

Barcode.Flutter

Pure Dart版 バーコード生成ライブラリ

マニュアル

バージョン 1.0.0

有限会社 パオ・アット・オフィス

<https://www.pao.ac/>

目次

1. Pure Dart バーコード生成ライブラリ ユーザーズマニュアル
2. 1.1 pao_barcodeとは
3. 1.2 できること
4. 1.3 対応バーコード一覧 (19種)
5. 1.4 3つの出力方式
6. 1.5 他のBarcode製品との違い
7. 1.6 デモサイト
8. 2.1 pubspec.yamlへの追加
9. 2.2 依存パッケージ
10. 2.3 動作確認
11. 3.1 エンコーダーとレンダラー
12. 3.2 BarcodeWidgetで画面表示 (最も簡単)
13. 3.3 PdfRendererでPDF出力
14. 3.4 SVGベクター出力 (レンダラー不要)
15. 4.1 CanvasRendererの基本 — Canvas に何でも描ける
16. 4.2 BarcodeWidget による画面表示 — いちばん簡単な方法
17. 4.3 CustomPainter での描画 — 複数バーコードを自由にレイアウト
18. 4.4 PNG画像への出力 — ファイル保存もBase64もおまかせ
19. 4.5 透明背景のバーコード — 既存デザインに溶け込ませる
20. 4.6 色のカスタマイズ — ブランドカラーで差をつける
21. 5.1 PdfRendererの基本 — pdf パッケージのキャンバスに描く
22. 5.2 実例1: 納品書 — 注文番号バーコード付き帳票
23. 5.3 実例2: 商品ラベル一括印刷 — GridView で並べる
24. 5.4 実例3: コンビニ払込票 — GS1-128 コンビニバーコード
25. 5.5 実例4: 配送伝票 (QR + 1D複合) — 追跡番号 + 住所 + QR
26. 5.6 座標系とサイズ — ミリでもセンチでも思いのまま
27. 5.7 日本語フォント — Noto Sans JP で日本語テキストもきれいに
28. 6.1 1次元バーコード — 物流からPOSまで

29. 6.2 2次元バーコード — 情報をぎっしり詰め込む
30. 6.3 GS1系バーコード — 流通のグローバルスタンダード
31. 6.4 郵便カスタマバーコード — DMハガキの自動仕分け
32. 6.5 Flutter Web連携 — ブラウザでバーコードを表示・ダウンロード
33. 7.1 共通メソッド (全バーコード) — BarcodeBase クラス
34. 7.2 1次元バーコード共通メソッド — BarcodeBase1D クラス
35. 7.3 Code39
36. 7.4 Code93
37. 7.5 Code128
38. 7.6 GS1-128
39. 7.7 NW-7 (Codabar)
40. 7.8 ITF
41. 7.9 Matrix 2of5
42. 7.10 NEC 2of5
43. 7.11 JAN-8 (EAN-8)
44. 7.12 JAN-13 (EAN-13)
45. 7.13 UPC-A
46. 7.14 UPC-E
47. 7.15 GS1 DataBar 標準型 (GS1 DataBar 14)
48. 7.16 GS1 DataBar 限定型 (GS1 DataBar Limited)
49. 7.17 GS1 DataBar 拡張型 (GS1 DataBar Expanded)
50. 7.18 郵便カスタマバーコード
51. 7.19 QRコード
52. 7.20 DataMatrix
53. 7.21 PDF417
54. 7.22 CanvasRenderer
55. 7.23 PdfRenderer
56. 7.24 BarcodeWidget
57. Flutter / Dart バージョン
58. 対応プラットフォーム
59. 依存パッケージ
60. 使用許諾

- 61. ライセンス
- 62. 試用版について
- 63. お問い合わせ
- 64. 関連製品

Pure Dart バーコード生成ライブラリ ユーザーズマニュアル

たった数行で、19種のバーコードを Flutter アプリに。PDF帳票も Canvas 描画も、Pure Dart

バージョン 1.1

2026年2月

有限会社 パオ・アット・オフィス

<https://www.pao.ac/>

1. はじめに — pao_barcode の世界へようこそ

- ・ 1.1 pao_barcodeとは
- ・ 1.2 できること
- ・ 1.3 対応バーコード一覧 (19種)
- ・ 1.4 3つの出力方式
- ・ 1.5 他のBarcode製品との違い
- ・ 1.6 デモサイト

2. インストール — pubspec.yaml に1行追加するだけ

- ・ 2.1 pubspec.yamlへの追加
- ・ 2.2 依存パッケージ
- ・ 2.3 動作確認

3. クイックスタート — 最初の1本を表示しよう

- ・ 3.1 エンコーダーとレンダラー
- ・ 3.2 BarcodeWidgetで画面表示 (最も簡単)
- ・ 3.3 PdfRendererでPDF出力
- ・ 3.4 SVGベクター出力 (レンダラー不要)

4. CanvasRenderer ガイド — 画面表示もPNG出力も

- ・ 4.1 CanvasRendererの基本
- ・ 4.2 BarcodeWidget による画面表示
- ・ 4.3 CustomPainter での描画
- ・ 4.4 PNG画像への出力
- ・ 4.5 透明背景のバーコード
- ・ 4.6 色のカスタマイズ

5. PdfRenderer ガイド — PDF帳票にバーコードを直接描こう

- ・ 5.1 PdfRendererの基本
- ・ 5.2 実例1: 納品書
- ・ 5.3 実例2: 商品ラベル一括印刷
- ・ 5.4 実例3: コンビニ払込票
- ・ 5.5 実例4: 配送伝票 (QR + 1D複合)

- ・ 5.6 座標系とサイズ
- ・ 5.7 日本語フォント

6. サンプルコード集 — 全19種を動かそう

- ・ 6.1 1次元バーコード
- ・ 6.2 2次元バーコード
- ・ 6.3 GS1系バーコード
- ・ 6.4 郵便カスタマバーコード
- ・ 6.5 Flutter Web連携

7. APIリファレンス

- ・ 7.1 共通メソッド (全バーコード)
- ・ 7.2 1次元バーコード共通メソッド
- ・ 7.3 ~ 7.18 各バーコードタイプ
- ・ 7.19 ~ 7.21 2次元バーコード
- ・ 7.22 CanvasRenderer
- ・ 7.23 PdfRenderer
- ・ 7.24 BarcodeWidget

8. 動作環境

9. ライセンス・お問い合わせ

- ・ 9.1 使用許諾
- ・ 9.2 ライセンス

1.1 pao_barcode とは

pao_barcode は、Pure Dart で書かれた Flutter 向けバーコード生成ライブラリです。

C++で開発された高性能バーコードエンジンを Dart に忠実に移植。全19種類のバーコードに対応し、PDF帳票への直接描画、Canvas描画（画面表示・PNG出力）、SVGベクター出力の3つの出力方式をサポートしています。

Pure Dart だから、ネイティブプラグインもFFIも不要。iOS / Android / Web / Windows / macOS / Linux — Flutter が動くすべてのプラットフォームで、そのまま使えます。

```
//  
BarcodeWidget(  
  barcode: Code128(),  
  data: 'Hello-2024',  
  width: 300,  
  height: 100,  
)
```

1.2 できること

pao_barcode を使うと、こんなことができます:

やりたいこと	方法	必要なもの
Flutter アプリにバーコードを表示	BarcodeWidget	追加パッケージ不要
CustomPainter で自由にレイアウト	CanvasRenderer	追加パッケージ不要
PNG画像としてエクスポート	CanvasRenderer.renderToPng()	追加パッケージ不要
PDF帳票にバーコードを直接描画	PdfRenderer	+ pdf パッケージ
SVGベクター画像を生成	エンコーダー内蔵	追加パッケージ不要
バーコードのパターンデータだけ取得	encode() / getPattern()	追加パッケージ不要

> ヒント: BarcodeWidget、CanvasRenderer、SVG出力は追加パッケージ不要。pao_barcode 単体で動きます。pdfパッケージが必要なのはPDF帳票出力のときだけ。

1.3 対応バーコード一覧（19種）

1次元バーコード（11種類）

バーコード	クラス名	どんなところで使われている？
Code39	Code39	工場の部品ラベル、米軍の物資管理
Code93	Code93	郵便局の仕分け、高密度が必要な場面
Code128	Code128	物流ラベル、医療現場、あらゆる業種で
GS1-128	GS1_128	医薬品のトレーサビリティ、コンビニ払込票
NW-7 (Codabar)	NW7	宅配便の送り状、図書館の貸出カード
ITF	ITF	段ボール箱の物流管理（集合包装）
Matrix 2of5	Matrix2of5	航空貨物、工業用途
NEC 2of5	NEC2of5	日本国内の工業用途
JAN-8 (EAN-8)	JAN8	小さなお菓子、リップクリーム
JAN-13 (EAN-13)	JAN13	コンビニ・スーパーの商品コード
UPC-A	UPCA	北米のスーパーマーケットで毎日数十億回

UPCバーコード（1種類）

バーコード	クラス名	どんなところで使われている？
UPC-E	UPCE	小型商品（UPC-Aのゼロ圧縮版）

GS1 DataBar（3種類）

バーコード	クラス名	どんなところで使われている？
GS1 DataBar 標準型	GS1DataBar14	青果・精肉の量り売りラベル
GS1 DataBar 限定型	GS1DataBarLimited	小さな医薬品パッケージ
GS1 DataBar 拡張型	GS1DataBarExpanded	クーポン、有効期限付き商品

2次元バーコード（3種類）

バーコード	クラス名	どんなところで使われている？
QRコード	QR	スマホ決済、Webサイト誘導、名刺
DataMatrix	DataMatrix	電子部品の刻印、手術器具のマーキング
PDF417	PDF417	運転免許証、航空券の搭乗券

特殊バーコード（1種類）

バーコード	クラス名	どんなところで使われている？
郵便カスタマバーコード	YubinCustomer	DMハガキの住所仕分け自動化

> ヒント: 「どのバーコードを使えばいいかわからない」という方へ。迷ったらず Code128 (1D) か QRコード (2D) から始めてみてください。この2つでほとんどのユースケースをカバーできます。

1.4 3つの出力方式

pao_barcode

は「エンコーダー」と「レンダラー」が分離された設計です。エンコーダーがバーコードデータを計算し、レンダラーがそれを可視化します。

出力方式	レンダラー	出力形式	主な用途
Canvas描画	CanvasRenderer / BarcodeWidget	Flutter UI / PNG画像	アプリ画面表示、画像エクスポート
PDF帳票	PdfRenderer	PDF (pdfパッケージのキャンバスに	納品書、請求書、ラベル印刷
SVG	エンコーダー内蔵	SVGベクター画像	高品質印刷、Web表示

> ヒント: Flutter アプリに表示するだけなら BarcodeWidget が最も簡単。PDF帳票を生成するなら PdfRenderer。どちらも Pure Dart なので、プラットフォームを問わず動きます。

