Reports.web WPF／.NET版 プログラマーズガイド／APIリファレンス

Reports.web WPF／.NET版は、C#から帳票定義へ業務データを差し込み、プレビュー、PDF、ページ単位のSVG／PNG、プリンター出力、印刷データの保存まで行うための帳票エンジンです。WindowsではWPF版デザイナーとプレビュアーを利用でき、Webサーバーでは画面を起動せずに帳票を生成できます。

この版はブラウザー内のWASMを呼び出すC#ラッパーではありません。.NETで実装した帳票エンジンをC#から直接呼び出す構成です。共通WASMを使うWebアプリケーションについては、別冊の「C#／.NET WASM言語ラッパー プログラマーズガイド」を参照してください。

この一冊では、次の処理を実装できることを目標にします。

- PREPDJを読み、C#から値と明細を設定する
- 帳票全体を1つのPDFへ出力する
- SVG／PNGをページ単位で出力する
- WPFプレビュアーを業務アプリケーションから開く
- 完成した帳票をPREPEJへ保存し、再表示・再印刷する
- ASP.NET CoreからPDF、SVG、PREPEJを返す

画面の使い方は、WPF版 帳票デザイナー操作ガイドとWPF版 帳票プレビュアー操作ガイドを参照してください。

目次

1. 最初に理解する3つのデータ

帳票は「レイアウト」と「今回印刷する値」を分けて扱います。

データ	C#の型	内容
帳票定義	`ReportDocument`	用紙、文字、罫線、画像、バーコード、明細の繰返しなどのレイアウト
作成中の印刷データ	`PrintDataDocument`	帳票定義と、ページごとの値・属性
出力用の完成ページ	`IReadOnlyList<ResolvedReportPage>`	値を帳票定義へ反映した、PDF・プレビュー・印刷に渡せるページ一覧

主なファイル形式は次のとおりです。

拡張子	内容	主な用途
`.prepdj`	JSON形式の帳票定義	デザイナーで作成し、C#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自己完結した印刷データ	保存、受け渡し、再表示、再印刷
`.prepe`	従来形式の印刷データ	既存データの読み込み

通常は次の順序で処理します。

1. `ReportCreator.Create()` で帳票処理を開始する
2. `LoadDefFile(...)` でPREPDJまたはPREPDを読む
3. `PageStart()`、`Write(...)`、`PageEnd()` でページを作る
4. PDF、SVG、PNG、プレビュー、プリンターのいずれかへ出力する
5. 後から同じ帳票を開く必要があれば `SaveJson(...)` でPREPEJを保存する

2. 開発環境と参照先

2.1 推奨環境

Windowsアプリへ組み込むDLLは、次の2系統から選びます。**.NET 10へ移行しないと帳票エンジンを使えない、ということではありません。**

お使いのアプリの環境	使用する配布SDK
.NET Framework 4.7.2以降	`SDK/net472`：Framework互換版
.NET 5〜9／Windows	`SDK/net472`：Framework互換DLLを参照して使用
.NET 10／Windows	`SDK/net10`：.NET 10版

どちらもReports.webの帳票エンジンとWPFプレビュアーです。旧Reports.NETのエンジンへ置き換える方式ではありません。使用する主なクラス・メソッドは共通で、`Pao.Reports.Maui.Engine.dll` と `Pao.Reports.Wpf.Windows.dll` および依存DLLを、選んだSDKからまとめて参照します。2系統のDLLを混ぜないでください。

Framework互換版は、net472アプリでの実請求書PDF出力・プレビュー表示と、同じ配布DLLを.NET 8／9アプリから直接参照したPDF出力を確認しています。.NET 8でのプレビュー表示も確認済みです。.NET 5〜9のすべての版・依存パッケージの組み合わせを実測した、という意味ではありません。既存システム独自の依存関係で問題がある場合や、.NET Framework 4.7.1以前で使用する場合は、環境とエラー内容を添えて info@pao.ac へご相談ください。

配布する01〜04の学習サンプルの標準プロジェクトは.NET 10です。Linuxで動くWeb APIも.NET 10の帳票エンジンを使用し、Windows専用のWPFプレビュアーは使用しません。デザイナーは帳票を作るための別ツールであり、アプリが参照するエンジンDLLの選択とは分けて考えます。

