

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

製品版のライセンス配置と `ReportsProductInfo.loadLicense` の使い方は、製品版を使い始めるをご覧ください。体験版では登録不要です。

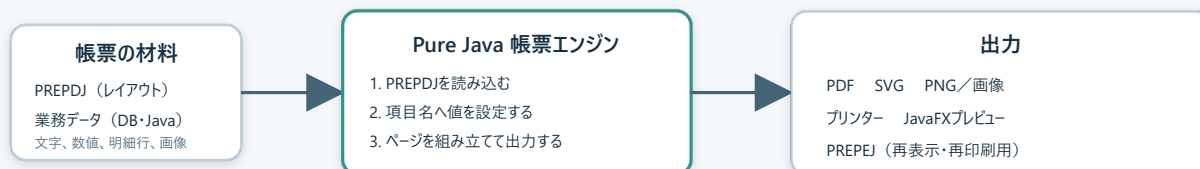
Reports.web Pure Javaは、ブラウザー内のWASMを使わず、Javaだけで帳票を作成・表示・保存・印刷するための帳票エンジンです。

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

## Reports.web Pure Java

ブラウザー内WASMを使わず、Javaだけで帳票を作成・保存・配信



同じエンジンを2つの場所で利用できます



帳票定義と出力処理を共通化できるため、ローカル用とWeb用で帳票ロジックを作り直す必要はありません。

# 目次

|                         |    |
|-------------------------|----|
| 1. 最初に理解する3つのデータ        | 5  |
| 2. 開発を始める               | 5  |
| 2.1 動作環境                | 5  |
| 2.2 パッケージ               | 6  |
| 3. 最初の帳票プログラム           | 6  |
| 4. 印刷データを設定する           | 8  |
| 4.1 1帳票に1つの値を設定する       | 8  |
| 4.2 明細行を設定する            | 8  |
| 4.3 DBの検索結果から明細を作る      | 9  |
| 4.4 印刷時だけ位置や色を変更する      | 9  |
| 5. ローカルアプリケーションで出力する    | 10 |
| 5.1 PDFを作る              | 10 |
| 5.2 SVGを作る              | 10 |
| 5.3 BufferedImageを作る    | 11 |
| 5.4 プリンターへ印刷する          | 11 |
| 5.5 デスクトップでプレビューする      | 12 |
| 5.6 デスクトップの操作UIを外側から隠す  | 12 |
| 6. 完成した帳票をPREPEJへ保存する   | 12 |
| 6.1 PREPEJを作る           | 13 |
| 6.2 PREPEJを読み、PDFへ再出力する | 13 |
| 6.3 値だけを保存する            | 13 |
| 7. Webアプリケーションで使う       | 14 |
| 7.1 業務データを受け取りPDFを返す    | 14 |

|                           |    |
|---------------------------|----|
| 7.2 SVGをWeb画面へ表示する        | 16 |
| 7.3 PREPEJをREST APIで受け取る  | 16 |
| 7.4 アプリ側からプレビューアを操作する     | 17 |
| 8. 実行の中止と進捗通知             | 19 |
| 9. 帳票定義を変換・保存する           | 19 |
| 9.1 帳票定義を保存する             | 19 |
| 9.2 読み込み時の診断を確認する         | 19 |
| 9.3 JSON文字列として扱う          | 20 |
| 10. APIリファレンス             | 20 |
| 10.1 ReportDefinitionIO   | 20 |
| 10.2 ReportData           | 21 |
| 10.3 ReportDataIO         | 21 |
| 10.4 PreviewJobIO         | 22 |
| 10.5 PreviewJob           | 22 |
| 10.6 PdfExporter          | 23 |
| 10.7 SvgExporter          | 23 |
| 10.8 ReportRenderer       | 24 |
| 10.9 PrinterOutput        | 24 |
| 10.10 RenderControl       | 25 |
| 11. 帳票モデルのプロパティ           | 25 |
| 11.1 ReportDocument       | 25 |
| 11.2 ReportObject 共通プロパティ | 26 |
| 11.3 文字プロパティ              | 27 |
| 11.4 線・図形・塗りのプロパティ        | 28 |
| 11.5 画像プロパティ              | 28 |

|                        |    |
|------------------------|----|
| 11.6 バーコードプロパティ        | 29 |
| 11.7 特定ページ・行・項目だけを変更する | 30 |
| 12. フォント、性能、安全性        | 31 |
| フォント                   | 31 |
| 性能                     | 31 |
| 安全性                    | 31 |
| 13. エラー処理の基本           | 32 |
| 14. 目的別の早見表            | 32 |
| 15. 付属サンプルを実行する        | 33 |
| 16. ライセンス              | 33 |