1.5 他のBarcode製品との違い

パオ・アット・オフィスのバーコードライブラリには複数の製品があります:

比較項目	Pure Dart版 (本製品)	Pure Python版 (ソースコード付)	C++ Import版
プラットフォーム	Flutter (全6プラットフォーム)	Python	Python
実装	Pure Dart	Pure Python	C++ (WASM/FFI)
プラグイン依存	なし	なし	あり
PDF帳票描画	PdfRenderer	ReportLabRenderer	--
画面表示	BarcodeWidget / CanvasRenderer	PillowRenderer	--
SVG出力	エンコーダー内蔵	エンコーダー内蔵	あり
ソースコード	なし	付属	なし
バーコード種類	19種	19種	18種
価格 (税込)	22,000円	22,000円	11,000円

Pure Dart版を選ぶ理由:

- Flutter アプリにバーコードを表示したい → BarcodeWidget で数行
- iOS/Android/Web/Desktop すべてで動かしたい → ネイティブプラグイン不要
- PDF帳票にバーコードを直接描画したい → PdfRenderer で pdf パッケージに統合
- ネイティブ依存なしで全プラットフォーム対応 → プラグイン不要で安定動作

> ヒント: Flutter アプリでバーコードを使いたいなら、Pure Dart版 (pao_barcode) 一択です。ネイティブプラグインに依存しないので、ビルドの問題も少なく、全プラットフォームで安定動作します。

1.6 デモサイト

実際の動作をブラウザで確認できるデモサイトを公開しています:

- ・ Flutter Pure Dart版デモ (All-in-One) : <https://www.pao.ac/demo/barcode-flutter/>
全19種のバーコード生成、PDF帳票出力、Canvas描画をお試しいただけます。

2.1 pubspec.yamlへの追加

```
dependencies:  
  pao_barcode:  
    path: /path/to/pao_barcode #  
    # git URL / pub.dev
```

その後:

```
flutter pub get
```

Pure Dart なので、ネイティブビルドの設定は一切不要です。

2.2 依存パッケージ

用途に応じて追加してください:

pdf (PDF帳票出力に必要)

```
dependencies:  
  pdf: ^3.10.0
```

PDF帳票のキャンバスにバーコードを直接描画する PdfRenderer を使うために必要です。

printing (PDF印刷・プレビューに必要)

```
dependencies:  
  printing: ^5.11.0
```

全部入り構成

```
dependencies:  
  flutter:  
    sdk: flutter  
  pao_barcode:  
    path: ./pao_barcode  
  pdf: ^3.10.0  
  printing: ^5.11.0
```

> ヒント: BarcodeWidget や CanvasRenderer だけなら pao_barcode 単体で動きます。pdf パッケージは PDF帳票を生成するときだけ追加すれば大丈夫。

2.3 動作確認

```
import 'package:pao_barcode/pao_barcode.dart';

void main() {
  //
  final bc = Code128();
  final pattern = bc.encode('TEST1234');
  print('Code128 : ${pattern.length} ');
  // → Code128 : 33

  // QR
  final qr = QR();
  final matrix = qr.getPattern('Hello World');
  print('QR: ${matrix.length}x${matrix[0].length} ');
  // → QR: 25x25
}
```

3.1 エンコーダーとレンダラー

`pao_barcode` は エンコーダー（バーコードデータの計算）とレンダラー（描画）を分離した設計です。エンコーダーが「バーの並び」を計算し、レンダラーがそれを「画面やPDFに見える形」にします。

```
import 'package:pao_barcode/pao_barcode.dart';

// --- 1D ---
// encode() →
final bc = Code128();
final pattern = bc.encode('Hello-2024');
// pattern = [2, 1, 1, 2, 3, 2, ...] ← bar, space, bar, ...

// --- 2D ---
// getPattern() → 2D
final qr = QR();
final matrix = qr.getPattern('https://www.pao.ac/');
// matrix[row][col] = true() or false()

// --- JAN/UPC ---
final jan = JAN13();
final janPattern = jan.encode('4901234567894');
```

エンコーダーの出力は数値データだけ。描画には `CanvasRenderer`（画面・PNG用）か `PdfRenderer`（PDF用）を使います。

> ヒント: エンコーダーの出力を独自のレンダラーで描画することも可能です。たとえば `encode()` の結果を使って、自前のCanvasロジックに組み込みます。

3.2 BarcodeWidgetで画面表示（最も簡単）

Flutter アプリにバーコードを表示する最も簡単な方法です:

```
import 'package:flutter/material.dart';
import 'package:pao_barcode/pao_barcode.dart';

class MyBarcodeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: BarcodeWidget(
          barcode: Code128(),
          data: 'Hello-2024',
          width: 300,
          height: 100,
        ),
      ),
    );
  }
}
```

> ヒント: BarcodeWidget は CustomPainter を内部で使っており、効率的に描画されます。サイズを null にすると親 Widget のサイズに合わせて自動的にフィットします。

PNG画像として出力

```
import 'package:pao_barcode/pao_barcode.dart';

Future<Uint8List> generateBarcodePng() async {
  final renderer = CanvasRenderer();
  final bc = Code128();

  // PNG bytes
  final pngBytes = await renderer.renderToPng(bc, 'Hello-2024', 400, 100);
  return pngBytes;
}
```

3.3 PdfRendererでPDF出力

```
import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:pao_barcode/pao_barcode.dart';

pw.Document createPdfWithBarcode() {
  final doc = pw.Document();

  doc.addPage(
    pw.Page(
      build: (context) {
        return pw.Column(
          children: [
            pw.Text(''),
            pw.SizedBox(height: 10),
            pw.CustomPaint(
              size: const PdfPoint(200, 50),
              painter: (canvas, size) {
                final renderer = PdfRenderer(canvas);
                final bc = Code128();
                renderer.draw(bc, '1234567890', 0, 0, size.x, size.y);
              },
            ),
          ],
        );
      },
    ),
  );

  return doc;
}
```

> ヒント: PdfRenderer は pdf パッケージのキャンバスにベクターデータとして直接描画します。画像を貼り付けるのとは違い、拡大してもくっきり、ファイルサイズも小さい。

3.4 SVGベクター出力（レンダラー不要）

SVG出力はエンコーダー単体で行えます。レンダラーは不要です。

```
import 'package:pao_barcode/pao_barcode.dart';

String generateSvg() {
  final bc = Code128();
  bc.setOutputFormat('svg'); // SVG
  bc.setShowText(true);

  // → SVG
  bc.draw('Hello-2024', 0, 0, 400, 100);
  return bc.getSvg();
}
```

> ヒント: SVG出力は追加パッケージ不要。flutter_svg と組み合わせてアプリ内に表示したり、ファイルとしてエクスポートしたりできます。

Flutter アプリにバーコードを組み込む道は2つ。お手軽な BarcodeWidget か、自由度の高い CanvasRenderer + CustomPainter か。どちらも Pure Dart、追加パッケージ不要です。

4.1 CanvasRendererの基本 — Canvas に何でも描ける

```
import 'package:pao_barcode/pao_barcode.dart';

// CanvasRenderer
final renderer = CanvasRenderer(
  foreColor: Colors.black,    // :
  backColor: Colors.white,    // :
  showText: true,            // : true
  fontSize: 14,              // : 14
  quietZone: 0,             // : 0
);

// Canvas
renderer.draw(canvas, barcode, data, x, y, width, height);
```

CanvasRenderer コンストラクタパラメータ:

パラメータ	型	説明	デフォルト
foreColor	Color	前景色 (バーの色)	Colors.black
backColor	Color	背景色	Colors.white
showText	bool	1Dバーコード下のテキスト表示	true
fontSize	double	テキストのフォントサイズ	14
quietZone	double	クワイエットゾーン比率 (0.0~0.5)	0

Canvas 座標系について:

Flutter の Canvas は 原点が左上、Y軸が下向き の画像座標系です。(x, y) はバーコード描画領域の 左上隅を指定します。

```
(0,0) -----> X
|
|  ┌───────────┐
|  │             │
|  │             │
|  │             │
|  └───────────┘
v
Y
```

> ヒント: CanvasRenderer は draw() 1回で背景塗りつぶし + バーコード描画 + テキスト描画をすべて行います。Canvas の save()/restore() は内部では使わないので、呼び出し側で自由にクリッピングや変換を適用できます。

4.2 BarcodeWidget による画面表示 — いちばん簡単な方法

BarcodeWidget は CanvasRenderer を内部に包んだ便利な Widget です。数行で Flutter アプリにバーコードを表示できます。

商品ラベルの例

```
import 'package:flutter/material.dart';
import 'package:pao_barcode/pao_barcode.dart';

class ProductLabel extends StatelessWidget {
  final String productName;
  final String janCode;

  const ProductLabel({
    super.key,
    required this.productName,
    required this.janCode,
  });

  @override
  Widget build(BuildContext context) {
    return Card(
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            Text(productName, style: const TextStyle(fontSize: 18)),
            const SizedBox(height: 12),
            BarcodeWidget(
              barcode: JAN13(),
              data: janCode,
              width: 250,
              height: 80,
            ),
          ],
        ),
      ),
    );
  }
}

// :
// ProductLabel(productName: '', janCode: '4901234567894')
```

BarcodeWidget プロパティ一覧

プロパティ	型	説明	デフォルト
barcode	dynamic	エンコーダーインスタンス	(必須)
data	String	バーコードデータ	(必須)
width	double?	幅。nullで親Widgetに合わせる	null
height	double?	高さ。nullで親Widgetに合わせる	null
foreColor	Color	前景色	Colors.black
backColor	Color	背景色	Colors.white
showText	bool	テキスト表示	true
fontSize	double	フォントサイズ	14
quietZone	double	クワイエットゾーン比率	0

> ヒント: width と height を null にすると、BarcodeWidget は LayoutBuilder を使って親 Widget のサイズに自動フィットします。Expanded や Flexible の中で使うときに便利です。

4.3 CustomPainter での描画 — 複数バーコードを自由にレイアウト

1つのキャンバスに複数のバーコードを配置したい場合は、CustomPainter を使います。

```
import 'package:flutter/material.dart';
import 'package:pao_barcode/pao_barcode.dart';

class MultiBarcodePainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    final renderer = CanvasRenderer(fontSize: 12);

    // : Code128
    final code128 = Code128();
    renderer.draw(canvas, code128, 'ORDER-2024-0042',
      10, 10, size.width - 20, 60);

    // : QR
    final qr = QR();
    renderer.draw(canvas, qr, 'https://www.pao.ac/',
      10, 80, 100, 100);

    // : JAN-13
    final jan13 = JAN13();
    renderer.draw(canvas, jan13, '4901234567894',
      120, 100, 180, 60);
  }

  @override
  bool shouldRepaint(covariant CustomPainter oldDelegate) => false;
}

// Widget
class MultiBarcodePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('')),
      body: Center(
        child: SizedBox(
          width: 320,
          height: 200,
          child: CustomPaint(painter: MultiBarcodePainter()),
        ),
      ),
    );
  }
}
```

