderPage(
            definition, data, RenderOptions.preview(), page, RenderControl.none());
    try {
        ImageIO.write(image, "png",
                Path.of("output", "page-" + (page + 1) + ".png").toFile());
    } finally {
        image.flush();
    }
}
```

5.4 プリンターへ印刷する

印刷ダイアログを表示して印刷します。

```
new PrinterOutput().print(definition, data, true);
```

ダイアログを表示せず、既定のプリンターへ送る場合は第3引数を `false` にします。

既存の印刷処理へ組み込む場合は `Pageable` を取得します。

```
Pageable pages = new PrinterOutput().createPageable(definition, data);
```

5.5 デスクトップでプレビューする

製品付属のJavaFXプレビュアーは、`ReportRenderer` のページ画像を使用します。独自画面へ組み込む場合も、`renderPage(...)` で必要なページだけを描画し、`SwingFXUtils.toFXImage(...)` でJavaFX画像へ変換できます。

```
BufferedImage buffered = new ReportRenderer().renderPage(
        definition, data, RenderOptions.preview(), pageIndex, RenderControl.none());
WritableImage fxImage = SwingFXUtils.toFXImage(buffered, null);
imageView.setImage(fxImage);
buffered.flush();
```

ページ移動時には `pageIndex` を変更して再描画します。最初のページは0、最後のページは `pageCount(...) - 1` です。

5.6 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 `SamplePreviewWindow` のソースを利用する場合は、JavaFXのUIスレッドから次のように操作できます。

```
var preview = new SamplePreviewWindow(owner, "帳票", definition, data);
preview.setPreviewChromeVisible(false); // 帳票だけにする
preview.show();
// アプリ側の再表示ボタンから呼び出します。
preview.setPreviewChromeVisible(true);  // メニュー・全ツールバー・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。これは同梱サンプルWindowの公開メソッドであり、帳票エンジンSDKが独立したUIコントロールを提供する、という意味ではありません。OSのウィンドウ枠は残ります。Web用の `ReportsWebNative` や、そのPDF・検索APIとは別のものです。

6. 完成した帳票をPREPEJへ保存する

PREPEJは、各ページの帳票定義と印刷データをまとめた自己完結ファイルです。元のPREPDJやDBへ接続しなくても、PREPEJだけで同じ帳票を再表示・再印刷できます。

6.1 PREPEJを作る

```
PreviewJob job = new PreviewJob();
job.name = "請求書 INV-2026-0012";

ReportRenderer renderer = new ReportRenderer();
int pageCount = renderer.pageCount(definition, data);

for (int pageIndex = 0; pageIndex < pageCount; pageIndex++) {
    job.pages.add(new PreviewJob.Page(definition, data, pageIndex));
}

new PreviewJobIO().write(
        Path.of("output", "invoice.prepej"), job);
```

`sourcePage` は、元の帳票定義と印刷データの何ページ目を表示するかを示す0始まりの番号です。

6.2 PREPEJを読み、PDFへ再出力する

```
PreviewJobIO.ReadResult loaded =
        new PreviewJobIO().read(Path.of("output", "invoice.prepej"));

if (!loaded.selfContained()) {
    throw new IOException("帳票定義を含むPREPEJではありません");
}

List<PdfExporter.Page> pages = loaded.job().pages.stream()
        .map(page -> new PdfExporter.Page(
                page.definition, page.data, page.sourcePage))
        .toList();

new PdfExporter().exportPages(
        pages,
        Path.of("output", "invoice-reprint.pdf"),
        PdfExporter.Options.image(),
        RenderControl.none());
```

`selfContained()` が `true` なら、そのファイルだけで出力できます。従来の値データだけを読み込んだ場合は `false` になり、`legacyData()` に値が返ります。その場合は別途PREPD／PREPDJが必要です。

6.3 値だけを保存する

`ReportDataIO` は `values` と `rows` だけを保存します。帳票定義を含まないため、このファイルだけではプレビューや印刷はできません。

```
ReportDataIO dataIO = new ReportDataIO();
dataIO.write(Path.of("work", "invoice-data.json"), data);

ReportData restored =
        dataIO.read(Path.of("work", "invoice-data.json"));
```

一時保存、テストデータ、サーバー内部の受け渡しには `ReportDataIO`、利用者へ渡す再表示・再印刷用ファイルには `PreviewJobIO` を使用します。