## 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／プレビューアを使う場合はJavaFX 21以上
- WebサーバーだけでPDFやSVGを作る場合、JavaFXは不要

ZIPを展開したら、Native/Pure-Java/samples/01.あっという間に帳票出力 をエディターやIDEで開いてください。ここにはソース、帳票定義、ビルド設定、起動手順が揃っています。最初はこのサンプルを土台にすると、ライブラリの配置で迷わずに始められます。

エンジンと依存ライブラリは **Native/Pure-Java/SDK/lib** にあります。例えば、この01フォルダーに置いた Gradleプロジェクトでは次のように参照します。同梱の **build.gradle.kts** は、この配置に対応済みです。

```
dependencies {  
    implementation(fileTree("../..../SDK/lib") { include("*.jar") })  
}
```

自分のプロジェクトへ移す場合も、エンジンJARだけでなく、ここにある依存JARと一緒に参照します。相対パスは新しいプロジェクトの場所に合わせて変更してください。製品ソースの **engine** プロジェクトを用意する必要はありません。

## 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

同梱の帳票定義を使い、タイトルと会社名を設定してPDFを1ページ作ってみましょう。

1. **Native/Pure-Java/samples/01**. あっという間に帳票出力 を開きます。
2. 元のソースを残したい場合は **src/main/java/quickreport/QuickReport.java** を別の場所へコピーします。同じソースフォルダーに別名の **.java** として残すと、クラスが重複するので注意してください。
3. **QuickReport.java** の内容を次のコードへ置き換えます。

```

package quickreport;

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 QuickReport {
    public static void main(String[] args) throws Exception {
        Path root = args.length == 0
            ? Path.of("").toAbsolutePath() : Path.of(args[0]).toAbsolutePath();
        Path definitionFile = root.resolve("definition/quick-report.prepdj");
        Path outputFile = root.resolve("output/first-report.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("Title", "はじめてのJava帳票");
        data.values.put("CustomerName", "株式会社サンプル");

        // この見本では、オブジェクト名と値のキーを結び付けます。
        for (var object : definition.objects) {
            if (data.values.containsKey(object.name)) object.binding = object.name;
        }

        // PDFを作る
        new PdfExporter().export(definition, data, outputFile);
        System.out.println("PDFを作成しました: " + outputFile);
    }
}

```

4. 01フォルダーのターミナルから、Windowsでは `./run.cmd`、macOS/Linuxでは `bash ./run.sh` を実行します。詳しいJDKの準備は、01のREADMEをご覧ください。

5. 作成された `output/first-report.pdf` を開き、タイトルと会社名を確認します。文字列を変えてもう一度実行すると、同じPDFを更新できます。

この例はPDFだけを作ります。PREPEJ・ページ単位SVGも作る元の01とは、出力ファイルが異なります。

`data.values` のキーは、文字オブジェクトの `binding`（連結する項目）と対応します。この見本ではコードでその対応を設定しています。自作の定義で既に `binding` を設定してあれば、上の `for` 文は不要です。

以降は機能別のコード例です。第3章のような単独実行プログラムではなく、使用中の定義・データ・接続先に合わせて組み込む部分を示します。請求書の明細やDB取得を含む動く実装は、02～04のサンプルと併せて確認してください。

## 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);
```

帳票定義側で繰り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 |

| 分類  | プロパティ名                                                                     |
|-----|----------------------------------------------------------------------------|
| 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プレビューアは、**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。

```

```

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ーアを操作する

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、プレビューア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ーアの外側に置きます。

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ーア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。