> ヒント: `shouldRepaint()` で `false` を返すと、再描画を抑制してパフォーマンスを最適化できます。データが変わる場合は、旧デリゲートと比較して変更がある場合だけ `true` を返しましょう。

4.4 PNG画像への出力 — ファイル保存もBase64もおまかせ

renderToPng で PNG バイト列を生成

```
import 'package:pao_barcode/pao_barcode.dart';

Future<Uint8List> generateBarcodePng() async {
  final renderer = CanvasRenderer(showText: true, fontSize: 14);
  final bc = Code128();

  // PNG bytes
  final pngBytes = await renderer.renderToPng(bc, 'Hello-2024', 400, 100);
  return pngBytes;
}
```

ファイルに保存

```
import 'dart:io';
import 'package:pao_barcode/pao_barcode.dart';

Future<void> saveBarcodeToFile() async {
  final renderer = CanvasRenderer();
  final bc = JAN13();

  final pngBytes = await renderer.renderToPng(
    bc, '4901234567894', 400, 120,
  );

  final file = File('barcode_jan13.png');
  await file.writeAsBytes(pngBytes);
  print(': ${file.path}');
}
```

Base64 データ URI (Flutter Web で便利)

```
import 'dart:convert';
import 'package:pao_barcode/pao_barcode.dart';

Future<String> generateBase64DataUri() async {
  final renderer = CanvasRenderer();
  final qr = QR();

  final pngBytes = await renderer.renderToPng(
    qr, 'https://www.pao.ac/', 300, 300,
  );

  // Base64 URI
  final base64String = base64Encode(pngBytes);
  return 'data:image/png;base64,$base64String';
}
```

> ヒント: `renderToPng()` は内部で `PictureRecorder` → `Canvas` → `Picture` → `Image` → `PNG bytes` という変換を行います。非同期処理なので `await` を忘れずに。Flutter Web でも動作します。

4.5 透明背景のバーコード — 既存デザインに溶け込ませる

背景を透明にすると、バーだけが描画されて既存のUIデザインに自然に溶け込みます。

```
import 'package:flutter/material.dart';
import 'package:pao_barcode/pao_barcode.dart';

// BarcodeWidget
BarcodeWidget(
  barcode: Code128(),
  data: 'TRANSPARENT',
  width: 300,
  height: 80,
  backgroundColor: Colors.transparent, //
)

// CanvasRenderer
final renderer = CanvasRenderer(
  backgroundColor: Colors.transparent,
  foreColor: Colors.black87,
);
```

> ヒント: 透明背景は Container の decoration で背景画像やグラデーションを設定した上にバーコードを重ねるときに活躍します。ただし、バーコードの読み取り精度を保つため、バーと背景のコントラストは十分に確保してください。

4.6 色のカスタマイズ — ブランドカラーで差をつける

```
import 'package:flutter/material.dart';
import 'package:pao_barcode/pao_barcode.dart';

//
BarcodeWidget(
  barcode: QR(),
  data: 'https://example.com/',
  width: 200,
  height: 200,
  foreColor: const Color(0xFF1A237E), //
  backColor: const Color(0xFFFFF8E1), //
)
```

実用的なカラーパターン

```
// 1: ×
BarcodeWidget(
  barcode: Code128(),
  data: 'INVOICE-001',
  width: 280, height: 70,
  foreColor: const Color(0xFF0D47A1),
  backColor: Colors.white,
)

// 2: ×
BarcodeWidget(
  barcode: JAN13(),
  data: '4901234567894',
  width: 250, height: 80,
  foreColor: const Color(0xFF1B5E20),
  backColor: const Color(0xFFE8F5E9),
)

// 3: ×
BarcodeWidget(
  barcode: QR(),
  data: 'https://www.pao.ac/',
  width: 150, height: 150,
  foreColor: Colors.white,
  backColor: const Color(0xFF212121),
)
```

>

ヒント:

QRコードは色のコントラストに対して比較的寛容ですが、1Dバーコード（特にJAN/UPC）はスキャナーの読み取り精

度に直結します。業務用途では黒×白を推奨します。カラーバーコードはWebやアプリ表示専用と考えてください。

pao_barcode の強力な機能、それは pdf
パッケージで作成するPDF帳票にバーコードをベクターデータとして直接描画できること。画像を貼り付けるのとは違い、拡大しても鮮明、ファイルサイズも最小限です。

5.1 PdfRendererの基本 — pdf パッケージのキャンバスに描く

```
import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:pao_barcode/pao_barcode.dart';

// PdfRenderer pdf PdfGraphics
// pw.CustomPaint
```

PdfRenderer コンストラクタ:

```
PdfRenderer(
  canvas, // PdfGraphics
  foreColor: PdfColors.black, // :
  backColor: PdfColors.white, // :
  showText: true, // : true
  fontName: 'Helvetica', // : 'Helvetica'
  fontSize: 8, // (pt): 8
  quietZone: 0, // : 0
);
```

パラメータ	型	説明	デフォルト
canvas	PdfGraphics	pdfパッケージのキャンバス	(必須)
foreColor	PdfColor	前景色 (バーの色)	PdfColors.black
backColor	PdfColor	背景色	PdfColors.white
showText	bool	テキスト表示	true
fontName	String	フォント名	'Helvetica'
fontSize	double	フォントサイズ (ポイント)	8
quietZone	double	クワイエットゾーン比率	0

PDF 座標系について (Canvas と逆です!) :

pdf パッケージは 原点が左下、Y軸が上向き のPDF標準座標系です。(x, y) はバーコード描画領域の 左下隅を指定します。単位は ポイント (1ポイント = 1/72インチ)。

```
Y
^
|
| [ ]
|
|
|
(0,0) -----> X
```

> ヒント: CanvasRenderer は「原点が左上、Y下向き」、PdfRenderer は「原点が左下、Y上向き」。逆なので、Canvas用のコードをそのままPDFに持ち込むと上下反転します。PdfRender

erはこの違いを内部で吸収しているので、描画結果は正しく出ます。

5.2 実例1: 納品書 — 注文番号バーコード付き帳票

```

import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:pao_barcode/pao_barcode.dart';

pw.Document createDeliveryNote() {
  final doc = pw.Document();

  doc.addPage(
    pw.Page(
      pageFormat: PdfPageFormat.a4,
      build: (context) {
        return pw.Column(
          crossAxisAlignment: pw.CrossAxisAlignment.start,
          children: [
            //
            pw.Text('',
              style: pw.TextStyle(fontSize: 24, fontWeight: pw.FontWeight.bold)),
            pw.SizedBox(height: 10),
            pw.Text('2024121'),
            pw.SizedBox(height: 20),

            //
            pw.Text(': ORD-2024-0042', style: const pw.TextStyle(fontSize: 12)),
            pw.SizedBox(height: 5),
            pw.CustomPaint(
              size: const PdfPoint(250, 50),
              painter: (PdfGraphics canvas, PdfPoint size) {
                final renderer = PdfRenderer(canvas);
                final bc = Code128();
                renderer.draw(bc, 'ORD-2024-0042', 0, 0, size.x, size.y);
              },
            ),
            pw.SizedBox(height: 20),

            //
            pw.Table.fromTextArray(
              headers: ['', '', '', ''],
              data: [
                ['Flutter', '2', '3,200', '6,400'],
                ['Dart', '1', '4,500', '4,500'],
                ['USB', '5', '800', '4,000'],
              ],
            ),
            pw.SizedBox(height: 10),
            pw.Align(
              alignment: pw.Alignment.centerRight,
              child: pw.Text(': 14,900',
                style: pw.TextStyle(fontSize: 16, fontWeight: pw.FontWeight.bold)),
            ),
          ],
        );
      },
    );
}

```

```
    },  
  ),  
);  
  
  return doc;  
}
```

> ヒント: `pw.CustomPaint` の `painter` コールバックは (`PdfGraphics`, `PdfPoint`) を受け取ります。この `PdfGraphics` をそのまま `PdfRenderer` のコンストラクタに渡すだけ。Widget ツリーの中にバーコードを自然に埋め込みます。

5.3 実例2: 商品ラベル一括印刷 — GridView で並べる

```

import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:pao_barcode/pao_barcode.dart';

pw.Document createProductLabels(List<Map<String, String>> products) {
  final doc = pw.Document();

  // Widget
  final labels = products.map((p) {
    return pw.Container(
      padding: const pw.EdgeInsets.all(8),
      decoration: pw.BoxDecoration(
        border: pw.Border.all(color: PdfColors.grey300),
      ),
      child: pw.Column(
        mainAxisAlignment: pw.MainAxisSize.min,
        children: [
          pw.Text(p['name']!, style: pw.TextStyle(fontSize: 10,
            fontWeight: pw.FontWeight.bold)),
          pw.SizedBox(height: 4),
          pw.Text('${p['price']}', style: const pw.TextStyle(fontSize: 8)),
          pw.SizedBox(height: 6),
          pw.CustomPaint(
            size: const PdfPoint(120, 40),
            painter: (PdfGraphics canvas, PdfPoint size) {
              final renderer = PdfRenderer(canvas, fontSize: 6);
              final bc = JAN13();
              renderer.draw(bc, p['jan']!, 0, 0, size.x, size.y);
            },
          ),
        ],
      ),
    );
  }).toList();

  doc.addPage(
    pw.Page(
      pageFormat: PdfPageFormat.a4,
      build: (context) {
        return pw.GridView(
          crossAxisCount: 3,
          childAspectRatio: 1.5,
          children: labels,
        );
      },
    ),
  );

  return doc;
}

```

```
// :  
// final products = [  
//   {'name': '', 'price': '150', 'jan': '4901234567894'},  
//   {'name': '', 'price': '200', 'jan': '4902345678901'},  
//   {'name': '', 'price': '100', 'jan': '4903456789012'},  
//   // ...  
// ];  
// final doc = createProductLabels(products);
```

> ヒント: 1ページに収まりきらない場合は `pw.MultiPage` を使えば自動で改ページされます。`pw.GridView` と組み合わせれば、何十枚ものラベルを一度に生成できます。