2.2 配布SDKを参照する際の注意

`SDK/net472/README.md` にFrameworkアプリのbinding redirect設定と、.NET 8から参照する場合の例を記載しています。製品DLLだけでなく、同じSDKの `System.Text.Json.dll` などの依存DLLも参照し、出力先へコピーしてください。実行環境に入っている同名DLLが、そのまま必要な版と一致するとは限りません。独自のアセンブリ読み込み処理を追加する必要はありません。

Skiaのネイティブ DLLはアプリのビット数に合わせて配置します。x64アプリでは `runtimes/win-x64/native/libSkiaSharp.dll`、x86アプリでは `runtimes/win-x86/native/libSkiaSharp.dll` を実行ファイルと同じフォルダーへコピーします。具体的な参照例は、選んだSDKのREADMEをご覧ください。

2.3 開発リポジトリ内で参照する

現在のソース構成でC#プロジェクトから使用する場合は、用途に応じて次のプロジェクトを参照します。

```
<ItemGroup>
  <ProjectReference Include="..\Pao.Reports.Maui\Engine\Pao.Reports.Maui.Engine.csproj" />
  <ProjectReference Include="..\Pao.Reports.Wpf\src\Pao.Reports.Wpf.Windows\Pao.Reports.Wpf.Windows.csproj" />
</ItemGroup>
```

- `Pao.Reports.Maui.Engine`：帳票定義、印刷データ、解決処理、SkiaSharp PDF／SVG
- `Pao.Reports.Wpf.Windows`：Windows GDI+描画、WPFプレビュー、プリンター、Windows PDF

- `Pao.Reports.WindowsServer`：Windowsサーバー用GDI+／PDF。WPF、WinForms、プリンター列挙に非依存

製品配布物をDLLで組み込む場合は、同梱されている対象フレームワーク用DLLと依存DLLをまとめて参照してください。

3. 最初の帳票プログラム

次の例は `invoice.prepdj` を読み、値を設定し、PREPEJとPDFを作ります。

```
using Pao.Reports.Maui.Engine.Rendering;
using Pao.Reports.Maui.Engine.Reports;
using Pao.Reports.Wpf.Windows;

var report = ReportCreator.Create();
report.LoadDefFile("reports/invoice.prepdj");

report.PageStart();
report.Write("顧客名", "株式会社サンプル");
report.Write("請求日", "2026年9月14日");
report.Write("合計金額", "120,000 円");
report.PageEnd();

Directory.CreateDirectory("output");

// 帳票定義と値を含む、再表示可能な印刷データ
report.SaveJson("output/invoice.prepej");

// Windows互換PDF。解決済みの全ページを1つのPDFにする
var pages = report.ResolvePages().Select(page => page.Document).ToArray();
var pdf = new WindowsPdfExporter().Render(
    pages, new PdfRenderOptions { Title = "請求書" });
File.WriteAllBytes("output/invoice.pdf", pdf);
```

`Write(...)` の名前には、デザイナーで設定したオブジェクト名を指定します。存在しない名前を渡すと例外になります。`PageStart()` と `PageEnd()` の間だけ値を書き込めます。

4. 値と明細を設定する

4.1 1ページに1つの値

```
report.PageStart();
report.Write("伝票番号", "INV-2026-0012");
report.Write("顧客名", "株式会社サンプル");
report.Write("発行日", DateTime.Today.ToString("yyyy年M月d日"));
report.Write("合計金額", 120000.ToString("N0") + " 円");
report.PageEnd();
```

表示形式は業務プログラム側で文字列へ整形してから渡すと明確です。日付や金額の書式を帳票ごとに変える場合も、値の作成処理を一か所へまとめると保守しやすくなります。

4.2 繰返し明細

繰返し設定されたオブジェクトへは、1から始まる明細番号を指定します。

```
report.PageStart();

var lines = new[]
{
    new { Name = "商品A", Quantity = 2, Price = 50000 },
    new { Name = "商品B", Quantity = 1, Price = 20000 }
};

for (var index = 0; index < lines.Length; index++)
{
    var row = index + 1;
    report.Write("商品名", lines[index].Name, row);
    report.Write("数量", lines[index].Quantity.ToString("N0"), row);
    report.Write("単価", lines[index].Price.ToString("N0"), row);
}

report.PageEnd();
```

