Barcode.Flutter

Flutter バーコード生成ライブラリ（Plugin / WASM版）

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. たった2行で、19種のバーコードをFlutterアプリに。
2. 1.1 Barcode.Flutter C++ Import版とは
3. 1.2 特長
4. 1.3 対応バーコード一覧
5. 1.4 他のBarcode.Flutter製品との違い
6. 1.5 デモサイト
7. 2.1 PNG画像出力 — そのまま画面に表示
8. 2.2 SVGベクター出力 — 拡大しても美しい
9. 2.3 カスタマイズ — 色もテキストも思いのままに
10. 3.1 WASM版のインストール（全プラットフォーム対応）
11. 3.2 FFI版のインストール（ネイティブプラットフォーム）
12. 3.3 動作確認
13. 4.1 PNG出力（Base64）
14. 4.2 SVG出力
15. 4.3 画面に表示する
16. 4.4 バイト列取得（FFI版のみ）
17. 5.1 Image.memoryで表示（PNG）
18. 5.2 動的に生成するフォーム
19. 5.3 リスト表示（商品一覧）
20. 6.1 1次元バーコード — 物流・工業の定番
21. 6.2 2次元バーコード — 大容量データを小さな面積に
22. 6.3 GS1系バーコード — 流通のインフラ
23. 6.4 商品・郵便バーコード — 身の回りのバーコード
24. 7.1 共通メソッド（全バーコード）
25. 7.2 1次元バーコード共通メソッド
26. 7.3 Code39
27. 7.4 Code93
28. 7.5 Code128

29. 7.6 GS1-128
30. 7.7 NW-7（Codabar）
31. 7.8 ITF
32. 7.9 Matrix 2of5
33. 7.10 NEC 2of5
34. 7.11 JAN-8（EAN-8）
35. 7.12 JAN-13（EAN-13）
36. 7.13 UPC-A
37. 7.14 UPC-E
38. 7.15 GS1 DataBar 標準型
39. 7.16 GS1 DataBar 限定型
40. 7.17 GS1 DataBar 拡張型
41. 7.18 郵便カスタマバーコード
42. 7.19 QRコード
43. 7.20 DataMatrix
44. 7.21 PDF417
45. 比較表
46. 選び方
47. 使用許諾
48. ライセンス
49. お問い合わせ
50. 関連製品

たった2行で、19種のバーコードをFlutterアプリに。

ユーザーズマニュアル

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

- 1.1 Barcode.Flutter C++ Import版とは
- 1.2 特長
- 1.3 対応バーコード一覧
- 1.4 他のBarcode.Flutter製品との違い
- 1.5 デモサイト

2. できること

- 2.1 PNG画像出力 — そのまま画面に表示
- 2.2 SVGベクター出力 — 拡大しても美しい
- 2.3 カスタマイズ — 色もテキストも思いのままに

3. 導入方法

- 3.1 WASM版のインストール（全プラットフォーム対応）
- 3.2 FFI版のインストール（ネイティブプラットフォーム）
- 3.3 動作確認

4. クイックスタート — 最初の1本を生成しよう

- 4.1 PNG出力（Base64）
- 4.2 SVG出力
- 4.3 画面に表示する
- 4.4 バイト列取得（FFI版のみ）

5. 画面に表示する — Widget連携

- 5.1 Image.memoryで表示（PNG）
- 5.2 動的に生成するフォーム
- 5.3 リスト表示（商品一覧）

6. 実践サンプル集

- 6.1 1次元バーコード — 物流・工業の定番
- 6.2 2次元バーコード — 大容量データを小さな面積に
- 6.3 GS1系バーコード — 流通のインフラ
- 6.4 商品・郵便バーコード — 身の回りのバーコード

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 Code39
- 7.4 Code93
- 7.5 Code128
- 7.6 GS1-128
- 7.7 NW-7（Codabar）
- 7.8 ITF
- 7.9 Matrix 2of5
- 7.10 NEC 2of5
- 7.11 JAN-8（EAN-8）
- 7.12 JAN-13（EAN-13）
- 7.13 UPC-A
- 7.14 UPC-E
- 7.15 GS1 DataBar 標準型
- 7.16 GS1 DataBar 限定型
- 7.17 GS1 DataBar 拡張型
- 7.18 郵便カスタマバーコード
- 7.19 QRコード
- 7.20 DataMatrix
- 7.21 PDF417