```
<!-- プレビューア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、プレビュー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ー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                                                                                |
|------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <code>hideAll()</code>                                                             | 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                                            |
| <code>showAll()</code>                                                             | 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                                                      |
| <code>getState()</code>                                                            | 現在のページ・倍率・表示設定などの状態を取得する                                                             |
| <code>subscribe("chrome-changed", handler)</code>                                  | 表示設定の変更を受け取る。戻り値は購読解除用関数                                                             |
| <code>setChrome({commands: {print: false}})</code>                                 | 印刷など特定のコマンドだけを隠す                                                                     |
| <code>showPdfPreview()</code> / <code>showSvgPreview()</code>                      | PDF表示／通常プレビューへ切り替える。切り替え処理は <code>await</code> で待てるが、ブラウザー内蔵PDFビューアーの読み完了を保証するものではない |
| <code>print()</code> / <code>savePrintData()</code> / <code>openPrintData()</code> | 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                                                     |
|-----------------------------------------------------------------|-----------------------------|--------------------------------------------------------|
| <code>read(Path path)</code>                                    | <code>ReportDocument</code> | ファイル名の拡張子を判定し、帳票定義を読み込む                                |
| <code>read(InputStream input, String sourceName)</code>         | <code>ReportDocument</code> | ストリームから読み込む。 <code>sourceName</code> には形式判定可能なファイル名を渡す |
| <code>readWithDiagnostics(...)</code>                           | <code>ReadResult</code>     | 帳票定義に加え、互換性に関する診断情報を返す                                 |
| <code>write(Path path, ReportDocument document)</code>          | <code>void</code>           | 拡張子に応じて帳票定義を保存する                                       |
| <code>writeCanonical(Path path, ReportDocument document)</code> | <code>WriteResult</code>    | 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                            |
|-----------------------------------------------------------------|-------------------------------|
| <code>export(definition, data, target)</code>                   | 全ページを標準設定のイメージPDFへ出力する        |
| <code>export(definition, data, target, options, control)</code> | PDF方式、解像度、フォント、中止・進捗を指定して出力する |
| <code>exportPages(pages, target, options, control)</code>       | 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                       |
|---------------------------------------------|--------------------------|
| <code>Options.image()</code>                | 100 DPIのイメージPDF          |
| <code>Options.image(dpi)</code>             | 指定DPIのイメージPDF            |
| <code>Options.vector(dpi, fontFiles)</code> | 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                                                 |
|-----------------------------|----------------------------------------------------|
| <code>mode()</code>         | 実際に使用した <code>IMAGE</code> または <code>VECTOR</code> |
| <code>pages()</code>        | 出力ページ数                                             |
| <code>renderingDpi()</code> | 描画に使用したDPI                                         |
| <code>diagnostics()</code>  | フォントなどの診断情報                                        |

## 10.7 SvgExporter

| メソッド                                                        | 戻り値                           | 説明                         |
|-------------------------------------------------------------|-------------------------------|----------------------------|
| <code>export(document, data, target)</code>                 | <code>void</code>             | 先頭ページをSVGファイルへ保存する         |
| <code>export(document, data, target, rasterDpi)</code>      | <code>void</code>             | SVG内で画像化される要素のDPIを指定して保存する |
| <code>svg(document, data)</code>                            | <code>String</code>           | 先頭ページのSVG文字列を返す            |
| <code>svg(document, data, pageIndex)</code>                 | <code>String</code>           | 指定ページのSVG文字列を返す            |
| <code>exportPages(document, data, directory, prefix)</code> | <code>List&lt;Path&gt;</code> | 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                                         |
|----------------------------------------------------------------------|----------------------------------------|--------------------------------------------|
| <code>render(document, data, options)</code>                         | <code>BufferedImage</code>             | 先頭ページを画像化する                                |
| <code>render(document, data, options, control)</code>                | <code>BufferedImage</code>             | 中止・進捗通知を指定して画像化する                          |
| <code>renderPages(document, data, options)</code>                    | <code>List&lt;BufferedImage&gt;</code> | 全ページを画像化する                                 |
| <code>renderPages(document, data, options, control)</code>           | <code>List&lt;BufferedImage&gt;</code> | 中止・進捗通知を指定して全ページを画像化する                     |
| <code>pageCount(document, data)</code>                               | <code>int</code>                       | 明細の繰返しを含めたページ数を返す                          |
| <code>renderPage(document, data, options, pageIndex, control)</code> | <code>BufferedImage</code>             | 指定ページだけを画像化する                              |
| <code>paintPage(graphics, document, data, dpi, page, control)</code> | <code>void</code>                      | 呼び出し側が用意した <code>Graphics2D</code> へ直接描画する |

RenderOptions :