7. Webアプリケーションで使う

Pure Javaエンジンは、Spring Boot、Jakarta EE、Servletなどのサーバー側Javaから利用できます。ブラウザーへJavaエンジンを配布する方式ではありません。サーバーで帳票を作り、PDF、SVGまたはPREPEJをHTTPレスポンスとして返します。

7.1 業務データを受け取りPDFを返す

次のSpring Boot例では、帳票定義をサーバーに置き、リクエストで受け取った値を設定してPDFを返します。

```
@RestController
@RequestMapping("/api/invoices")
public class InvoiceController {
    private final ReportDocument invoiceDefinition;

    public InvoiceController() throws IOException {
        invoiceDefinition = new ReportDefinitionIO().read(
                Path.of("reports", "invoice.prepdj"));
    }

    @PostMapping(value = "/pdf", produces = MediaType.APPLICATION_PDF_VALUE)
    public ResponseEntity<byte[]> createPdf(
            @RequestBody InvoiceRequest request) throws Exception {

        ReportData data = new ReportData();
        data.values.put("顧客名", request.customerName());
        data.values.put("請求日", request.invoiceDate());
        data.values.put("合計金額", request.total());

        for (InvoiceLine line : request.lines()) {
            Map<String, Object> row = new LinkedHashMap<>();
            row.put("商品名", line.productName());
            row.put("数量", line.quantity());
            row.put("単価", line.unitPrice());
            data.rows.add(row);
        }

        Path temporary = Files.createTempFile("invoice-", ".pdf");
        try {
            new PdfExporter().export(invoiceDefinition, data, temporary);
            byte[] body = Files.readAllBytes(temporary);

            return ResponseEntity.ok()
                    .header(HttpHeaders.CONTENT_DISPOSITION,
                            "inline; filename=invoice.pdf")
                    .contentType(MediaType.APPLICATION_PDF)
                    .body(body);
        } finally {
            Files.deleteIfExists(temporary);
        }
    }
}
```

公開APIでは、入力項目、文字数、明細件数、出力ページ数を検証してください。クライアントから任意のファイルパスやURLを受け取り、画像として読み込ませないでください。

7.2 SVGをWeb画面へ表示する

```
@GetMapping(value = "/{id}/svg", produces = "image/svg+xml")
public ResponseEntity<String> createSvg(
        @PathVariable long id,
        @RequestParam(defaultValue = "0") int page) {

    ReportData data = invoiceService.createReportData(id);
    int pageCount = new ReportRenderer().pageCount(invoiceDefinition, data);
    if (page < 0 || page >= pageCount) {
        return ResponseEntity.badRequest().build();
    }

    String svg = new SvgExporter().svg(invoiceDefinition, data, page);
    return ResponseEntity.ok()
            .contentType(MediaType.parseMediaType("image/svg+xml"))
            .body(svg);
}
```

ブラウザー側では、通常の画像と同様にエンドポイントを表示できます。

```
<img src="/api/invoices/123/svg?page=0" alt="請求書プレビュー">
```

SVGへ含めるデータはHTMLと同様に信頼境界を意識し、認証・認可と入力検証を行ってください。

7.3 PREPEJをREST APIで受け取る

クライアントが完成済みPREPEJを持っている場合は、`PreviewJobIO` で読み込み、各ページをPDFやSVGへ出力できます。

```
Path upload = Files.createTempFile("report-", ".prepej");
try {
    Files.write(upload, requestBody);
    PreviewJobIO.ReadResult loaded = new PreviewJobIO().read(upload);

    if (!loaded.selfContained()) {
        throw new IllegalArgumentException("自己完結PREPEJが必要です");
    }

    PreviewJob job = loaded.job();
    PreviewJob.Page first = job.pages.get(0);
    String svg = new SvgExporter().svg(
            first.definition, first.data, first.sourcePage);
} finally {
    Files.deleteIfExists(upload);
}
```

アップロードサイズ、ページ数、項目数を制限し、一時ファイルは必ず削除してください。PDF生成には同時実行数、処理時間、キャンセルの上限を設けることを推奨します。

7.4 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
`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。

8. 実行の中止と進捗通知

時間のかかるPDFや画像の生成には `RenderControl` を渡せます。

```
AtomicBoolean cancelled = new AtomicBoolean(false);

RenderControl control = new RenderControl(
        cancelled::get,
        progress -> System.out.printf("%.0f%%%n", progress * 100));

new PdfExporter().export(
        definition,
        data,
        Path.of("output", "large-report.pdf"),
        PdfExporter.Options.image(150),
        control);
```