5.4 実例3: コンビニ払込票 — GS1-128 コンビニバーコード

コンビニ収納代行バーコード（払込票）は `GS1-128` の特殊形式です。`drawGs1128Convenience` メソッドで2行テキスト付きのバーコードを描画できます。

```

import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:pao_barcode/pao_barcode.dart';

pw.Document createPaymentSlip() {
  final doc = pw.Document();

  // {FNC1} + 44
  final convenienceData =
    '{FNC1}91234567890123456789012345678901234567890123';

  doc.addPage(
    pw.Page(
      pageFormat: PdfPageFormat.a4,
      build: (context) {
        return pw.Column(
          crossAxisAlignment: pw.CrossAxisAlignment.start,
          children: [
            pw.Text('',
              style: pw.TextStyle(fontSize: 20, fontWeight: pw.FontWeight.bold)),
            pw.SizedBox(height: 20),
            pw.Row(
              mainAxisAlignment: pw.MainAxisAlignment.spaceBetween,
              children: [
                pw.Text(': 12,800'),
                pw.Text(': 2025331'),
              ],
            ),
            pw.SizedBox(height: 30),

            // GS1-128 2
            pw.CustomPaint(
              size: const PdfPoint(350, 80),
              painter: (PdfGraphics canvas, PdfPoint size) {
                final renderer = PdfRenderer(canvas, fontSize: 7);
                final gs1 = GS1_128();
                renderer.drawGs1128Convenience(
                  gs1, convenienceData, 0, 0, size.x, size.y);
              },
            ),
          ],
        );
      },
    ),
  );

  return doc;
}

```

> ヒント: コンビニバーコードの44桁は先頭に {FNC1} を付けて渡します。drawGs1128Convenience()

は自動的に2行テキスト（上段: AI+企業コード、下段: 金額+チェックディジット）を描画します。通常の `draw()` ではなく専用メソッドを使うのがポイントです。

5.5 実例4: 配送伝票 (QR + 1D複合) — 追跡番号 + 住所 + QR

```

import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:pao_barcode/pao_barcode.dart';

pw.Document createShippingLabel() {
  final doc = pw.Document();

  doc.addPage(
    pw.Page(
      pageFormat: PdfPageFormat(
        10 * PdfPageFormat.cm, 15 * PdfPageFormat.cm,
      ),
      build: (context) {
        return pw.Column(
          crossAxisAlignment: pw.CrossAxisAlignment.start,
          children: [
            // (Code128)
            pw.Text('', style: const pw.TextStyle(fontSize: 8)),
            pw.CustomPaint(
              size: const PdfPoint(250, 40),
              painter: (PdfGraphics canvas, PdfPoint size) {
                final renderer = PdfRenderer(canvas, fontSize: 7);
                final bc = Code128();
                renderer.draw(bc, '4912345678901', 0, 0, size.x, size.y);
              },
            ),
            pw.SizedBox(height: 10),

            //
            pw.Text('', style: const pw.TextStyle(fontSize: 8)),
            pw.CustomPaint(
              size: const PdfPoint(200, 20),
              painter: (PdfGraphics canvas, PdfPoint size) {
                final renderer = PdfRenderer(canvas);
                final yb = YubinCustomer();
                renderer.draw(yb, '1050011', 0, 0, size.x, size.y);
              },
            ),
            pw.SizedBox(height: 15),

            //
            pw.Text(':',
              style: pw.TextStyle(fontSize: 10, fontWeight: pw.FontWeight.bold)),
            pw.Text('105-0011 4-2-8'),
            pw.Text(' ', style: const pw.TextStyle(fontSize: 14)),
            pw.SizedBox(height: 15),

            // QRURL
            pw.Row(
              children: [
                pw.CustomPaint(

```

```
        size: const PdfPoint(80, 80),
        painter: (PdfGraphics canvas, PdfPoint size) {
          final renderer = PdfRenderer(canvas);
          final qr = QR();
          renderer.draw(qr,
            'https://track.example.com/4912345678901',
            0, 0, size.x, size.y);
        },
      ),
    pw.SizedBox(width: 10),
    pw.Text('QR\n',
      style: const pw.TextStyle(fontSize: 8)),
  ],
),
],
);
},
),
);

return doc;
}
```

> ヒント: 1つのPDFページに異なる種類のバーコードを何個でも描画できます。Code128（追跡番号） + 郵便カスタマバーコード（住所） + QR（追跡URL）の組み合わせは、配送伝票の定番パターンです。

5.6 座標系とサイズ — ミリでもセンチでも思いのまま

pdf のユーティリティでミリやセンチに変換できます。

パッケージはポイント (pt) が基本単位ですが、PdfPageFormat

```
//
final mm = PdfPageFormat.mm; // 1mm = 2.8346pt
final cm = PdfPageFormat.cm; // 1cm = 28.346pt
final inch = PdfPageFormat.inch; // 1inch = 72pt

// : 50mm x 15mm
renderer.draw(bc, data, 10 * mm, 200 * mm, 50 * mm, 15 * mm);
```

標準用紙サイズ:

用紙	PdfPageFormat	サイズ (pt)	サイズ (mm)
A4	PdfPageFormat.a4	595 x 842	210 x 297
A3	PdfPageFormat.a3	842 x 1191	297 x 420
B5	PdfPageFormat.a5	420 x 595	148 x 210
Letter	PdfPageFormat.letter	612 x 792	216 x 279
Legal	PdfPageFormat.legal	612 x 1008	216 x 356

```
//
final labelFormat = PdfPageFormat(
  62 * PdfPageFormat.mm, // 62mm
  29 * PdfPageFormat.mm, // 29mm
);
```

> ヒント: 実際の印刷では余白 (マージン) も考慮が必要です。PdfPageFormat.a4.copyWith(marginAll: 1.5 * cm) のように余白を指定すると、context.page.pageFormat.availableWidth で印刷可能領域が取得できます。

5.7 日本語フォント — Noto Sans JP で日本語テキストもきれいに

pdf

パッケージのデフォルトフォント（Helvetica）は日本語を含みません。帳票に日本語テキストを表示するには、日本語フォントを読み込みます。

```

import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:printing/printing.dart';

Future<pw.Document> createJapaneseInvoice() async {
  final doc = pw.Document();

  // Google Fonts Noto Sans JP
  final jaFont = await PdfGoogleFonts.notoSansJPRegular();
  final jaBoldFont = await PdfGoogleFonts.notoSansJPBold();

  doc.addPage(
    pw.Page(
      pageFormat: PdfPageFormat.a4,
      theme: pw.ThemeData.withFont(
        base: jaFont,
        bold: jaBoldFont,
      ),
      build: (context) {
        return pw.Column(
          crossAxisAlignment: pw.CrossAxisAlignment.start,
          children: [
            pw.Text('',
              style: pw.TextStyle(fontSize: 24,
                fontWeight: pw.FontWeight.bold)),
            pw.Text(' '),
            pw.SizedBox(height: 20),
            pw.CustomPaint(
              size: const PdfPoint(200, 50),
              painter: (PdfGraphics canvas, PdfPoint size) {
                final renderer = PdfRenderer(canvas);
                final bc = Code128();
                renderer.draw(bc, 'INV-2024-0001', 0, 0, size.x, size.y);
              },
            ),
          ],
        );
      },
    ),
  );

  return doc;
}

```

> ヒント: PdfGoogleFonts を使うには printing パッケージが必要です。pubspec.yaml に printing: ^5.11.0 を追加してください。フォントファイルは初回にダウンロードされ、以降はキャッシュされます。

ここでは全19種類のバーコードの生成コードを一挙に紹介します。すべて Dart/Flutter のコードです。CanvasRenderer での画面表示・PNG出力を基本としつつ、エンコーダー単体での使い方も示します。

6.1 1次元バーコード — 物流からPOSまで

Code128 — あらゆる業種で活躍する万能選手

```
import 'package:pao_barcode/pao_barcode.dart';

// : AUTO
// CODE-A/B/C
final bc = Code128();
final pattern = bc.encode('Hello-2024');
print('Code128 AUTO: ${pattern.length} ');

// CODE-B ASCII
final bcB = Code128();
bcB.codeMode = CodeSet128.codeB;
final patternB = bcB.encode('Hello-2024');

// CODE-C
final bcC = Code128();
bcC.codeMode = CodeSet128.codeC;
final patternC = bcC.encode('12345678'); //
```

CodeSet128 列挙型:

値	説明	対応文字
CodeSet128.auto_	自動最適化 (デフォルト)	全文字
CodeSet128.codeA	CODE-A固定	ASCII制御文字 + 英数大文字
CodeSet128.codeB	CODE-B固定	ASCII印字可能文字 (小文字含む)
CodeSet128.codeC	CODE-C固定	数字ペア (00-99)

BarcodeWidget で表示:

```
BarcodeWidget(
  barcode: Code128(),
  data: 'Hello-2024',
  width: 300,
  height: 80,
)
```

> ヒント: 数字だけのデータなら `CODE-C` が最もコンパクト (2桁を1シンボルにエンコード)。AUTO モードなら自動的に最短幅を選んでくれるので、通常は AUTO のままで OK です。

Code39 — 工場現場の定番

```
final bc = Code39();
bc.showStartStop = true; // /*

BarcodeWidget(
  barcode: bc,
  data: 'HELLO-123',
  width: 350, height: 80,
)
```

入力可能文字: 数字 (0-9)、英大文字 (A-Z)、記号 (-. \$ / + % スペース)

Code93 — Code39 の高密度版

```
BarcodeWidget(
  barcode: Code93(),
  data: 'Hello123',
  width: 300, height: 80,
)
```

入力可能文字: ASCII 全文字 (内部でシフト文字を使って拡張)

NW-7 (Codabar) — 宅配便の送り状でおなじみ

```
final bc = NW7();
bc.showStartStop = true; // /

BarcodeWidget(
  barcode: bc,
  data: 'A1234567890B', // A-D A-D
  width: 350, height: 80,
)
```

入力可能文字: 数字 (0-9)、記号 (-. \$ / + :)、スタート/ストップ (A, B, C, D)

> ヒント: NW-7 のデータは必ず A~D のいずれかで開始・終了します。スタート/ストップ文字がない場合は自動的に A が付与されます。

ITF — 段ボール箱の物流管理

```
BarcodeWidget(
  barcode: ITF(),
  data: '1234567890', //
  width: 300, height: 80,
)
```

入力可能文字: 数字のみ (偶数桁)

Matrix 2of5 — 航空貨物・工業用途

```
BarcodeWidget(  
  barcode: Matrix2of5(),  
  data: '1234567890',  
  width: 300, height: 80,  
)
```

NEC 2of5 — 日本国内の工業用途

```
BarcodeWidget(  
  barcode: NEC2of5(),  
  data: '1234567890',  
  width: 300, height: 80,  
)
```

JAN-13 (EAN-13) — 日本の商品コード

```
// 12  
BarcodeWidget(  
  barcode: JAN13(),  
  data: '490123456789', // 12 → 13  
  width: 250, height: 80,  
)  
  
// 130K  
BarcodeWidget(  
  barcode: JAN13(),  
  data: '4901234567894', // 13  
  width: 250, height: 80,  
)
```

JAN-8 (EAN-8) — 小さな商品に

```
BarcodeWidget(  
  barcode: JAN8(),  
  data: '4912345', // 7 → 8  
  width: 200, height: 80,  
)
```