縦横2方向の繰返しは、互換形式の1始まりインデックスを使う `Write(name, value, indexX, indexY)`、または0始まりの `WriteAt(name, value, indexX, indexY)` を使用します。2つを同じ処理で混在させないでください。

4.3 DBの検索結果を流し込む

```
report.PageStart();

await using var command = connection.CreateCommand();
command.CommandText = "select product_name, quantity, unit_price from invoice_line where invoice_id = @id";
var parameter = command.CreateParameter();
parameter.ParameterName = "@id";
parameter.Value = invoiceId;
command.Parameters.Add(parameter);

await using var reader = await command.ExecuteReaderAsync();
var row = 1;
while (await reader.ReadAsync())
{
    report.Write("商品名", reader.GetString(0), row);
    report.Write("数量", reader.GetInt32(1).ToString("N0"), row);
    report.Write("単価", reader.GetDecimal(2).ToString("N0"), row);
    row++;
}

report.PageEnd();
```

DB列名と帳票オブジェクト名の対応は、このようにC#コード上で明示すると、DBまたは帳票の変更箇所を見つけやすくなります。

4.4 複数ページを作る

`PageStart()` から `PageEnd()` までが1ページです。同じ帳票定義でページを増やす場合は、この組合せを繰り返します。途中で `LoadDefFile(...)` を呼び、表紙と明細などで帳票定義を切り替えることもできます。

作り直す場合は、ページ作成中でないことを確認して `ClearData()` を呼びます。帳票定義は保持され、完成済みページだけが消去されます。

5. 帳票全体をPDFへ出力する

PDFは、解決済みの全ページを1つのPDFファイルへまとめます。SVG／PNGのページ単位出力とは扱いが異なります。

5.1 Windows互換PDF

Windows上でReports.NETのGDI+描画とPDFライターを利用する経路です。

```
using Pao.Reports.Maui.Engine.Rendering;
using Pao.Reports.Wpf.Windows;

var documents = report.ResolvePages()
    .Select(page => page.Document)
    .ToArray();

var exporter = new WindowsPdfExporter();
var bytes = exporter.Render(
    documents,
    new PdfRenderOptions
    {
        Title = "請求書",
        Progress = (current, total) =>
            Console.WriteLine($"{current}/{total}")
    },
    cancellationToken);

await File.WriteAllBytesAsync("output/invoice.pdf", bytes, cancellationToken);
```

各ページを画像化してからPDFへ格納する場合は `RenderImagePdf(...)` を使用します。通常PDFとImagePDFは別の出力方式です。

5.2 ポータブルなSkiaSharp PDF

帳票エンジンの `SavePdf(...)` はSkiaSharp経路を使用し、帳票全体を出力します。

```
report.SavePdf(
    "output/invoice-skia.pdf",
    new PdfRenderOptions { Title = "請求書", RasterDpi = 144 },
    cancellationToken);
```

バイト配列、`Stream`、非同期APIも用意されています。

```
byte[] bytes = await report.SavePdfAsync(
    new PdfRenderOptions { Title = "請求書" },
    cancellationToken);
```

Windowsで既存Reports.NETとの見た目を優先するときは `WindowsPdfExporter`、Windowsに依存しないサーバー処理には `Report.SavePdf(...)` または `SkiaPdfRenderer` を選びます。

6. SVG／PNGをページ単位で出力する

6.1 SVG

`GetSvgPages()` は、1ページにつき1つのSVG文字列を返します。

```
var svgPages = report.GetSvgPages();

for (var index = 0; index < svgPages.Count; index++)
{
    var path = $"output/invoice-{index + 1}.svg";
    await File.WriteAllTextAsync(path, svgPages[index], cancellationToken);
}
```

解決済みページから直接作る場合は `SvgReportRenderer.Render(...)` を使います。

```
var svgRenderer = new SvgReportRenderer();
var pages = report.ResolvePages();
var svg = svgRenderer.Render(pages[0].Document); // 先頭ページ
```

`SaveSVGFile(...)` は全ページを閲覧・印刷できるHTMLとして保存します。ページごとの純粋な `.svg` が必要な場合は、上記の `GetSvgPages()` または `SvgReportRenderer` を使用してください。