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

## 10.9 PrinterOutput

| メソッド                                             | 説明                                          |
|--------------------------------------------------|---------------------------------------------|
| <code>print(definition, data, showDialog)</code> | 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                   |
| <code>createPageable(definition, data)</code>    | Java印刷APIで使用できる <code>Pageable</code> を返す   |
| <code>createPageable(pages)</code>               | 複数の帳票定義を含むページ一覧から <code>Pageable</code> を作る |

## 10.10 RenderControl

| メンバー                                                | 説明                                                   |
|-----------------------------------------------------|------------------------------------------------------|
| <code>new RenderControl(cancelled, progress)</code> | 中止判定と進捗通知を指定する                                       |
| <code>none()</code>                                 | 中止・進捗通知を使用しない設定                                      |
| <code>checkpoint()</code>                           | 中止が要求されていれば <code>CancellationException</code> を送出する |
| <code>progress(double value)</code>                 | 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                            |
|---------------------------------|----------------------|-------------------------------|
| <code>format</code>             | <code>String</code>  | 帳票形式名                         |
| <code>formatVersion</code>      | <code>int</code>     | 帳票形式のバージョン                    |
| <code>title</code>              | <code>String</code>  | 帳票名                           |
| <code>paperSize</code>          | <code>String</code>  | A4などの用紙名                      |
| <code>widthMm, heightMm</code>  | <code>double</code>  | 用紙の幅と高さ。単位はmm                 |
| <code>gridMm</code>             | <code>double</code>  | デザイナーのグリッド間隔。単位はmm            |
| <code>designScalePercent</code> | <code>double</code>  | デザイナーの表示倍率                    |
| <code>showGrid</code>           | <code>boolean</code> | グリッドを表示するか                    |
| <code>snapToGrid</code>         | <code>boolean</code> | オブジェクトをグリッドへ合わせるか             |
| <code>objectsLocked</code>      | <code>boolean</code> | 帳票全体のオブジェクトをロックするか            |
| <code>defaultFontFamily</code>  | <code>String</code>  | 新規文字オブジェクトの既定フォント             |
| <code>defaultFontSizePt</code>  | <code>double</code>  | 新規文字オブジェクトの既定サイズ              |
| <code>isInch</code>             | <code>boolean</code> | 用紙サイズの編集表示をインチにする。内部の寸法はmmのまま |

| プロパティ                                                       | 型                   | 説明                         |
|-------------------------------------------------------------|---------------------|----------------------------|
| designColor                                                 | String              | 新規オブジェクトの文字・線の既定色。背景色ではない  |
| defaultStrokeWidth, defaultStrokeStyle, defaultDashPattern  | 線幅・線種・破線            | 新規オブジェクトの既定線設定             |
| displayRepeatObjects, repeatObjectColor                     | 表示・色                | デザイナーの繰返し補助表示。印刷内容には加えない   |
| backgroundX, backgroundY, backgroundWidth, backgroundHeight | double              | 帳票背景画像の位置・寸法。mm。幅・高さ0は用紙寸法 |
| backgroundImagePath, backgroundImageDataBase64              | String              | 帳票背景画像のファイルまたは埋込データ        |
| backgroundImageSizeMode, backgroundImagePosition            | String              | 帳票背景画像のサイズ方法・配置位置          |
| 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       |
| 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                               |

| プロパティ        | 説明                   |
|--------------|----------------------|
| 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 <code>fontUnit</code> に従う                                                                                                                                                 |
| fontUnit                                | 文字サイズ単位。通常は <code>Point</code> 。従来の <code>World</code> 、 <code>Display</code> 、 <code>Pixel</code> 、 <code>Inch</code> 、 <code>Document</code> 、 <code>Millimeter</code> も出力解像度に合わせて換算 |
| bold, italic, underline, strikeout      | 太字、斜体、下線、取消線                                                                                                                                                                           |
| 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                                                                                                                                                                            |
| elasticOverCharacters, elasticOverBytes | 指定文字数・バイト数を超えたときだけ伸縮する。0は条件なし                                                                                                                                                          |
| justify                                 | 通常文字を均等割り付けする                                                                                                                                                                          |
| justifyHalfWidth, justifyHalfWidthKana  | 半角英数字・半角カナを均等割り付けの対象にする                                                                                                                                                                |
| justifyExcludedCharacters               | 指定文字の連続部分を一まとまりとして扱う                                                                                                                                                                   |
| autoJustifyEnds, endSpaceCount          | 両端の自動空白・指定数の空白                                                                                                                                                                         |
| pdfAutoWrap, pdfClipOverflow            | PDFでの自動折返し・高さによる切取り                                                                                                                                                                    |
| pdfFontFamily                           | PDF出力に使用する専用フォント。空文字は通常フォント                                                                                                                                                            |
| textColor                               | #RRGGBB 形式の文字色                                                                                                                                                                         |