UPC-A — 北米の商品コード

```
BarcodeWidget(  
  barcode: UPCA(),  
  data: '01234567890', // 11 → 12  
  width: 250, height: 80,  
)
```

UPC-E — 小型商品（UPC-A のゼロ圧縮版）

```
BarcodeWidget(  
  barcode: UPCE(),  
  data: '0123456', // 7 → 8  
  width: 180, height: 80,  
)
```

> ヒント: JAN/UPC
バーコードはチェックディジットなしで渡すのが一般的です。ライブラリが自動計算してくれます。もちろん、チェックディジット込みの完全な桁数で渡しても正しく動作します。

6.2 2次元バーコード — 情報をぎっしり詰め込む

QRコード — スマホ時代の主役

```
import 'package:pao_barcode/pao_barcode.dart';

//
final qr = QR();
final matrix = qr.getPattern('https://www.pao.ac/');
print('QR: ${matrix.length}x${matrix[0].length}');

//
final qrH = QR();
qrH.setErrorCorrectionLevel(QRErrorLevel.h); // 30%
```

QRErrorLevel 列挙型:

レベル	復元率	用途
QRErrorLevel.l	約7%	データ量優先 (名刺等)
QRErrorLevel.m	約15%	標準 (デフォルト)
QRErrorLevel.q	約25%	やや汚れる環境
QRErrorLevel.h	約30%	工場・屋外など過酷な環境

バージョン指定:

```
final qr = QR();
qr.setVersion(5); // Version 537x37
// 0 = 1-40
```

QRコード バージョン・サイズ・容量:

バージョン	モジュール数	数字 (L)	英数字 (L)	バイト (L)
1	21 x 21	41	25	17
5	37 x 37	154	93	64
10	57 x 57	652	395	271
20	97 x 97	1,817	1,101	756
40	177 x 177	7,089	4,296	2,953

BarcodeWidget で表示:

```
BarcodeWidget(
  barcode: QR()..setErrorCorrectionLevel(QRErrorLevel.h),
  data: 'https://www.pao.ac/',
  width: 200,
  height: 200,
)
```

> ヒント: QRコードは正方形で描画するのが基本。BarcodeWidget の width と height を同じ値にしてください。誤り訂正レベル H にすると、QRコードの中央にロゴを重ねても（30%まで隠れても）読み取り可能です。

DataMatrix — 極小部品のマーキングに

```
final dm = DataMatrix();
final matrix = dm.getPattern('Hello DataMatrix');
print('DataMatrix: ${matrix.length}x${matrix[0].length}');

//
final dmFixed = DataMatrix();
dmFixed.setCodeSize(DxCodeSize.dxCs20x20);
```

DxCodeSize 列挙型（主要なもの）:

値	サイズ	用途
DxCodeSize.dxCsAuto	自動（デフォルト）	データに応じた最適サイズ
DxCodeSize.dxCsShapeAuto	正方形自動	正方形のみから選択
DxCodeSize.dxCsRectAuto	長方形自動	長方形のみから選択
DxCodeSize.dxCs10x10	10x10	最小サイズ
DxCodeSize.dxCs20x20	20x20	小型
DxCodeSize.dxCs32x32	32x32	標準
DxCodeSize.dxCs144x144	144x144	最大サイズ
DxCodeSize.dxCs8x18	8x18	長方形
DxCodeSize.dxCs16x48	16x48	長方形（長い）

```
BarcodeWidget(
  barcode: DataMatrix(),
  data: 'PAO-BARCODE-2024',
  width: 150,
  height: 150,
)
```

> ヒント: DataMatrix は小さなスペースに多くのデータを詰め込めるため、電子部品やICチップのマーキングで世界的に使われています。GS1 モードを使う場合はデータの先頭に {FNC1} を付けます。

PDF417 — 運転免許証や航空券に

```
final pdf417 = PDF417();
pdf417.setErrorLevel(PDF417ErrorLevel.level5);
pdf417.setSizeMode(PDF417SizeMode.columns);
pdf417.setColumns(4);

final matrix = pdf417.getPattern('Hello PDF417');
print('PDF417: ${matrix.length} x ${matrix[0].length}');
```

PDF417ErrorLevel 列挙型:

値	訂正コードワード数	用途
PDF417ErrorLevel.auto_	自動 (デフォルト)	データ量に応じて自動選択
PDF417ErrorLevel.level0	2	最小
PDF417ErrorLevel.level2	8	標準
PDF417ErrorLevel.level5	64	高い信頼性
PDF417ErrorLevel.level8	512	最高レベル

PDF417SizeMode 列挙型:

値	説明
PDF417SizeMode.auto_	自動 (デフォルト)
PDF417SizeMode.columns	列数を指定
PDF417SizeMode.rows	行数を指定
PDF417SizeMode.columnsAndRows	列数と行数を同時指定

```
BarcodeWidget(
  barcode: PDF417()..setErrorLevel(PDF417ErrorLevel.level3),
  data: 'Flight: NH123 Seat: 15A Gate: 42',
  width: 300,
  height: 100,
)
```

> ヒント: PDF417 はバーコードの名前に "PDF" がつきますが、ファイル形式のPDFとは無関係。"Portable Data File" の略です。横に長いバーコードなので、width:height の比率は 3:1 程度がきれいに見えます。

6.3 GS1系バーコード — 流通のグローバルスタンダード

GS1-128 — 医薬品トレーサビリティからコンビニ払込票まで

```
import 'package:pao_barcode/pao_barcode.dart';

// : AI ()
final gs1 = GS1_128();
final pattern = gs1.encode('{FNC1}0104912345678904');
// {FNC1} GS1-128
// 01 = AI (GTIN)04912345678904 = GTIN-14

BarcodeWidget(
  barcode: GS1_128(),
  data: '{FNC1}0104912345678904',
  width: 350, height: 80,
)
```

コンビニ払込票 (Convenience store payment) :

```
// encodeConvenience: 44 → CODE-C
final gs1 = GS1_128();
final convData = '{FNC1}91234567890123456789012345678901234567890123';

// encodeConvenience (List<int>, String)
final result = gs1.encodeConvenience(convData);
final pattern = result.$1; //
final data44 = result.$2; // 44

// 2
final lines = gs1.buildConvenienceTextLines(data44);
print('1: ${lines.$1}');
print('2: ${lines.$2}');
```

> ヒント: GS1-128 の {FNC1} はデータ先頭に必須です。encode() は通常の GS1-128、encodeConvenience() はコンビニ払込票専用。用途に応じて使い分けてください。

GS1 DataBar 標準型 (GS1 DataBar 14) — 青果の量り売りに

```
import 'package:pao_barcode/pao_barcode.dart';

// OMNIDIRECTIONAL1
final db14 = GS1DataBar14(symbolType: SymbolType.omnidirectional);
final pattern = db14.encode('01234567890128');

// STACKED2 13X
final dbStacked = GS1DataBar14(symbolType: SymbolType.stacked);
dbStacked.encode('01234567890128');

// STACKED_OMNIDIRECTIONAL2 33X
final dbStackedOmni = GS1DataBar14(
  symbolType: SymbolType.stackedOmnidirectional);
dbStackedOmni.encode('01234567890128');
```

SymbolType 列挙型:

値	説明	高さ
SymbolType.omnidirectional	標準1行（デフォルト）	33X
SymbolType.stacked	2行積み（コンパクト）	13X
SymbolType.stackedOmnidirectional	2行積み（全方向）	33X

```
BarcodeWidget(
  barcode: GS1DataBar14(symbolType: SymbolType.omnidirectional),
  data: '01234567890128',
  width: 200, height: 60,
)
```

GS1 DataBar 限定型 (GS1 DataBar Limited) — 小型パッケージに

```
// 0 1 GTIN
BarcodeWidget(
  barcode: GS1DataBarLimited(),
  data: '01234567890128',
  width: 200, height: 50,
)
```

> ヒント: GS1 DataBar Limited は先頭桁が 0 か 1 のデータのみ対応します。先頭が 2 以上の場合はエンコードエラーになるので、GS1 DataBar 14 を使ってください。

GS1 DataBar 拡張型 (GS1 DataBar Expanded) — AI付きデータを柔軟に

```

// UNSTACKED1
final dbExp = GS1DataBarExpanded(
  symbolType: ExpandedSymbolType.unstacked,
);
dbExp.encode('{FNC1}0104912345678904');

// STACKED
final dbExpStacked = GS1DataBarExpanded(
  symbolType: ExpandedSymbolType.stacked,
  numColumns: 4, //
);
dbExpStacked.encode('{FNC1}0104912345678904');

```

ExpandedSymbolType 列挙型:

値	説明
ExpandedSymbolType.unstacked	1行 (デフォルト)
ExpandedSymbolType.stacked	複数行に分割

```

BarcodeWidget(
  barcode: GS1DataBarExpanded(
    symbolType: ExpandedSymbolType.stacked,
    numColumns: 4,
  ),
  data: '{FNC1}0104912345678904',
  width: 250, height: 80,
)

```

> ヒント: GS1 DataBar Expanded は有効期限やロット番号などの追加情報を GTIN と一緒にエンコードできる唯一の DataBar です。クーポンや精肉の消費期限ラベルに最適。

6.4 郵便カスタマバーコード — DMハガキの自動仕分け

```
import 'package:pao_barcode/pao_barcode.dart';

// 7
final yb = YubinCustomer();
final bars = yb.encode('1050011'); //
print(': ${bars.length}'); // → 67

// +
final bars2 = yb.encode('10500114-2-8');
// List<int>: 1=Long, 2=Semi-upper, 3=Semi-lower, 4=Timing
```

4-state パータイプ:

値	名前	見た目
1	Long	全高さ
2	Semi-long upper	上2/3
3	Semi-long lower	下2/3
4	Timing	中央1/3

```
BarcodeWidget(
  barcode: YubinCustomer(),
  data: '10500114-2-8',
  width: 250,
  height: 30,
  showText: false, //
)
```

> ヒント:
郵便カスタマバーコードは常に67本固定です。DMハガキにこのバーコードを印刷すると、郵便局の自動仕分け機が住所を高速に読み取り、配達が早くなります（大量発送時の割引対象にもなります）。

6.5 Flutter Web連携 — ブラウザでバーコードを表示・ダウンロード

DynamicBarcodeDemo — リアルタイムプレビュー

テキスト入力に連動してバーコードが即時更新される、Flutter Web 向けのデモです。

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:pao_barcode/pao_barcode.dart';

class DynamicBarcodeDemo extends StatefulWidget {
  const DynamicBarcodeDemo({super.key});