中止条件が `true` になると `CancellationException` が送出されます。Webアプリケーションでは、クライアント切断やタイムアウトと連携できます。

9. 帳票定義を変換・保存する

通常の帳票出力ではPREPDJを読み込むだけです。この章は、Javaで帳票定義自体を編集・変換する場合に使用します。

9.1 帳票定義を保存する

```
ReportDefinitionIO definitionIO = new ReportDefinitionIO();
ReportDocument definition = definitionIO.read(Path.of("invoice.prepdj"));

definition.title = "請求書（社内控え）";
definitionIO.write(Path.of("invoice-copy.prepdj"), definition);
```

`ReportDefinitionIO.write(...)` は帳票定義の保存です。帳票項目へ印刷値を設定するメソッドではありません。値は `ReportData.values` と `ReportData.rows` に設定します。

9.2 読み込み時の診断を確認する

従来形式の帳票定義には、Pure Javaモデルへ完全に対応付けられない設定が含まれる場合があります。移行確認では `readWithDiagnostics(...)` を使います。

```
ReportDefinitionIO.ReadResult result =
        new ReportDefinitionIO().readWithDiagnostics(
                Path.of("legacy-report.prepd"));

ReportDocument definition = result.document();

for (ReportDefinitionIO.ReadDiagnostic diagnostic : result.diagnostics()) {
    System.out.printf("%s %s: %s%n",
            diagnostic.code(), diagnostic.path(), diagnostic.message());
}
```

診断情報は「読み込みに失敗した」という意味ではありません。互換情報として保持した設定や、同じ見た目を保証できない設定を、移行担当者が確認するための情報です。新しく作成したPREPDJを通常利用する場合は `read(...)` で構いません。

9.3 JSON文字列として扱う

```
ReportDefinitionIO definitionIO = new ReportDefinitionIO();

String json = definitionIO.toJson(definition);
ReportDocument restored = definitionIO.fromJson(json);
```

`toJson(...)` と `fromJson(...)` が扱うのは、帳票定義 `ReportDocument` です。印刷データやPREPEJではありません。

10. APIリファレンス

ここからは、プログラム作成中にクラス、メソッド、プロパティを調べるためのリファレンスです。

10.1 `ReportDefinitionIO`

PREPD／PREPDJの読み書きを行います。

メソッド	戻り値	説明
`read(Path path)`	`ReportDocument`	ファイル名の拡張子を判定し、帳票定義を読み込む
`read(InputStream input, String sourceName)`	`ReportDocument`	ストリームから読み込む。`sourceName`には形式判定可能なファイル名を渡す
`readWithDiagnostics(...)`	`ReadResult`	帳票定義に加え、互換性に関する診断情報を返す
`write(Path path, ReportDocument document)`	`void`	拡張子に応じて帳票定義を保存する
`writeCanonical(Path path, ReportDocument document)`	`WriteResult`	標準PREPDJとして保存し、未解決の互換性情報を返す

メソッド	戻り値	説明
`toJson(ReportDocument document)`	`String`	帳票定義をJSON文字列へ変換する
`fromJson(String value)`	`ReportDocument`	JSON文字列から帳票定義を復元する

`ReadResult` のプロパティ：

プロパティ	型	説明
`document()`	`ReportDocument`	読み込んだ帳票定義
`diagnostics()`	`List<ReadDiagnostic>`	互換性に関する診断一覧

`ReadDiagnostic` のプロパティ：

プロパティ	説明
`code()`	診断を識別するコード
`path()`	対象となった帳票定義内の位置
`message()`	診断内容

10.2 `ReportData`

帳票へ差し込む値を保持します。

プロパティ	型	説明
`values`	`Map<String, Object>`	帳票全体で使用する項目名と値
`rows`	`List<Map<String, Object>>`	繰返し明細。リストの1要素が1行

10.3 `ReportDataIO`

帳票定義を含まない、値だけのデータを読み書きします。

メソッド	戻り値	説明
`read(Path path)`	`ReportData`	値データを読み込む
`write(Path path, ReportData data)`	`void`	値データを保存する