6.2 PNG

Windows版では、解決済みページを `GdiPageRenderer` で1ページずつ画像化します。

```
using System.Drawing.Imaging;
using Pao.Reports.Wpf.Windows;

var renderer = new GdiPageRenderer();
var pages = report.ResolvePages();

for (var index = 0; index < pages.Count; index++)
{
    using var bitmap = renderer.RenderBitmap(pages[index].Document, dpi: 144);
    bitmap.Save($"output/invoice-{index + 1}.png", ImageFormat.Png);
}
```

PDFは全ページ、SVG／PNGはページ単位、という違いを意識すると、Web APIのURLや保存先を設計しやすくなります。

7. PREPEJを保存・再利用する

PREPEJは、帳票定義とページごとの値をまとめた自己完結形式です。元のPREPDJやDBへ再接続しなくても、ファイルだけで再表示・PDF作成・再印刷できます。

7.1 保存する

```
report.SaveJson("output/invoice.prepej");
```

XML形式が必要な場合は `SaveXml(...)` を使用します。通常の新規処理にはJSON形式のPREPEJを推奨します。

7.2 読み直す

```
var reopened = ReportCreator.Create();
reopened.LoadData("output/invoice.prepej");

Console.WriteLine($"{reopened.PageCount}ページ");
reopened.SavePdf("output/invoice-reprint.pdf");
```

データ層だけで扱う場合は `PrintDataStore` を使用します。

```
var data = PrintDataStore.LoadJson("output/invoice.prepej");
PrintDataStore.Validate(data);
var resolved = PrintDataStore.Resolve(data);
```

`LoadData(...)` と `PrintDataStore.Load(...)` はXML／JSONを内容から判定できます。公開Web APIで受け取る場合は、形式判定だけに頼らずファイルサイズ、ページ数、画像、診断結果も検証してください。

8. WPFプレビュアーをC#から開く

`ReportPreviewWindow` は、業務アプリケーションから所有して表示できる公開WPF Windowです。PREPEJの保存・再読込を有効にする場合は `PrintDataDocument` を渡します。

```
using Pao.Reports.Maui.Engine.Data;
using Pao.Reports.Wpf.Windows;

var data = PrintDataStore.LoadJson("output/invoice.prepej");
var preview = new ReportPreviewWindow(data, new ReportPreviewOptions
{
    Title = "請求書プレビュー",
    InitialPage = 1,
    InitialZoomMode = PreviewZoomMode.FitPage,
    InitialZoomPercent = 100,
    InitialDataPath = "output/invoice.prepej",
    InitialPdfPath = "output/invoice.pdf",
    ShowOpenDataCommand = true,
    ShowSaveDataCommand = true,
    ShowPdfCommand = true,
    ShowPrintCommand = true,
    ShowPageNavigation = true,
    AllowDataDrop = true
});

preview.Owner = this;
preview.ShowDialog();
```

主な公開操作は次のとおりです。

操作	API
ページ移動・倍率	`GoToPage(...)`、`SetZoom(...)`、`PreloadPages(...)`
印刷データ	`LoadData(...)`、`SaveData(...)`
検索	`Search(...)`、`ClearSearch()`
帳票全体のPDF	`ExportPdfAsync(...)`、`ExportImagePdfAsync(...)`
現在ページの画像	`ExportCurrentPngAsync(...)`、`ExportCurrentSvgAsync(...)`
表示領域	`SetChromeVisibility(...)`
操作UIの一括表示・非表示	`SetPreviewChromeVisible(bool)`

8.1 プレビューだけを表示し、アプリ側から操作する

```
preview.SetPreviewChromeVisible(false); // 設定を含むツールバーとステータスを隠す
await preview.ExportPdfAsync("output/invoice.pdf"); // 非表示中もPDFを保存できる
preview.SetPreviewChromeVisible(true);  // 全操作UIを再表示する
```

`preview` は上で作成した `ReportPreviewWindow`、出力フォルダーは作成済みとします。表示切り替えはWPFのUIスレッドから呼び出します。通常の初期表示は変えず、`false` を指定したときだけ帳票表示に絞ります。設定ボタンも隠れるので、アプリ側に再表示する操作を用意してください。OSのウィンドウ枠は残ります。このAPIはFramework互換版と.NET 10版の両方で利用できます。

`ReportPreviewWindow` は再利用可能なWindowです。別WindowのVisual Treeへ直接挿入する公開UserControlではありません。より詳しい制約と表示設定はプレビュアー操作ガイドを参照してください。