| プロパティ                              | 説明        |
|------------------------------------|-----------|
| textBorderEnabled                  | 文字枠を表示するか |
| textOutlineWidth, textOutlineColor | 文字の輪郭幅と色  |

伸縮はtextElastic=trueで有効になります。しきい値と同じ長さでは通常表示し、超えた場合だけ伸縮します。文字数はUTF-16の単位数、バイト数は従来のReports.NETに合わせたCP932で数えます。文字数とバイト数は選択して使い、デザイナーでは一方を設定すると他方を解除します。通常の伸縮は縦横を枠に合わせ、高さ固定では指定フォントの高さを保ったまま横方向を合わせます。

## 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       |

| プロパティ                           | 説明        |
|---------------------------------|-----------|
| <code>imageAlignment</code>     | 枠内の配置位置   |
| <code>imageReverse</code>       | 左右・上下の反転  |
| <code>imageBorderEnabled</code> | 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                                   |
|---------------------------------------------------------------------------|--------------------------------------|
| <code>barcodeKind</code>                                                  | CODE128、QR_CODEなどのバーコード種別            |
| <code>barcodeDrawMode</code>                                              | STRETCH または DIRECT                   |
| <code>barcodeTextVisible</code>                                           | バーコード下の文字を表示するか                      |
| <code>barcodeJustify</code>                                               | バー幅を枠へ合わせるか                          |
| <code>barcodeEvenSpacing</code>                                           | 文字間隔を均等にするか                          |
| <code>barcodeBlackBarAdjusterByDot</code>                                 | 黒バー幅のドット補正                           |
| <code>barcodeImageMargin</code>                                           | 画像余白                                 |
| <code>barcodeStartStop</code>                                             | スタート／ストップ文字を表示するか                    |
| <code>barcodeCodeSet</code>                                               | CODE128などのコードセット                     |
| <code>qrErrorCorrection</code>                                            | QRコードの誤り訂正レベル                        |
| <code>qrVersion</code>                                                    | QRコードの型番                             |
| <code>stringEncoding</code>                                               | 文字エンコーディング                           |
| <code>pdf417Rows, pdf417Columns</code>                                    | PDF417の行数と列数                         |
| <code>barcodeWhiteBarAdjustment</code>                                    | 対応するGS1系バーコードの白バー補正                  |
| <code>barcodeTextFontFamily,</code><br><code>barcodeTextFontSizePt</code> | バーコード下の文字の専用フォント・サイズ。空文字・0は通常フォントを使用 |
| <code>postalCustomerPoint</code>                                          | 郵便カスタマバーコードのポイント指定                   |
| <code>dataMatrixCodeSize</code>                                           | DataMatrixのコードサイズ                    |

| プロパティ                                  | 説明                  |
|----------------------------------------|---------------------|
| pdf417SizeKind, pdf417AspectRatio      | PDF417のサイズ決定方法・縦横比  |
| pdf417ErrorLevel, pdf417AutoErrorLevel | PDF417の誤り訂正レベル・自動設定 |

## 11.7 特定ページ・行・項目だけを変更する

ReportPageBuilderは、帳票定義をコピーし、1ページ分の繰返しを展開してから、指定した項目の属性を変更するためのクラスです。フォント専用ではなく、文字色、位置、罫線、表示・非表示など、通常のReportObject属性を変更できます。

```
var pages = new java.util.ArrayList<ReportDocument>();
for (int pageNumber = 1; pageNumber <= 3; pageNumber++) {
    var builder = new ReportPageBuilder(definition);
    if (pageNumber == 2) {
        // オブジェクト名が「金額」の5行目だけを変更します。行番号は0始まりです。
        var target = builder.object("金額", 4);
        target.fontSizePt = 12;
        target.fontUnit = "Point";
        target.textColor = "#0000FF";
    }
    pages.add(builder.build());
}
```