10.4 PreviewJobIO

完成した印刷データであるPREPE／PREPEJを読み書きします。

メソッド	戻り値	説明
`read(Path path)`	`ReadResult`	PREPE／PREPEJを読み込む
`write(Path path, PreviewJob job)`	`void`	帳票定義と値を含む印刷ジョブを保存する

`ReadResult` のプロパティ：

プロパティ	説明
`selfContained()`	帳票定義を含み、そのファイルだけで表示・印刷できる場合は `true`
`job()`	自己完結形式を読み込んだ場合の `PreviewJob`
`legacyData()`	従来の値だけの形式を読み込んだ場合の `ReportData`
`diagnostics()`	PREPEJ読み込み時の診断一覧

10.5 PreviewJob

プロパティ	型	説明
`name`	`String`	印刷ジョブの表示名
`pages`	`List<PreviewJob.Page>`	表示・印刷するページの一覧
`canonicalSourceJson`	`String`	読み込んだ標準PREPEJの原文を互換性維持のため保持する領域。通常は変更しない
`diagnostics`	`List<String>`	読み込み・変換時の診断情報

`PreviewJob.Page`：

プロパティ	型	説明
`definition`	`ReportDocument`	このページで使う帳票定義
`data`	`ReportData`	このページで使う印刷データ
`sourcePage`	`int`	元の帳票のページ番号。0始まり

10.6 PdfExporter

メソッド	説明
export(definition, data, target)	全ページを標準設定のイメージPDFへ出力する
export(definition, data, target, options, control)	PDF方式、解像度、フォント、中止・進捗を指定して出力する
exportPages(pages, target, options, control)	異なる帳票定義を含むページ一覧を1つのPDFへ出力する

PdfExporter.Options：

作成方法	説明
Options.image()	100 DPIのイメージPDF
Options.image(dpi)	指定DPIのイメージPDF
Options.vector(dpi, fontFiles)	指定したローカルフォントを使用するベクターPDF

PdfExporter.Result：

プロパティ	説明
mode()	実際に使用した IMAGE または VECTOR
pages()	出力ページ数
renderingDpi()	描画に使用したDPI
diagnostics()	フォントなどの診断情報

10.7 SvgExporter

メソッド	戻り値	説明
export(document, data, target)	void	先頭ページをSVGファイルへ保存する
export(document, data, target, rasterDpi)	void	SVG内で画像化される要素のDPIを指定して保存する
svg(document, data)	String	先頭ページのSVG文字列を返す
svg(document, data, pageIndex)	String	指定ページのSVG文字列を返す
exportPages(document, data, directory, prefix)	List<Path>	全ページを別々のSVGファイルへ保存する

10.8 ReportRenderer

メソッド	戻り値	説明
render(document, data, options)	BufferedImage	先頭ページを画像化する
render(document, data, options, control)	BufferedImage	中止・進捗通知を指定して画像化する
renderPages(document, data, options)	List<BufferedImage>	全ページを画像化する
renderPages(document, data, options, control)	List<BufferedImage>	中止・進捗通知を指定して全ページを画像化する
pageCount(document, data)	int	明細の繰返しを含めたページ数を返す
renderPage(document, data, options, pageIndex, control)	BufferedImage	指定ページだけを画像化する
paintPage(graphics, document, data, dpi, page, control)	void	呼び出し側が用意した Graphics2D へ直接描画する

RenderOptions：

プロパティ	説明
dpi	描画解像度
transparent	背景を透明にするか
pageIndex	対象ページ番号。0始まり
preview()	144 DPI、不透明背景のプレビュー設定
print()	100 DPI、不透明背景の互換印刷設定

10.9 PrinterOutput

メソッド	説明
print(definition, data, showDialog)	印刷ダイアログの有無を指定してプリンターへ出力する
createPageable(definition, data)	Java印刷APIで使用できる Pageable を返す
createPageable(pages)	複数の帳票定義を含むページ一覧から Pageable を作る

10.10 RenderControl

メンバー	説明
new RenderControl(cancelled, progress)	中止判定と進捗通知を指定する
none()	中止・進捗通知を使用しない設定
checkpoint()	中止が要求されていれば CancellationException を送出する
progress(double value)	0〜1の進捗を通知する

11. 帳票モデルのプロパティ

通常はデザイナーでPREPDJを作成するため、Javaからこれらを直接設定する必要はありません。帳票定義を動的に変更する場合や、独自デザイナーを作る場合に参照してください。