9. Windowsプリンターへ出力する

```
using Pao.Reports.Wpf.Windows;

var documents = report.ResolvePages()
    .Select(page => page.Document)
    .ToArray();

var options = new WindowsPrintOptions
{
    PrinterName = "",
    Copies = 1,
    LeftHundredthsInch = 0,
    RightHundredthsInch = 0,
    TopHundredthsInch = 0,
    BottomHundredthsInch = 0,
    ScaleToFitMargins = false
};

using var printDocument = new GdiPageRenderer()
    .CreatePrintDocument(documents, "請求書", options);
printDocument.Print();
```

空の `PrinterName` は既定プリンターを使用します。実際の印刷前には、WPFプレビュアーの印刷設定画面でプリンター、部数、余白、余白内への縮小を利用者に確認してもらう方法もあります。

サーバー処理ではプリンター名を外部入力から無制限に指定させず、利用可能なプリンターをアプリケーション側で限定してください。

10. ASP.NET Coreで使う

WPF／.NET版には、WebAssemblyを使わず、サーバー側の.NETエンジンがPDF・SVG・PREPEJを返すサンプルがあります。

- Windows：GDI+とWindows PDFライターを使用。WPF、WinForms、プリンター機能には依存しない
- Linux：SkiaSharpを使用

10.1 PDFを返す

```
app.MapGet("/api/invoices/{id}/pdf", async (
    long id,
    CancellationToken cancellationToken) =>
{
    var report = await invoiceService.CreateReportAsync(id, cancellationToken);
    var documents = report.ResolvePages()
        .Select(page => page.Document)
        .ToArray();

    var pdf = new Pao.Reports.WindowsServer.WindowsServerPdfExporter().Render(
        documents,
        new PdfRenderOptions { Title = $"請求書 {id}" },
        cancellationToken);

    return Results.File(pdf, "application/pdf", $"invoice-{id}.pdf");
});
```

この例の `invoiceService` はアプリケーション側のサービスです。Linuxでは同じ `ReportDocument` 一覧を `SkiaPdfRenderer` へ渡します。

10.2 ページ単位のSVGを返す

```
app.MapGet("/api/invoices/{id}/svg", async (
    long id,
    int? page,
    CancellationToken cancellationToken) =>
{
    var report = await invoiceService.CreateReportAsync(id, cancellationToken);
    var pages = report.ResolvePages();
    var index = page ?? 0;

    if (index < 0 || index >= pages.Count)
        return Results.BadRequest(new { error = "ページ番号が範囲外です。" });

    cancellationToken.ThrowIfCancellationRequested();
    var svg = new SvgReportRenderer().Render(pages[index].Document);
    return Results.Content(svg, "image/svg+xml; charset=utf-8");
});
```

SVGのページ番号は0から始める設計にしています。PDFは帳票全体、SVGは指定した1ページを返します。

10.3 PREPEJを受け取る

公開APIでアップロードを受け付ける場合は、リクエストを無制限にメモリーへ読み込まないでください。製品サンプルは次の境界を実装しています。

- `application/json` または `application/vnd.pao.reports-printdata+json` のみ許可
- 8MiBを超える本文はHTTP 413
- 自己完結したPREPEJだけを許可
- 外部画像パスを拒否し、埋め込み画像のBase64を検証
- 同時生成数、待機時間、処理時間を制限
- 読み込み診断があれば不正データとして扱う

サンプルの実在APIは次のとおりです。

API	内容
`GET /api/samples`	サンプル帳票一覧
`GET /api/reports/{id}/data`	自己完結PREPEJ
`GET /api/reports/{id}/pdf`	帳票全体のPDF
`GET /api/reports/{id}/svg?page=0`	指定ページのSVG
`POST /api/print-data/pdf`	アップロードしたPREPEJから帳票全体のPDFを生成
`POST /api/print-data/svg?page=0`	アップロードしたPREPEJから指定ページのSVGを生成

10.4 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

API	用途・注意
`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。

11. 中止と進捗通知

PDF出力は `CancellationToken` と `PdfRenderOptions.Progress` に対応しています。

```
using var timeout = new CancellationTokenSource(TimeSpan.FromSeconds(60));

var options = new PdfRenderOptions
{
    Title = "月次帳票",
    Progress = (current, total) =>
        Console.WriteLine($"PDF {current}/{total}")
};

var documents = report.ResolvePages()
    .Select(page => page.Document)
    .ToArray();

var pdf = new WindowsPdfExporter().Render(
    documents, options, timeout.Token);
```