  @override
  State<DynamicBarcodeDemo> createState() => _DynamicBarcodeDemoState();
}

class _DynamicBarcodeDemoState extends State<DynamicBarcodeDemo> {
  final _controller = TextEditingController(text: 'Hello-2024');
  String _barcodeType = 'Code128';
  String? _errorMessage;

  dynamic _createBarcode() {
    switch (_barcodeType) {
      case 'Code128':
        return Code128();
      case 'QR':
        return QR();
      case 'JAN13':
        return JAN13();
      case 'Code39':
        return Code39();
      case 'DataMatrix':
        return DataMatrix();
      default:
        return Code128();
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Barcode Demo')),
      body: Padding(
        padding: const EdgeInsets.all(24),
        child: Column(
          children: [
            //
            DropdownButton<String>(
              value: _barcodeType,
              items: ['Code128', 'QR', 'JAN13', 'Code39', 'DataMatrix']
                .map((t) => DropdownMenuItem(value: t, child: Text(t)))
                .toList(),
              onChanged: (v) => setState(() => _barcodeType = v!),
            ),
            const SizedBox(height: 16),
          ],
        ),
      ),
    );
  }
}
```

```
//
TextField(
  controller: _controller,
  decoration: const InputDecoration(
    labelText: '',
    border: OutlineInputBorder(),
  ),
  onChanged: (_) => setState(() {}),
),
const SizedBox(height: 24),

//
Container(
  decoration: BoxDecoration(
    border: Border.all(color: Colors.grey.shade300),
    borderRadius: BorderRadius.circular(8),
  ),
  padding: const EdgeInsets.all(16),
  child: _buildBarcode(),
),
const SizedBox(height: 16),

// PNG
ElevatedButton.icon(
  icon: const Icon(Icons.download),
  label: const Text('PNG '),
  onPressed: _downloadPng,
),
],
),
);
}

Widget _buildBarcode() {
  try {
    _errorMessage = null;
    final bc = _createBarcode();
    final isQrOrDm = _barcodeType == 'QR' || _barcodeType == 'DataMatrix';
    return BarcodeWidget(
      barcode: bc,
      data: _controller.text,
      width: isQrOrDm ? 200 : 350,
      height: isQrOrDm ? 200 : 100,
    );
  } catch (e) {
    _errorMessage = e.toString();
    return Text(': $_errorMessage',
      style: const TextStyle(color: Colors.red));
  }
}
```

```

Future<void> _downloadPng() async {
  try {
    final renderer = CanvasRenderer();
    final bc = _createBarcode();
    final isQrOrDm = _barcodeType == 'QR' || _barcodeType == 'DataMatrix';
    final w = isQrOrDm ? 400 : 600;
    final h = isQrOrDm ? 400 : 150;
    final pngBytes = await renderer.renderToPng(
      bc, _controller.text, w, h,
    );

    // Base64 URI
    final dataUri = 'data:image/png;base64,${base64Encode(pngBytes)}';
    // Flutter Web: AnchorElement
    // (dart:html Web )
    print('Data URI : ${dataUri.length} ');
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(': $e')),
    );
  }
}

@override
void dispose() {
  _controller.dispose();
  super.dispose();
}
}

```

Base64 データ URI — Web表示やメール埋め込みに

```

import 'dart:convert';
import 'package:pao_barcode/pao_barcode.dart';

Future<String> barcodeToDataUri(dynamic barcode, String data,
  {int width = 400, int height = 100}) async {
  final renderer = CanvasRenderer();
  final pngBytes = await renderer.renderToPng(barcode, data, width, height);
  return 'data:image/png;base64,${base64Encode(pngBytes)}';
}

// :
// final uri = await barcodeToDataUri(Code128(), 'Hello-2024');
// → "..."
//
// HTML <img> :
// 

```

> ヒント: Base64 データ URI
は画像ファイルをサーバーに保存せずにブラウザ上で直接表示できます。メール本文への埋め込みや、JavaScript
との連携にも便利。ただし Base64
はデータ量が約1.33倍になるので、大量のバーコードを同時に表示する場合はファイル保存方式を検討してください。

7.1 共通メソッド（全バーコード） — BarcodeBase クラス

全てのバーコードクラス（BarcodeBase を継承）で使用できるメソッドです。

setOutputFormat(format)

出力形式を設定します。SVGモードに切り替える際に使用します。

パラメータ	型	説明
format	String	'png' または 'svg'

```
bc.setOutputFormat('svg');
```

デフォルト: 'png'

setForegroundColor(r, g, b, ¥[a¥])

前景色（バーの色）を設定します。

パラメータ	型	説明
r	int	赤成分 (0-255)
g	int	緑成分 (0-255)
b	int	青成分 (0-255)
a	int	透明度 (0-255、255=不透明)。デフォルト: 255

```
bc.setForegroundColor(0, 0, 128, 255); //
```

デフォルト: (0, 0, 0, 255) (黒)

setBackgroundColor(r, g, b, ¥[a¥])

背景色を設定します。

パラメータ	型	説明
r	int	赤成分 (0-255)
g	int	緑成分 (0-255)
b	int	青成分 (0-255)
a	int	透明度 (0-255、255=不透明)。デフォルト: 255

```
bc.setBackgroundColor(255, 255, 240, 255); //
```

デフォルト: (255, 255, 255, 255) (白)

setFitWidth(fit)

バーコードを指定幅いっぱいに描画するかどうかを設定します。

パラメータ	型	説明
fit	bool	true で幅いっぱいに描画

```
bc.setFitWidth(true);
```

デフォルト: false (GS1-128 は true)

draw(code, x, y, width, height)

バーコードを描画します (SVGモード時はSVG文字列を内部生成)。

パラメータ	型	説明
code	String	バーコードデータ
x	int	X座標
y	int	Y座標
width	int	幅
height	int	高さ

```
bc.setOutputFormat('svg');
bc.draw('Hello-2024', 0, 0, 400, 100);
```

戻り値: bool — 成功時 true

getSvg()

SVGモードで draw() 実行後、SVG文字列を取得します。

```
bc.setOutputFormat('svg');
bc.draw('Hello-2024', 0, 0, 400, 100);
final svgString = bc.getSvg();
// → "<?xml version="1.0" ...><svg ...>...</svg>"
```

戻り値: String — SVG文字列

> ヒント: getSvg() は setOutputFormat('svg') を呼んでから draw() を実行した後にのみ使用可能です。PNG

モードで呼ぶと `StateError` がスローされます。

7.2 1次元バーコード共通メソッド — BarcodeBase1D クラス

1次元バーコード (Code39, Code93, Code128, NW7, ITF, Matrix2of5, NEC2of5, JAN8, JAN13, UPCA, UPCE, GS1-128, GS1 DataBar 系) が共通で持つメソッドです。

encode(code)

バーコードデータをエンコードし、バー/スペース幅パターンを返します。

パラメータ	型	説明
code	String	バーコードデータ

```
final pattern = bc.encode('Hello-2024');
// pattern = [2, 1, 1, 2, 3, 2, ...]
// , , , ...
```

戻り値: List<int> — バーとスペースの幅が交互に並んだリスト (バーから開始)

textEvenSpacing プロパティ

バーコード下テキストの配置モードを制御します。

値	説明
true	均等分散 (デフォルト) — 各文字をバーコード幅に均等配置
false	中央集約 — テキスト全体を中央に連続配置

```
final bc = Code128();
bc.textEvenSpacing = true; //
bc.textEvenSpacing = false; //

//
bc.setTextEvenSpacing(false);
```

setShowText(show)

バーコード下のテキスト表示/非表示を切り替えます。

パラメータ	型	説明
show	bool	true で表示、false で非表示

```
bc.setShowText(false); //
```

デフォルト: true

JAN/UPC 表示パターンマトリクス — extendedGuard x textEvenSpacing

JAN/UPC バーコード (JAN8, JAN13, UPCA, UPCE) は extendedGuard と textEvenSpacing の組み合わせで4パターンの描画モードを持ちます。

extendedGuard	textEvenSpacing	ガードバー	テキスト配置	描画モード
true	true	延長あり	セクション内均等配置	JAN/UPC標準
true	false	延長あり	セクション内中央集約	JAN/UPC集約
false	true	延長なし (フラット)	バーコード幅全体に均等配置	フラット均等
false	false	延長なし (フラット)	バーコード幅中央に集約	フラット集約

デフォルト: extendedGuard = true, textEvenSpacing = true

```
final jan = JAN13();
jan.extendedGuard = true; //
jan.textEvenSpacing = true; //
```

> ヒント: スーパーやコンビニの商品に印刷されている JAN バーコードは extendedGuard=true, textEvenSpacing=true の標準パターンです。特別な理由がなければデフォルトのまま使ってください。

7.3 Code39

クラス: `Code39`

プロパティ	型	説明	デフォルト
showStartStop	bool	テキストにスタート/ストップ文字	true

```
final bc = Code39();  
bc.showStartStop = false; // *  
final pattern = bc.encode('HELLO-123');
```

入力可能文字: 数字 (0-9)、英大文字 (A-Z)、記号 (- . \$ / + % スペース)

7.4 Code93

クラス: `Code93`

```
final bc = Code93();  
final pattern = bc.encode('Hello123');
```

入力可能文字: ASCII 全文字 (内部でシフト文字による拡張エンコーディング)

7.5 Code128

クラス: `Code128`

プロパティ	型	説明	デフォルト
codeMode	CodeSet128	エンコードモード	CodeSet128.auto_

CodeSet128 列挙型:

値	説明
CodeSet128.auto_	自動最適化 (DP で最短幅を選択)
CodeSet128.codeA	CODE-A 固定
CodeSet128.codeB	CODE-B 固定
CodeSet128.codeC	CODE-C 固定 (数字ペア専用)

```
final bc = Code128();  
bc.codeMode = CodeSet128.codeC;  
final pattern = bc.encode('12345678');
```

7.6 GS1-128

クラス: `GS1_128`

encode(code)

GS1-128 形式でエンコードします。データ先頭に {FNC1} が必要です。

```
final gs1 = GS1_128();
final pattern = gs1.encode('{FNC1}0104912345678904');
```

encodeConvenience(code)

コンビニ収納代行バーコード（払込票）用のエンコードです。CODE-C 固定。

パラメータ	型	説明
code	String	{FNC1} + 44桁の数字

```
final gs1 = GS1_128();
final result = gs1.encodeConvenience('{FNC1}912345678901234567890123456789012345678901234567890123');
final pattern = result.$1; // List<int>
final data44 = result.$2; // String 44
```

戻り値: (List<int>, String) — レコード型（バーパターン, 44桁数字）

buildConvenienceTextLines(code)

コンビニバーコードの2行テキストを生成します。

パラメータ	型	説明
code	String	44桁数字（{FNC1} 付きでも可）

```
final lines = gs1.buildConvenienceTextLines(data44);
final line1 = lines.$1; // "(91)234567 89012345678901"
final line2 = lines.$2; // "23456789 01234 5"
```

戻り値: (String, String) — レコード型（1行目, 2行目）

> ヒント: `GS1_128` はコンストラクタで `setFitWidth(true)` を自動設定しています。バーコードは指定幅いっぱいに描画されます。

7.7 NW-7 (Codabar)

クラス: `NW7`

プロパティ	型	説明	デフォルト
showStartStop	bool	テキストにスタート/ストップ文字を	true