8. WASM版とFFI版 — どちらを選ぶ？

9. 動作環境

10. ライセンス・お問い合わせ

- 10.1 使用許諾
- 10.2 ライセンス

1.1 Barcode.Flutter C++ Import版とは

商品のパッケージ、宅配便の送り状、病院の検体ラベル、工場の部品管理タグ——。

私たちの日常には、驚くほど多くのバーコードが存在しています。

Barcode.Flutter C++ Import版 は、そのバーコードを Flutter アプリから簡単に生成 できるライブラリです。

C++で開発された高性能バーコードエンジンを　Dart　から直接呼び出す仕組みで、1次元・2次元あわせて　全19種のバーコードを、PNG画像 または SVGベクター で出力できます。

必要なコードは、たった2行。

```
final bc = Code128();
final result = bc.draw('Hello-2026', 400, 100);
```

これだけで、400×100ピクセルのバーコードが Base64 PNG画像として手に入ります。

Image.memory() に渡せば、Flutter の画面にそのまま表示できます。

1.2 特長

特長	説明
たった2行で生成	インスタンス作成 → draw() 呼び出し。それだけ
C++譲りの高速描画	C++で実装されたエンジンが高速にバーコードを生成
全19種バーコード	1D・2D・GS1・郵便まで、業務で必要なバーコードを網羅
PNG / SVG 両対応	画面表示にはPNG、印刷にはSVGと、用途で使い分け可能
全プラットフォーム対応	WASM版1つで Web + Windows + macOS + Linux + iOS + Android すべてに対応
Flutter ネイティブ連携	Image.memory() に渡すだけで画面に表示できる
豊富なカスタマイズ	色、テキスト、バー幅調整、均等割付まで細かく制御可能
コンビニバーコード対応	GS1-128の標準料金代理収納用バーコードにも完全対応

1.3 対応バーコード一覧

1次元バーコード（12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、軍事規格（MIL-STD）にも採用
Code93	Code93	Code39の高密度版。郵便・物流で活用
Code128	Code128	物流の標準。ASCII全文字をエンコード可能
GS1-128	GS1_128	医薬品・物流。ロット番号や有効期限をAIで管理
NW-7 (Codabar)	NW7	宅配便の送り状、図書館の貸出管理でおなじみ
ITF	ITF	段ボール箱のインジケーター。物流の現場で毎日活躍
Matrix 2of5	Matrix2of5	工業用途。数字のみのシンプルな構成
NEC 2of5	NEC2of5	日本の工業現場で使われるバリエーション
JAN-8 (EAN-8)	Jan8	小さな商品用。ガムやキャンディーのパッケージに
JAN-13 (EAN-13)	Jan13	日本の商品バーコードの標準。スーパーのレジで毎日活躍
UPC-A	UPC_A	北米の商品コード。12桁
UPC-E	UPC_E	UPC-Aの短縮版。小さなパッケージに

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	スーパーの青果・精肉売り場。重量や価格を直接エンコー
GS1 DataBar 限定型	GS1DataBarLimited	小型商品向けのコンパクト版
GS1 DataBar 拡張型	GS1DataBarExpanded	可変長データ対応。クーポンや特売情報も格納

郵便バーコード（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	郵便物の住所バーコード。自動区分機で高速仕分け

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	URL、決済、名刺交換——。日本発、世界で最も普及した
DataMatrix	DataMatrix	電子部品の超小型マーキング。GS1ヘルスケアでも標準
PDF417	PDF417	運転免許証、搭乗券。大容量データを1本に集約

1.4 他のBarcode.Flutter製品との違い

パオ・アット・オフィスの Flutter バーコードライブラリには、用途に応じた2つの製品があります。

C++ Import版（本製品）	Pure Dart版
パッケージ	barcode_pao_wasm（Web） barcode_pao（ネイティブ）
こんな方に	手軽にバーコード画像を表示したい
描画方法	draw() で Base64 / SVG を直接取得
出力形式	PNG（Base64）/ SVG
Widget	Image.memory() で表示
価格	11,000円（税込）

C++ Import版（本製品）がおすすめな場面:

- バーコード画像をシンプルに画面に表示したい
- draw() → Image.memory() のパターンで十分
- Flutter Web でバーコードを使いたい

Pure Dart版がおすすめな場面:

- PdfRenderer で PDF帳票にバーコードを直接描画したい
- CanvasRenderer + CustomPainter でカスタムUIに統合したい
- BarcodeWidget でバーコードを宣言的に使いたい
- ネイティブ依存なしで全プラットフォームに対応したい

1.5 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。全バーコードの生成をお試しいただけます。

- Flutter WASM版デモ: https://www.pao.ac/demo/barcode-flutter-wasm/
- Flutter Pure Dart版デモ: https://www.pao.ac/demo/barcode-flutter/

2.1 PNG画像出力 — そのまま画面に表示

draw() メソッドを呼ぶだけで、Base64エンコードされたPNG画像が返ってきます。

Flutter の Image.memory() にデコードして渡すだけで、画面にバーコードが表示されます。

```
import 'dart:convert';

final bc = Code128();
bc.setShowText(true);
final base64 = bc.draw('Hello-2026', 400, 100);

// Base64 Image.memory()
final prefix = '';
final bytes = base64Decode(base64.substring(prefix.length));
Image.memory(bytes, fit: BoxFit.contain);
```

PNGが向いている場面:

- 画面上でのバーコードプレビュー
- 商品一覧にバーコードを表示する
- ネイティブの画像処理と組み合わせたい場合

2.2 SVGベクター出力 — 拡大しても美しい

出力形式を 'svg' に切り替えるだけで、ベクター形式のSVG文字列が得られます。

どれだけ拡大しても線がぼやけないため、印刷用途に最適です。

```
final bc = Code128();
bc.setShowText(true);
bc.setOutputFormat('svg');

final svgString = bc.draw('Hello-2026', 400, 100);
// → "<svg xmlns=..."

// flutter_svg :
// SvgPicture.string(svgString)
```

SVGが向いている場面:

- ラベル印刷（拡大しても劣化しない）
- ファイルサイズを小さく抑えたい場合
- flutter_svg でベクター表示したい場合

> ヒント: 同じバーコードオブジェクトで setOutputFormat() を切り替えれば、PNG版とSVG版の両方を生成できます。プレビューはPNG、エクスポートはSVG、という使い分けも簡単です。

2.3 カスタマイズ — 色もテキストも思いのままに

色を変える

前景色（バーの色）と背景色を自由に指定できます。

透明度（アルファ値）にも対応しているので、背景を透明にすることも可能です。

```
//
bc.setForegroundColor(0, 0, 128, 255);
bc.setBackgroundColor(255, 255, 240, 255);

//
bc.setBackgroundColor(0, 0, 0, 0);
```

テキスト表示を調整する

バーコード下部のテキスト（ヒューマンリーダブル）は、表示・非表示だけでなく、サイズや配置まで細かく調整できます。

```
bc.setShowText(true);              //
bc.setTextFontScale(1.2);         //
bc.setTextGap(0.5);               //
bc.setTextEvenSpacing(true);      // 1
```

> ヒント: setTextEvenSpacing(true) にすると、テキストが各バーの真下に揃って配置されます。見た目がすっきりするので、一般的な1Dバーコードではおすすめです。

バー幅を微調整する（印刷のにじみ対策）

実際に印刷すると、インクのにじみで黒バーが太くなることがあります。

バーコードリーダーの読み取り精度が落ちてしまう場合は、この機能で補正しましょう。

```
bc.setPxAdjustBlack(-1);    // 1px
bc.setPxAdjustWhite(1);     // 1px
```

幅ぴったり描画

指定した幅にバーコードをぴったり収めたい場合に使います。

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

2つのパッケージがあります。WASM版がおすすめです。

- WASM版（barcode_pao_wasm） — 全プラットフォーム対応（Web + Windows + macOS + Linux + iOS + Android）
- FFI版（barcode_pao） — ネイティブのみ（Windows / macOS / Linux / iOS / Android）

> なぜWASM版がおすすめ？ WASM版はバージョン1.1でクロスプラットフォーム対応になりました。WebではWebAssembly + JS interop、ネイティブでは dart:ffi で動作し、同じAPIで全プラットフォームをカバーします。FFI版は drawBytes() などの追加機能がありますが、WASM版1つで済むシンプルさが魅力です。