Web APIでは、HTTP切断の `RequestAborted` とアプリケーションのタイムアウトを連結し、同時生成数にも上限を設けてください。

12. 主要APIリファレンス

12.1 ReportCreator／IReport

API	説明
`ReportCreator.Create()`	新しい帳票処理を作る
`LoadDefFile(path／stream／bytes)`	PREPD／PREPDJ帳票定義を読む
`LoadData(path／stream／bytes)`	PREPE／PREPEJ印刷データを読む
`PageStart()／PageEnd()`	1ページの開始／確定
`Write(name, value)`	名前が一致するオブジェクトへ値を設定
`Write(name, value, index)`	繰返しまたは同名オブジェクトへ値を設定
`WriteAt(name, value, indexX, indexY)`	0始まりの縦横位置へ値を設定
`ClearData()`	帳票定義を残し、完成ページを消去
`ResolvePages()`	出力用の解決済みページ一覧を返す
`SaveJson(...)／SaveXml(...)`	完成した印刷データをPREPEJ／PREPEとして保存
`SavePdf(...)／SavePdfAsync(...)`	SkiaSharpで帳票全体をPDF出力
`GetSvgPages()`	ページごとのSVG文字列一覧を返す

`State`、`PageCount`、`IsPageOpen` で現在の処理状態も確認できます。

12.2 PrintDataStore

API	説明
`Load(...)`	XML／JSONを内容から判定して読み込む
`LoadJson(...)／LoadXml(...)`	形式を指定して読み込む
`SaveJson(...)／SaveXml(...)`	`PrintDataDocument` を保存する
`Validate(...)`	構造と値を検証する
`Resolve(...)`	値を反映した `ResolvedReportPage` 一覧を作る

12.3 Windows出力

型／API	単位	説明
`WindowsPdfExporter.Render(...)`	帳票全体	Windows互換PDFを返す
`WindowsPdfExporter.RenderImagePdf(...)`	帳票全体	全ページを画像化したPDFを返す
`GdiPageRenderer.RenderBitmap(...)`	1ページ	`Bitmap` を返す
`GdiPageRenderer.CreatePrintDocument(...)`	帳票全体	`PrintDocument` を作る
`ReportPreviewWindow.ExportPdfAsync(...)`	帳票全体	プレビュアーからPDF保存
`ReportPreviewWindow.ExportCurrentSvgAsync(...)`	現在ページ	SVG保存
`ReportPreviewWindow.ExportCurrentPngAsync(...)`	現在ページ	PNG保存

12.4 ポータブル出力

型／API	単位	説明
`SkiaPdfRenderer.Render(...)`	渡した全ページ	Windowsに依存しないPDF出力
`SvgReportRenderer.Render(...)`	1ページ	SVG文字列を返す
`SvgReportRenderer.Save(...)`	1ページ	SVGファイルを保存
`SvgReportRenderer.RenderHtml(...)`	渡した全ページ	複数SVGを含むHTMLを返す

13. 帳票定義をC#で扱う

通常はデザイナーでPREPDJを作り、C#では値だけを書き込みます。帳票定義を直接読む場合は `JsonDesignStore` を利用できます。

```
using Pao.Reports.Maui.Engine.Services;

var definition = await JsonDesignStore.LoadAsync(
    "reports/invoice.prepdj", cancellationToken);

definition.Title = "請求書（社内控え）";

await JsonDesignStore.SaveAsync(
    definition,
    "reports/invoice-copy.prepdj",
    cancellationToken);
```

主なモデルは次のとおりです。

型	主な内容
`ReportDocument`	`Title`、`Paper`、用紙寸法、グリッド、既定フォント、背景、`Objects`
`ReportObject`	`Name`、`Kind`、X／Y、幅／高さ、回転、表示、文字、画像、線、塗り、繰返し、バーコード
`PrintDataDocument`	共通帳票定義と `Pages`
`PrintDataPage`	ページ番号、ページ固有の帳票定義、値、動的属性
`ResolvedReportPage`	出力用ページ番号、解決済み帳票定義、診断一覧