```
final bc = NW7();  
bc.showStartStop = false;  
final pattern = bc.encode('A1234567890B');
```

入力可能文字: 数字 (0-9)、記号 (- . \$ / + :)、スタート/ストップ (A, B, C, D)

7.8 ITF

クラス: `ITF`

```
final bc = ITF();  
final pattern = bc.encode('1234567890');
```

入力可能文字: 数字のみ (偶数桁、奇数桁の場合は先頭に0を自動付加)

7.9 Matrix 2of5

クラス: `Matrix2of5`

```
final bc = Matrix2of5();  
final pattern = bc.encode('1234567890');
```

入力可能文字: 数字のみ

7.10 NEC 2of5

クラス: `NEC2of5`

```
final bc = NEC2of5();  
final pattern = bc.encode('1234567890');
```

入力可能文字: 数字のみ

7.11 JAN-8 (EAN-8)

クラス: `JAN8`

プロパティ	型	説明	デフォルト
extendedGuard	bool	ガードバー延長	true
textEvenSpacing	bool	テキスト均等配置	true

```
final bc = JAN8();  
bc.extendedGuard = true;  
bc.textEvenSpacing = true;  
final pattern = bc.encode('4912345'); // 7 → CD
```

入力: 7桁 (CD自動計算) または 8桁 (CD込み)

7.12 JAN-13 (EAN-13)

クラス: `JAN13`

プロパティ	型	説明	デフォルト
extendedGuard	bool	ガードバー延長	true
textEvenSpacing	bool	テキスト均等配置	true

```
final bc = JAN13();  
final pattern = bc.encode('490123456789'); // 12 → CD
```

入力: 12桁 (CD自動計算) または 13桁 (CD込み)

7.13 UPC-A

クラス: `UPCA`

プロパティ	型	説明	デフォルト
extendedGuard	bool	ガードバー延長	true
textEvenSpacing	bool	テキスト均等配置	true

```
final bc = UPCA();  
final pattern = bc.encode('01234567890'); // 11 → CD
```

入力: 11桁 (CD自動計算) または 12桁 (CD込み)

7.14 UPC-E

クラス: `UPCE`

プロパティ	型	説明	デフォルト
extendedGuard	bool	ガードバー延長	true
textEvenSpacing	bool	テキスト均等配置	true

```
final bc = UPCE();  
final pattern = bc.encode('0123456'); // 7 → CD
```

入力: 7桁 (CD自動計算) または 8桁 (CD込み)

> ヒント: UPC-E は UPC-A のゼロ圧縮バージョンです。内部で UPC-A に展開してからチェックディジットを計算するため、元の UPC-A データとチェックディジットが一致します。

7.15 GS1 DataBar 標準型 (GS1 DataBar 14)

クラス: `GS1DataBar14`

コンストラクタ引数	型	説明	デフォルト
symbolType	SymbolType	シンボルタイプ	SymbolType.omnidirectional

SymbolType 列挙型:

値	説明
SymbolType.omnidirectional	標準1行
SymbolType.stacked	2行積み (コンパクト)
SymbolType.stackedOmnidirectional	2行積み (全方向対応)

```
final db = GS1DataBar14(symbolType: SymbolType.stacked);
final pattern = db.encode('01234567890128');

// STACKED
print('${db.rowCount}');
print('${db.patterns}');
print('${db.rowHeights}');
print('${db.getHumanReadable()}');
```

入力: 13桁または14桁の数字 (GTIN)

7.16 GS1 DataBar 限定型 (GS1 DataBar Limited)

クラス: `GS1DataBarLimited`

```
final db = GS1DataBarLimited();  
final pattern = db.encode('01234567890128');  
print('GTIN-14: ${db.gtin14}');
```

入力: 13桁または14桁の数字 (先頭桁が 0 または 1)

7.17 GS1 DataBar 拡張型 (GS1 DataBar Expanded)

クラス: `GS1DataBarExpanded`

コンストラクタ引数	型	説明	デフォルト
symbolType	ExpandedSymbolType	シンボルタイプ	ExpandedSymbolType.unstacked
numColumns	int	STACKED時の列数	2

ExpandedSymbolType 列挙型:

値	説明
ExpandedSymbolType.unstacked	1行
ExpandedSymbolType.stacked	複数行に分割

```
final db = GS1DataBarExpanded(
  symbolType: ExpandedSymbolType.stacked,
  numColumns: 4,
);
final pattern = db.encode('{FNC1}0104912345678904');
print(': ${db.rowCount}');
```

入力: {FNC1} + AI形式データ

7.18 郵便カスタマバーコード

クラス: `YubinCustomer`

```
final yb = YubinCustomer();
final bars = yb.encode('10500114-2-8');
// bars = [1, 3, 2, 4, ...] (67)
```

出力: List<int> — バータイプ (1=Long, 2=Semi-upper, 3=Semi-lower, 4=Timing) 、67本固定

入力: 郵便番号 (7桁) 、またはハイフン区切り住所表示番号

7.19 QRコード

クラス: `QR`

setErrorCorrectionLevel(level)

誤り訂正レベルを設定します。

QRErrorLevel 列挙型:

値	復元率	説明
QRErrorLevel.l	約7%	低
QRErrorLevel.m	約15%	中 (デフォルト)
QRErrorLevel.q	約25%	高
QRErrorLevel.h	約30%	最高

```
final qr = QR();
qr.setErrorCorrectionLevel(QRErrorLevel.h);
```

setVersion(version)

QRコードのバージョンを指定します。

パラメータ	型	説明
version	int	0 = 自動 (デフォルト)、1-40

```
qr.setVersion(10); // Version 10
```

getPattern(code)

QRコードの2Dパターンを生成します。

```
final matrix = qr.getPattern('Hello World');
// matrix[row][col] = true() / false()
print('${matrix.length} x ${matrix[0].length}');
```

戻り値: List<List<bool>> — 2Dブーリアン行列

QRコード バージョン/サイズ/容量テーブル:

Ver	モジュール	数字 (L)	英数字 (L)	バイト (L)	数字 (H)	バイト (H)
1	21x21	41	25	17	17	7
2	25x25	77	47	32	34	14
3	29x29	127	77	53	58	24
5	37x37	154	93	64	106	46
10	57x57	652	395	271	346	152
15	77x77	1,276	774	531	688	302
20	97x97	1,817	1,101	756	1,022	451
25	117x117	2,520	1,528	1,049	1,450	637
30	137x137	3,289	1,994	1,370	1,897	834
40	177x177	7,089	4,296	2,953	3,993	1,757

> ヒント: バージョンを
0 (自動) にしておくと、データ量と誤り訂正レベルに応じて最適なバージョンが自動選択されます。バージョンを固定する必要があるのは、ラベル印刷でQRコードのサイズを一定にしたい場合などです。

7.20 DataMatrix

クラス: `DataMatrix`

setCodeSize(size)

シンボルサイズを指定します。

DxCodeSize 列挙型 (主要) :

値	サイズ
DxCodeSize.dxCszAuto	自動 (デフォルト)
DxCodeSize.dxCszShapeAuto	正方形自動
DxCodeSize.dxCszRectAuto	長方形自動
DxCodeSize.dxCsz10x10	10x10
DxCodeSize.dxCsz12x12	12x12
DxCodeSize.dxCsz14x14	14x14
DxCodeSize.dxCsz16x16	16x16
DxCodeSize.dxCsz18x18	18x18
DxCodeSize.dxCsz20x20	20x20
DxCodeSize.dxCsz22x22	22x22
DxCodeSize.dxCsz24x24	24x24
DxCodeSize.dxCsz26x26	26x26
DxCodeSize.dxCsz32x32	32x32
DxCodeSize.dxCsz36x36	36x36
DxCodeSize.dxCsz40x40	40x40
DxCodeSize.dxCsz44x44	44x44
DxCodeSize.dxCsz48x48	48x48
DxCodeSize.dxCsz52x52	52x52
DxCodeSize.dxCsz64x64	64x64
DxCodeSize.dxCsz72x72	72x72
DxCodeSize.dxCsz80x80	80x80
DxCodeSize.dxCsz88x88	88x88
DxCodeSize.dxCsz96x96	96x96
DxCodeSize.dxCsz104x104	104x104
DxCodeSize.dxCsz120x120	120x120

値	サイズ
DxCodeSize.dxCsz132x132	132x132
DxCodeSize.dxCsz144x144	144x144
DxCodeSize.dxCsz8x18	8x18 (長方形)
DxCodeSize.dxCsz8x32	8x32 (長方形)
DxCodeSize.dxCsz12x26	12x26 (長方形)
DxCodeSize.dxCsz12x36	12x36 (長方形)
DxCodeSize.dxCsz16x36	16x36 (長方形)
DxCodeSize.dxCsz16x48	16x48 (長方形)

```
final dm = DataMatrix();
dm.setCodeSize(DxCodeSize.dxCsz20x20);
```

getPattern(code)

DataMatrix の2Dパターンを生成します。

```
final matrix = dm.getPattern('Hello DataMatrix');
// matrix[row][col] = true() / false()
```

戻り値: List<List<bool>> — 2Dブーリアン行列

入力: String、Uint8List、または List<int> (バイナリデータも対応)

> ヒント: DataMatrix の encode() と getPattern() は同じ結果を返します。GS1 DataMatrix を使う場合はデータ先頭に {FNC1} を付けてください。

7.21 PDF417

クラス: `PDF417`

setErrorLevel(level)

誤り訂正レベルを設定します。

PDF417ErrorLevel 列挙型:

値	訂正コードワード数
PDF417ErrorLevel.auto_	自動 (デフォルト)
PDF417ErrorLevel.level0	2
PDF417ErrorLevel.level1	4
PDF417ErrorLevel.level2	8
PDF417ErrorLevel.level3	16
PDF417ErrorLevel.level4	32
PDF417ErrorLevel.level5	64
PDF417ErrorLevel.level6	128
PDF417ErrorLevel.level7	256
PDF417ErrorLevel.level8	512

```
final pdf417 = PDF417();
pdf417.setErrorLevel(PDF417ErrorLevel.level5);
```

setAspectRatio(ratio)

アスペクト比を設定します。

パラメータ	型	説明
ratio	double	アスペクト比 (0.001~1000.0)

```
pdf417.setAspectRatio(0.5); //
```

setSizeMode(mode)

サイズ指定モードを設定します。

PDF417SizeMode 列挙型:

値	説明
PDF417SizeMode.auto_	自動 (デフォルト)
PDF417SizeMode.columns	列数を指定
PDF417SizeMode.rows	行数を指定
PDF417SizeMode.columnsAndRows	列数・行数を同時指定

setColumns(columns)

列数を指定します (setSizeMode(PDF417SizeMode.columns) と併用)。

パラメータ	型	説明
columns	int	列数 (0-30)

setRows(rows)

行数を指定します (setSizeMode(PDF417SizeMode.rows) と併用)。

パラメータ	型	説明
rows	int	行数 (0-90)