11.1 ReportDocument

プロパティ	型	説明
format	String	帳票形式名
formatVersion	int	帳票形式のバージョン
title	String	帳票名
paperSize	String	A4などの用紙名
widthMm, heightMm	double	用紙の幅と高さ。単位はmm
gridMm	double	デザイナーのグリッド間隔。単位はmm
designScalePercent	double	デザイナーの表示倍率
showGrid	boolean	グリッドを表示するか
snapToGrid	boolean	オブジェクトをグリッドへ合わせるか
objectsLocked	boolean	帳票全体のオブジェクトをロックするか
defaultFontFamily	String	新規文字オブジェクトの既定フォント
defaultFontSizePt	double	新規文字オブジェクトの既定サイズ
objects	List<ReportObject>	帳票に配置されたオブジェクト
extensions	Map<String, Object>	互換性維持用の拡張情報。通常は変更しない

プロパティ	型	説明
copy()	ReportDocument	帳票定義の複製を作る

11.2 ReportObject 共通プロパティ

プロパティ	説明
id	オブジェクトを識別するID
name	デザイナー上の項目名
type	TEXT、IMAGE、BARCODE、LINE、RECTANGLE、ELLIPSE
x, y	配置位置。単位はmm
width, height	幅と高さ。単位はmm
rotation	回転角度
visible	出力するか
locked	デザイナーで編集を禁止するか
binding	ReportData のキーと結び付ける項目名
repeat, repeatX	縦方向、横方向の繰返し数
intervalX, intervalY	繰返し間隔。単位はmm
repeatXYBoth	縦横の格子状に繰り返すか
extensions	互換性維持用の拡張情報。通常は変更しない
copy()	オブジェクトの複製を作る

11.3 文字プロパティ

プロパティ	説明
text	値が指定されない場合に表示する固定文字
fontFamily	フォントファミリー名
fontSizePt	文字サイズ。実際の単位は fontUnit に従う
fontUnit	Point などの文字サイズ単位
bold, italic, underline, strikeout	太字、斜体、下線、取消線

プロパティ	説明
horizontalAlignment	LEFT、CENTER、RIGHT
verticalAlignment	TOP、MIDDLE／CENTER、BOTTOM
textVertical	縦書き
textElastic	文字を枠へ合わせる
fixingFontHeight	フォント高さを固定する
textColor	#RRGGBB 形式の文字色
textBorderEnabled	文字枠を表示するか
textOutlineWidth, textOutlineColor	文字の輪郭幅と色

11.4 線・図形・塗りのプロパティ

プロパティ	説明
strokeColor	線色
strokeWidth	線幅
strokeStyle	線種
dashPattern	独自の破線パターン
lineThickness	円弧などの形状を決める線パラメーター
lineType	単線・二重線などの線種情報
fillEnabled	塗りを有効にするか
fillStyle	None、Solid、Hatch
fillColor	塗り色
hatchDensity	ハッチの密度
cornerRadius	矩形の角丸半径
cornerTopLeft ほか	四隅ごとの角形状

11.5 画像プロパティ

プロパティ	説明
imagePath	画像ファイルのパス
imageDataBase64	Base64形式で埋め込んだ画像
imageSizeMode	画像の拡大・縮小方法
imageAlignment	枠内の配置位置
imageReverse	左右・上下の反転
imageBorderEnabled	画像枠を表示するか

Webサーバーでは、外部入力をそのまま `imagePath` として使用しないでください。アプリケーションが管理するファイルまたは検証済みの埋め込み画像に限定します。

11.6 バーコードプロパティ

プロパティ	説明
barcodeKind	CODE128、QR_CODEなどのバーコード種別
barcodeDrawMode	`STRETCH` または `DIRECT`
barcodeTextVisible	バーコード下の文字を表示するか
barcodeJustify	バー幅を枠へ合わせるか
barcodeEvenSpacing	文字間隔を均等にするか
barcodeBlackBarAdjusterByDot	黒バー幅のドット補正
barcodeImageMargin	画像余白
barcodeStartStop	スタート／ストップ文字を表示するか
barcodeCodeSet	CODE128などのコードセット
qrErrorCorrection	QRコードの誤り訂正レベル
qrVersion	QRコードの型番
stringEncoding	文字エンコーディング
pdf417Rows, pdf417Columns	PDF417の行数と列数