3.1 WASM版のインストール（全プラットフォーム対応）

pubspec.yaml に以下を追加します：

```
dependencies:
  barcode_pao_wasm:
    path: ../path/to/barcode_pao_wasm
  barcode_pao:                              #
    path: ../path/to/barcode_pao
```

> 注意: barcode_pao_wasm は Web 実行時には WebAssembly を使いますが、ネイティブ実行時には barcode_pao プラグインの DLL/SO を dart:ffi で呼び出します。そのため、ネイティブプラットフォームで使う場合は barcode_pao も一緒に追加してください。Flutter Web のみで使う場合は barcode_pao_wasm だけで動作します。

WASMファイルの配置（Flutter Web の場合のみ）：

web/wasm/ ディレクトリに以下の2ファイルを配置してください：

```
your_project/
├── web/
│   ├── wasm/
│   │   ├── barcode.js       ← WASM
│   │   └── barcode.wasm     ← WebAssembly
│   └── index.html
└── lib/
    └── main.dart
```

web/index.html の <head> セクションに以下を追加します：

```
<script src="wasm/barcode.js"></script>
```

> ヒント:
ネイティブプラットフォーム（Windows/macOS/Linux等）で実行する場合、WASMファイルの配置は不要です。barcode_pao プラグインが自動的にネイティブDLLをバンドルします。

要件: Flutter 3.0+, Dart 3.0+

3.2 FFI版のインストール（ネイティブプラットフォーム）

> 通常は WASM版をお使いください。 FFI版は drawBytes() や drawRect() など WASM版にない追加メソッドが必要な場合にのみ直接使います。WASM版はネイティブプラットフォームでも FFI版と同じ C++ エンジンを使用するため、性能差はありません。

pubspec.yaml に以下を追加します：

```
dependencies:
  barcode_pao:
    path: ../path/to/barcode_pao
```

ネイティブライブラリの配置:

プラットフォームに応じて、ビルド済みのネイティブライブラリを配置します：

プラットフォーム	ファイル	配置先
Windows	barcode_pao.dll	Flutter プラグインが自動バンドル
macOS	libbarcode_pao.dylib	バンドル内
Linux	libbarcode_pao.so	LD_LIBRARY_PATH 内
iOS	libbarcode_pao.a	Xcode プロジェクトにリンク
Android	libbarcode_pao.so	jniLibs/ 内

要件: Flutter 3.0+, Dart 3.0+

3.3 動作確認

WASM版

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

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initBarcodeWasm();    // 1

  final bc = Code128();
  bc.setShowText(true);
  final result = bc.draw('TEST-1234', 400, 100);
  print('OK: ${result.substring(0, 50)}...');
}
```

FFI版

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

void main() {
  final bc = Code128();
  try {
    bc.setShowText(true);
    final result = bc.draw('TEST-1234', 400, 100);
    print('OK: ${result.substring(0, 50)}...');
  } finally {
    bc.dispose();    //
  }
}
```

... で始まる長い文字列が表示されれば成功です。

> ヒント: WASM版とFFI版は ほぼ同じAPI です。主な違いは「WASM版は initBarcodeWasm() が必要」「FFI版は dispose() が必要」の2点だけ。以降のサンプルでは WASM版の import を使用しますが、FFI版でも同様に動きます。

ここでは、コピー＆ペーストですぐ動くサンプルを紹介します。

4.1 PNG出力（Base64）

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

final bc = Code128();
bc.setShowText(true);
bc.setTextEvenSpacing(true);

final base64 = bc.draw('Hello-2026', 400, 100);
// → "..."
```

draw() の戻り値は ... 形式の Data URI 文字列です。

4.2 SVG出力

```
final bc = Code128();
bc.setShowText(true);
bc.setTextEvenSpacing(true);
bc.setOutputFormat('svg');

final svgString = bc.draw('Hello-2026', 400, 100);
// → "<svg xmlns=..."
```

SVGモードでは <svg xmlns="...">...</svg> 形式の文字列が返ります。

4.3 画面に表示する

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

class BarcodeView extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final bc = Code128();
    bc.setShowText(true);
    final base64 = bc.draw('Hello-2026', 400, 100);

    // ""
    final prefix = '';
    final bytes = base64Decode(base64.substring(prefix.length));

    return Image.memory(bytes, fit: BoxFit.contain);
  }
}
```