```
final pdf417 = PDF417();
pdf417.setErrorLevel(PDF417ErrorLevel.level3);
pdf417.setSizeMode(PDF417SizeMode.columnsAndRows);
pdf417.setColumns(6);
pdf417.setRows(10);
```

```
final matrix = pdf417.getPattern('Hello PDF417');
```

getPattern(code) 戻り値: List<List<bool>> — 2Dブーリアン行列

> ヒント: PDF417 の列数は少ないほど横幅がコンパクトになりますが、行数が増えます。ラベル印刷では columns=4 程度が使いやすいバランスです。

7.22 CanvasRenderer

クラス: `CanvasRenderer`

Flutter の Canvas にバーコードを描画するレンダラーです。

インポート:

```
import 'package:pao_barcode/pao_barcode.dart';
```

コンストラクタ

```
CanvasRenderer({
  Color foreColor = Colors.black,
  Color backColor = Colors.white,
  bool showText = true,
  double fontSize = 14,
  double quietZone = 0,
})
```

パラメータ	型	説明	デフォルト
foreColor	Color	前景色 (バーの色)	Colors.black
backColor	Color	背景色	Colors.white
showText	bool	テキスト表示	true
fontSize	double	フォントサイズ	14
quietZone	double	クワイエットゾーン比率	0

draw(canvas, barcode, data, x, y, width, height, {...})

バーコードを Canvas に描画します。全19タイプに対応。

パラメータ	型	説明
canvas	Canvas	Flutter の Canvas
barcode	dynamic	エンコーダーインスタンス
data	String	バーコードデータ
x	double	左端X座標
y	double	上端Y座標
width	double	幅
height	double	高さ

名前付きオプション:

キー	型	説明
showTextOverride	bool?	コンストラクタ設定の一時上書き
fontSizeOverride	double?	コンストラクタ設定の一時上書き

```
final renderer = CanvasRenderer();
renderer.draw(canvas, Code128(), 'Hello', 10, 10, 300, 80);
```

drawGs1128Convenience(canvas, barcode, data, x, y, width, height, {...})

GS1-128 コンビニ払込票バーコードを2行テキスト付きで描画します。

パラメータ	型	説明
canvas	Canvas	Flutter の Canvas
barcode	dynamic	GS1_128 インスタンス
data	String	{FNC1} + 44桁数字
x	double	左端X座標
y	double	上端Y座標
width	double	幅
height	double	高さ

```
final renderer = CanvasRenderer();
final gs1 = GS1_128();
renderer.drawGs1128Convenience(
  canvas, gs1, '{FNC1}91234567890123456789012345678901234567890123',
  10, 10, 350, 80);
```

renderToPng(barcode, data, width, height, {...})

バーコードを PNG バイト列としてレンダリングします。

パラメータ	型	説明
barcode	dynamic	エンコーダーインスタンス
data	String	バーコードデータ
width	int	画像幅 (ピクセル)
height	int	画像高さ (ピクセル)

戻り値: Future<Uint8List> — PNG バイト列

```
final renderer = CanvasRenderer();  
final pngBytes = await renderer.renderToPng(Code128(), 'Hello', 400, 100);
```

> ヒント: `renderToPng()` は非同期メソッドです。await を使うか、`.then()` チェーンで処理してください。内部で `PictureRecorder` を使うため、Flutter のレンダリングエンジンが必要です（コンソールアプリでは使用不可）。

7.23 PdfRenderer

クラス: `PdfRenderer`

pdf パッケージの PdfGraphics にバーコードを描画するレンダラーです。

インポート:

```
import 'package:pao_barcode/pao_barcode.dart';
// + pdf
import 'package:pdf/pdf.dart';
```

コンストラクタ

```
PdfRenderer(
  PdfGraphics canvas, {
  PdfColor foreColor = PdfColors.black,
  PdfColor backColor = PdfColors.white,
  bool showText = true,
  String fontName = 'Helvetica',
  double fontSize = 8,
  double quietZone = 0,
})
```

パラメータ	型	説明	デフォルト
canvas	PdfGraphics	pdfパッケージのキャンバス	(必須)
foreColor	PdfColor	前景色	PdfColors.black
backColor	PdfColor	背景色	PdfColors.white
showText	bool	テキスト表示	true
fontName	String	フォント名	'Helvetica'
fontSize	double	フォントサイズ (pt)	8
quietZone	double	クワイエットゾーン比率	0

draw(barcode, data, x, y, width, height, {...})

バーコードを PDF キャンバスに描画します。全19タイプに対応。

パラメータ	型	説明
barcode	dynamic	エンコーダーインスタンス
data	String	バーコードデータ
x	double	左端X座標 (ポイント)
y	double	下端Y座標 (ポイント)
width	double	幅 (ポイント)
height	double	高さ (ポイント)

名前付きオプション:

キー	型	説明
showTextOverride	bool?	コンストラクタ設定の一時上書き
fontSizeOverride	double?	コンストラクタ設定の一時上書き

```
final renderer = PdfRenderer(canvas);
renderer.draw(Code128(), 'Hello', 50, 700, 200, 50);
```

drawGs128Convenience(barcode, data, x, y, width, height, {...})

GS1-128 コンビニ払込票バーコードを2行テキスト付きで描画します。

```
final renderer = PdfRenderer(canvas, fontSize: 7);
final gs1 = GS1_128();
renderer.drawGs128Convenience(
  gs1, '{FNC1}91234567890123456789012345678901234567890123',
  50, 600, 350, 80);
```

> ヒント: PdfRenderer の座標系は PDF 標準 (原点が左下、Y上向き) です。y=700 は用紙の上の方、y=50 は用紙の下の方を意味します。A4 用紙は高さ 842pt なので、上端付近に描画したい場合は y=750 あたりを指定します。

7.24 BarcodeWidget

クラス: `BarcodeWidget`

Flutter の Widget ツリーに直接組み込めるバーコード表示 Widget です。

インポート:

```
import 'package:pao_barcode/pao_barcode.dart';
```

プロパティ一覧

プロパティ	型	説明	デフォルト
barcode	dynamic	エンコーダーインスタンス	(必須)
data	String	バーコードデータ	(必須)
width	double?	幅 (null で親に合わせる)	null
height	double?	高さ (null で親に合わせる)	null
foreColor	Color	前景色	Colors.black
backColor	Color	背景色	Colors.white
showText	bool	テキスト表示	true
fontSize	double	フォントサイズ	14
quietZone	double	クワイエットゾーン比率	0

```
BarcodeWidget(
  barcode: Code128(),
  data: 'Hello-2024',
  width: 300,
  height: 100,
  foreColor: Colors.black,
  backColor: Colors.white,
  showText: true,
  fontSize: 14,
  quietZone: 0.02,
)
```

> ヒント: BarcodeWidget は内部で CustomPaint + _BarcodePainter を使っています。shouldRepaint() が data、foreColor、backColor、showText、fontSize、quietZone の変更を検出して、必要なときだけ再描画します。

Flutter / Dart バージョン

項目	要件
Flutter	3.0 以上
Dart	3.0 以上
Null Safety	対応済み

対応プラットフォーム

プラットフォーム	対応状況	備考
iOS		iPhone / iPad
Android		API 21 以上
Web		Chrome, Safari, Firefox, Edge
Windows		Windows 10 以上
macOS		macOS 10.14 以上
Linux		x64

Pure Dart で書かれているため、ネイティブプラグインやFFIは一切不要です。Flutter が動くすべてのプラットフォームでそのまま動作します。

依存パッケージ

パッケージ	バージョン	必要な場面	用途
pao_barcode	1.1	常に必要	バーコードエンコーダー + CanvasRenderer
pdf	^3.10.0	PDF帳票出力時	PdfRenderer で PDF に直接描画
printing	^5.11.0	PDF印刷/プレビュー時	PDF印刷ダイアログ、日本語フォント

> ヒント: BarcodeWidget、CanvasRenderer、SVG出力は pao_barcode 単体で動きます。pdf / printing パッケージは PDF帳票を作るときだけ追加してください。

使用許諾

pao_barcode

の使用について、利用者様と有限会社パオ・アット・オフィス（以下「弊社」）は、以下の各項目に同意するものとします。

1. 使用許諾書

この使用許諾書は、利用者様がお使いのパソコンにおいて

pao_barcode

を使用する場合に同意しなければならない契約書です。

2. 同意

利用者様が pao_barcode を使用する時点で、本使用許諾書に同意されたものとします。

3. ライセンスの購入

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

4. 著作権

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

5. 免責

pao_barcode

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

6. 禁止事項

pao_barcode

およびその複製物を第三者に譲渡・貸与することはできません。開発ツールとしての再販・再配布を禁止します。ただし、モジュールとして組み込みを行い再販・再配布する場合は問題ございません。

7. 保証の範囲

弊社は pao_barcode の仕様を予告なしに変更することがあります。利用者様への情報提供は弊社 Web サイトにて行います。

8. 適用期間

本使用許諾条件は利用者様が pao_barcode を使用した日より有効です。

ライセンス

項目	内容
製品名	pao_barcode (Pure Dart版)
ライセンス価格	22,000円（税込） / 20,000円（税抜）
ライセンス単位	1ライセンス = 開発PC 1台
ランタイムライセンス	不要（配布自由）
ソースコード	なし

ライセンスの考え方:

- ・ 開発者1人（PC 1台）につき1ライセンスが必要です
- ・ 開発したアプリの配布先（エンドユーザー）にはライセンス不要です
- ・ iOS / Android / Web / Desktop — 1ライセンスで全プラットフォーム対応
- ・ 社内利用・商用利用いずれも同一価格です

試用版について

ライセンス購入前に全機能をお試しいただけます。

項目	試用版	ライセンス版
機能	全機能利用可	全機能利用可
バーコード種類	19種すべて	19種すべて
出力制限	SAMPLE 透かし表示	なし
利用期間	無制限	無制限

試用版では生成されるバーコード画像の左上に赤字の透かしが表示されます。ライセンス購入後、ライセンスキーを設定すると透かしが消えます。

"SAMPLE"

お問い合わせ

項目	内容
会社名	有限会社 パオ・アット・オフィス
Webサイト	https://www.pao.ac/
メール	info@pao.ac
所在地	日本

関連製品

製品	プラットフォーム	価格 (税込)
pao_barcode (Pure Dart版)	Flutter (全6プラットフォーム)	22,000円
pao_barcode (Pure Python版・ソースコード付)	Python	22,000円
pao_barcode (C++ Import版)	Python (WASM/FFI)	11,000円

バージョン 1.1

(C) 2026 有限会社 パオ・アット・オフィス