12. フォント、性能、安全性

フォント

帳票で指定したフォントが実行環境にない場合は代替フォントが使われ、文字幅や改行位置が変わることがあります。開発PCだけでなく、本番サーバーにも使用フォントを配置して確認してください。

ベクターPDFへフォントを埋め込む場合は、フォントのライセンスと再配布条件も確認してください。エンジンがインターネットからフォントを取得することはありません。

性能

- 帳票定義は毎回読み直さず、変更されない間は安全にキャッシュできます。
- `ReportData` はリクエストや処理ごとに新しく作成してください。
- 大量ページは `renderPages(...)` で一括画像化せず、`renderPage(...)` で順次処理してください。
- Webサーバーではpdf生成の同時実行数、最大ページ数、処理時間を制限してください。
- 不要になった `BufferedImage` は `flush()` してください。

安全性

- PREPDJ、PREPEJ、画像のファイルサイズと内容を入力時に検証してください。
- 外部入力から任意のローカルパスやURLを参照させないでください。
- 出力先はアプリケーション管理下のディレクトリに限定してください。
- 一時ファイルは `finally` またはtry-with-resources相当の管理で削除してください。
- 公開REST APIには認証、認可、レート制限を適用してください。

13. エラー処理の基本

```
try {
    ReportDocument definition =
            new ReportDefinitionIO().read(definitionPath);
    new PdfExporter().export(definition, data, outputPath);
} catch (NoSuchFileException ex) {
    // 帳票定義が見つからない
} catch (CancellationException ex) {
    // 利用者またはタイムアウトにより中止された
} catch (IOException ex) {
    // PREPDJの解析、画像読込、PDF保存などに失敗した
} catch (IllegalArgumentException ex) {
    // ページ番号、DPI、帳票モデルなどの指定が不正
}
```

利用者向けのエラーメッセージと、調査用の例外・ログは分けてください。Web APIでは内部例外をそのままレスポンスへ返さず、要求IDとともにサーバーログへ記録します。

14. 目的別の早見表

やりたいこと	最初に見る箇所	主なAPI
PDFを作る	3、5.1	`ReportDefinitionIO`、`ReportData`、`PdfExporter`
明細帳票を作る	4.2	`ReportData.rows`
画像プレビューを作る	5.3、5.5	`ReportRenderer`
プリンターへ出力する	5.4	`PrinterOutput`
完成帳票を保存・再印刷する	6	`PreviewJob`、`PreviewJobIO`
Web APIからPDFを返す	7.1	`PdfExporter`
Web画面にSVGを表示する	7.2	`SvgExporter`
PREPEJをRESTで受け取る	7.3	`PreviewJobIO`
従来帳票を移行する	9.2	`readWithDiagnostics(...)`
メソッドを調べる	10	APIリファレンス
帳票プロパティを調べる	11	モデルプロパティリファレンス

15. 付属サンプルを実行する

製品ソースに含まれるサンプルでは、ローカルプレビューとWeb／RESTの両方を確認できます。

```
cd C:\Pao\Pao.Reports.Java
$env:JAVA_HOME='C:\Program Files\Android\openjdk\jdk-21.0.8'
.\gradlew.bat :samples:clean :samples:test :samples:installDist --no-daemon
.\samples\build\install\samples\bin\samples.bat
```

Spring BootのWebサンプル：

```
.\gradlew.bat :samples:bootRun --no-daemon
```

起動後、ブラウザーで `http://localhost:8080/` を開きます。

主な参照用API：

API	内容
`GET /api/samples`	サンプル帳票一覧
`GET /api/reports/{id}/pdf`	サンプルPDF
`POST /api/reports/{id}/pdf`	`values` を受け取りPDFを生成
`GET /api/reports/{id}/svg?page=0`	指定ページのSVG
`GET /api/reports/{id}/data`	自己完結PREPEJ
`POST /api/print-data/pdf`	自己完結PREPEJからPDFを生成
`POST /api/print-data/svg?page=0`	自己完結PREPEJからSVGを生成

最初は第3章のPDF作成を動かし、次に自分のPREPDJと項目名へ置き換えてください。そのプログラムを基に、必要な出力形式やWeb APIを追加していくのが最短です。

16. ライセンス

本製品は商用ライセンスです。評価、開発環境、サーバー配置、再配布、組み込み利用の条件は、製品に付属するライセンス文書を確認してください。