位置と大きさはmmを基本にします。モデルを直接変更するときは、元の定義を変更したくない場合に `Clone()` を使用してください。

14. 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、本番サーバーにも必要なフォントを配置し、利用条件を確認してください。

性能

- 変更されない帳票定義はアプリケーション側でキャッシュできます。
- `IReport` と印刷データは要求ごとに新しく作り、同時要求で共有しないでください。
- 大量ページをPNGへ出力するときは1ページずつ生成し、`Bitmap` を確実に破棄してください。
- Webサーバーでは、本文サイズ、最大ページ数、処理時間、同時生成数を制限してください。
- WPFプレビュアーの描画キャッシュは最大12ページです。

安全性

- 外部から受け取ったPREPDJ、PREPEJ、画像はサイズと内容を検証してください。
- 利用者入力をそのままローカルファイルパスやURLとして読み込ませないでください。
- 出力先をアプリケーション管理下のディレクトリへ限定してください。
- 公開APIには認証、認可、レート制限を適用してください。
- 内部例外をそのままHTTP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てください。

15. エラー処理の基本

```
try
{
    var report = ReportCreator.Create();
    report.LoadDefFile(definitionPath);
    report.PageStart();
    report.Write("顧客名", customerName);
    report.PageEnd();
    report.SavePdf(outputPath, cancellationToken: cancellationToken);
}
catch (OperationCanceledException)
{
    // 利用者操作またはタイムアウトで中止
}
catch (FileNotFoundException)
{
    // 帳票定義または画像が見つからない
}
catch (InvalidDataException)
{
    // PREPDJ／PREPEJの形式または内容が不正
}
catch (ArgumentException)
{
    // オブジェクト名、ページ番号などの指定が不正
}
catch (IOException)
{
    // 読み書きに失敗
}
```

利用者向けメッセージと調査用ログは分けてください。失敗時に、作成途中のファイルを完成ファイルとして残さない設計も重要です。

16. 目的別の早見表

やりたいこと	最初に見る章	主なAPI
PREPDJへ値を入れる	3、4	`ReportCreator`、`LoadDefFile`、`Write`
Windows互換PDFを作る	5.1	`WindowsPdfExporter`
Linuxでも使えるPDFを作る	5.2	`Report.SavePdf`、`SkiaPdfRenderer`
SVG／PNGを作る	6	`GetSvgPages`、`SvgReportRenderer`、`GdiPageRenderer`
完成帳票を保存・再表示する	7	`SaveJson`、`LoadData`、`PrintDataStore`
WPFプレビューを開く	8	`ReportPreviewWindow`

やりたいこと	最初に見る章	主なAPI
プリンターへ出力する	9	`CreatePrintDocument`
Web APIから出力する	10	ASP.NET Coreサンプル
API名を調べる	12	APIリファレンス
デザイナーを操作する	操作ガイド	デザイナー操作ガイド
プレビュアーを操作する	操作ガイド	プレビュアー操作ガイド

17. 付属サンプルを実行する

WPFデスクトップサンプル

```
cd C:\Pao\Pao.Reports.Wpf
dotnet run -c Release -f net10.0-windows10.0.19041.0 `
  --project samples\Pao.Reports.Wpf.Sample\Pao.Reports.Wpf.Sample.csproj
```

11種類の帳票で、GDI+プレビュー、通常PDF、ImagePDF、PREPEJ、印刷を確認できます。全サンプルを非対話で出力する場合は次の形式です。

```
Pao.Reports.Wpf.Sample.exe --export-all C:\temp\reports-wpf-samples
```

Windows Web APIサンプル

```
dotnet run -c Release `
  --project samples\Pao.Reports.Wpf.WebApi\Pao.Reports.Wpf.WebApi.csproj `
  --urls http://127.0.0.1:5288
```

Linux Web APIサンプル

```
dotnet run -c Release `
  --project samples\Pao.Reports.Wpf.WebApi.Linux\Pao.Reports.Wpf.WebApi.Linux.csproj `
  --urls http://127.0.0.1:5289
```

最初は第3章の最小コードを自分のPREPDJとオブジェクト名へ置き換え、次にPREPEJの保存・再読込、必要な出力形式、Web APIの順で追加するのが最短です。

18. ライセンス

本製品は商用ライセンスです。評価、開発用PC、サーバー配置、再配布、組み込み利用の条件は、製品に付属する使用許諾契約を確認してください。