ReportPageBuilderとReportDocumentはcom.pao.reports.engine.modelにあります。この例では、definitionに「金額」という名前の項目があり、1ページ内で5行以上繰返す設定になっているものとします。build()したページは独立したコピーです。後からBuilderを変更しても、元の定義、生成済みのページ、別ページには反映されません。

| 指定方法                      | 対象               |
|---------------------------|------------------|
| builder.object("金額")      | 最初の1件            |
| builder.object("金額", 4)   | 5行目の1件           |
| builder.cell("ラベル", 1, 2) | XY繰返しの2列目・3行目の1件 |

名前の重複、存在しない項目、範囲外の行・列はエラーになります。XY繰返しにはcell()を使用してください。生成した各ページはPdfExporter.Pageに渡し、exportPages()で1つのPDFにまとめられます。

## 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。

#### MS ゴシックがない環境での動作

MS ゴシック (MS Gothic、MS-Gothic) を指定した場合は、指定フォントを優先します。使えない場合は既存の等幅代替フォントを順に探し、それも使えない場合に、Javaエンジンに同梱した **BIZ UDゴシック** (比例幅のBIZ UDPゴシックではありません) を使用します。利用者による追加インストールやインターネットからの取得は不要です。WindowsでMS ゴシックが使える場合は、従来どおりMS ゴシックを使用します。

この代替はMS ゴシック系の指定に限ります。MS Pゴシック・明朝体・未知のフォントに一律適用するものではなく、同梱フォントにない文字には従来の字形代替処理を使用します。標準・太字には同梱の各書体を使い、斜体はJavaの合成表示です。元のMS ゴシックと字形・文字幅が完全に同一とは限らないため、使用する帳票のプレビューとPDFを確認してください。

**同梱フォントとPDFへの埋込みは別です。** フォントがエンジンに同梱されていても、PDFへ自動的に埋め込まれるわけではありません。画像PDFの本文は画像です。ベクターPDFでは、フォントファイルを指定しない場合は文字を輪郭として描画し、その文字は検索・抽出できません。同梱BIZ UDゴシックが選択されている場合に、同じ公式TTFのパスを`Options.vector(dpi, fontFiles)`へ明示すると、条件を満たす標準・太字を埋め込みます。合成斜体などは輪郭での出力になります。BIZのTTFを渡すだけで、使用中のMS ゴシックがBIZに切り替わることはありません。

同梱の公式BIZ UDゴシックはSIL Open Font License 1.1による配布です。ライセンス全文と取得元はengine JAR内の`com/pao/reports/engine/fonts/biz-ud-gothic/OFL.txt`と`provenance.json`に収録しています。

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、画像読み込み、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                                 |

| やりたいこと      | 最初に見る箇所 | 主なAPI          |
|-------------|---------|----------------|
| 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

ZIPを全体ごと展開し、`Native/Pure-Java/samples/README.html`を開いてください。開発者の作業フォルダーや、本製品全体のソースビルド環境は不要です。

| 番号フォルダー              | 学ぶこと                                    |
|----------------------|-----------------------------------------|
| 01. あっという間に帳票出力      | 短いコードで定義読込・値設定・帳票出力を試す                  |
| 02. 全帳票をパソコンでプレビュー   | 同じサンプルデータをSQLiteから読み、PC上でプレビュー・PDF・印刷する |
| 03. 全帳票をWebでプレビュー    | Javaエンジンを使うWebアプリで帳票を表示する               |
| 04. Web API・デスクトップ連携 | サーバーで作った印刷データをHTTPで受け取り、PCで表示する         |

使いたい番号フォルダーをエディターやIDEで開き、その中のREADMEの手順で起動します。READMEには、必要なJava環境、実行コマンド、読むソース、変更する箇所、停止方法をまとめています。Java 21を基準に、同梱SDKと同じ版のサンプルを使ってください。

03のWebアプリと04のRESTサーバーは、直接起動する方法に加えてDockerからの起動も選べます。04のデスクトップ画面はDockerに入れず、PC側で直接起動します。画面のURLとAPIのパスは、各サンプルのREADMEにあるものを使ってください。

最初は01のタイトルや会社名を書き換えて、帳票へ反映されることを確認してください。次に02以降へ進むと、同じ基本を明細帳票やWeb連携へ広げられます。

## 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。