4.4 バイト列取得（FFI版のみ）

FFI版では drawBytes() で PNG のバイト列（Uint8List）を直接取得できます。Base64のデコード処理が不要になるため、大量のバーコードを生成する場合に便利です。

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

final bc = Code128();
bc.setShowText(true);

try {
  final pngBytes = bc.drawBytes('Hello-2026', 400, 100);
  // → Uint8ListPNG

  Image.memory(pngBytes, fit: BoxFit.contain);
} finally {
  bc.dispose();    // FFI
}
```

> 注意: drawBytes() は FFI版（barcode_pao）のみで利用可能です。WASM版には含まれません。

5.1 Image.memoryで表示（PNG）

最もシンプルな表示方法です。

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

class SimpleBarcodeWidget extends StatelessWidget {
  final String data;
  final int width;
  final int height;

  const SimpleBarcodeWidget({
    required this.data,
    this.width = 400,
    this.height = 100,
    super.key,
  });

  @override
  Widget build(BuildContext context) {
    final bc = Code128();
    bc.setShowText(true);

    try {
      final base64 = bc.draw(data, width, height);
      final prefix = '';
      final bytes = base64Decode(base64.substring(prefix.length));
      return Image.memory(bytes, fit: BoxFit.contain);
    } catch (e) {
      return Text(': $e', style: TextStyle(color: Colors.red));
    }
  }
}

//
SimpleBarcodeWidget(data: 'HELLO-2026')
```

5.2 動的に生成するフォーム

テキストフィールドに入力した文字をリアルタイムでバーコードにする例です。

```
class BarcodeGeneratorPage extends StatefulWidget {
  @override
  State<BarcodeGeneratorPage> createState() => _BarcodeGeneratorPageState();
}

class _BarcodeGeneratorPageState extends State<BarcodeGeneratorPage> {
  String _code = 'HELLO-2026';
  Uint8List? _imageBytes;
  String? _error;

  void _generate() {
    final bc = Code128();
    bc.setShowText(true);

    try {
      final base64 = bc.draw(_code, 400, 100);
      final prefix = '';
      setState(() {
        _imageBytes = base64Decode(base64.substring(prefix.length));
        _error = null;
      });
    } catch (e) {
      setState(() {
        _imageBytes = null;
        _error = e.toString();
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        TextField(
          decoration: InputDecoration(labelText: ''),
          onChanged: (v) => _code = v,
        ),
        SizedBox(height: 8),
        ElevatedButton(
          onPressed: _generate,
          child: Text(''),
        ),
        SizedBox(height: 16),
        if (_error != null)
          Text(_error!, style: TextStyle(color: Colors.red)),
        if (_imageBytes != null)
          Image.memory(_imageBytes!, fit: BoxFit.contain),
      ],
    );
  }
}
```

5.3 リスト表示（商品一覧）

商品リストにバーコードを表示する実践的な例です。

```
class ProductListPage extends StatelessWidget {
  final products = [
    {'name': '', 'jan': '4901234567894', 'price': 98},
    {'name': ' ', 'jan': '4912345678904', 'price': 130},
    {'name': '', 'jan': '4923456789014', 'price': 248},
  ];

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: products.length,
      itemBuilder: (context, index) {
        final p = products[index];
        final bc = Jan13();
        bc.setShowText(true);
        bc.setExtendedGuard(true);

        final base64 = bc.draw(p['jan'] as String, 250, 100);
        final prefix = '';
        final bytes = base64Decode(base64.substring(prefix.length));

        return Card(
          child: ListTile(
            title: Text(p['name'] as String),
            subtitle: Text('¥${p['price']}'),
            trailing: SizedBox(
              width: 150,
              child: Image.memory(bytes, fit: BoxFit.contain),
            ),
          ),
        );
      },
    );
  }
}
```

ここからは、バーコードの種類ごとに実践的なサンプルを紹介します。

各バーコードが「どんな場面で使われているか」も添えていますので、用途に合ったバーコードを選ぶ参考にしてください。

以降のサンプルでは　WASM版の　import　を使用しますが、FFI版でも同様に動作します（FFI版では　dispose()　の呼び出しが必要です）。

6.1 1次元バーコード — 物流・工業の定番

Code39 — 工場で最も古くから使われるバーコード

英数字と一部の記号を表現できます。スタート/ストップコード（*）で囲まれるのが特徴です。

```
final bc = Code39();
bc.setShowText(true);
bc.setTextEvenSpacing(true);
bc.setShowStartStop(true);     // *HELLO-123*

final result = bc.draw('HELLO-123', 400, 100);
```

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

Code93 — Code39の高密度版

Code39と同じ文字を、より狭いスペースでエンコードできます。さらにASCII全文字に対応。

```
final bc = Code93();
bc.setShowText(true);
bc.setTextEvenSpacing(true);

final result = bc.draw('Code93-Test', 400, 100);
```

Code128 — 物流業界の標準

ASCII全文字に対応し、数字は高密度でエンコードできるため、物流伝票で広く使われています。コードモードは通常 'AUTO' にしておけば、最短幅になるよう自動で最適化されます。

```
final bc = Code128();
bc.setShowText(true);
bc.setTextEvenSpacing(true);
bc.setCodeMode('AUTO');    // AUTO / A / B / C

final result = bc.draw('Hello-2026', 400, 100);
```

コードモード	説明
'AUTO'	自動で最短幅に最適化（おすすめ）
'A'	制御文字 + 数字 + 英大文字
'B'	数字 + 英大文字 + 英小文字 + 記号
'C'	数字のみ（2桁ずつ高密度エンコード）

> ヒント: 制御文字を入力するには {CR}, {LF}, {TAB} のように中括弧で囲みます。

NW-7 (Codabar) — 宅配便の送り状でおなじみ

先頭と末尾にスタート/ストップコード（A/B/C/D）を付けるのがルールです。

```
final bc = NW7();
bc.setShowText(true);
bc.setTextEvenSpacing(true);
bc.setShowStartStop(true);

final result = bc.draw('A12345B', 400, 100);
```

ITF — 段ボール箱のインジケーター

```
final bc = ITF();
bc.setShowText(true);
bc.setTextEvenSpacing(true);

final result = bc.draw('123456', 400, 100);
```

入力可能: 数字のみ（0-9）。偶数桁 が必要です。

Matrix 2of5 / NEC 2of5 — 工業用途の数字専用バーコード

```
final m25 = Matrix2of5();
m25.setShowText(true);
m25.setTextEvenSpacing(true);
final result1 = m25.draw('1234567890', 400, 100);

final n25 = NEC2of5();
n25.setShowText(true);
n25.setTextEvenSpacing(true);
final result2 = n25.draw('1234567890', 400, 100);
```

6.2 2次元バーコード — 大容量データを小さな面積に

QRコード — 日本発、世界で最も使われている2Dコード

URL、テキスト、連絡先——なんでも格納できる万能選手です。日本語もそのままエンコードできます。

```
final qr = QR();
qr.setStringEncoding('UTF-8');
qr.setErrorCorrectionLevel('M');    // L(7%) / M(15%) / Q(25%) / H(30%)
qr.setVersion(0);                   // 0=

final result = qr.draw('https://www.pao.ac/', 300);
```

誤り訂正レベル	復元能力	こんなときに
'L'	約7%	データ量を最優先したい
'M'	約15%	一般的な用途（おすすめ）
'Q'	約25%	やや過酷な環境（汚れ・傷）
'H'	約30%	ロゴを重ねたい場合にも

DataMatrix — 極小マーキングの世界標準

電子部品やヘルスケア製品の超小型マーキングに使われています。小さくても大容量。

```
final dm = DataMatrix();
dm.setStringEncoding('UTF-8');
dm.setCodeSize('AUTO');
dm.setEncodeScheme('AUTO');

final result = dm.draw('Hello World', 200);
```

PDF417 — 運転免許証にも使われている大容量コード

1次元バーコードを積み重ねたような構造で、テキスト・数字・バイナリの大量データを格納できます。

```
final pdf = PDF417();
pdf.setStringEncoding('UTF-8');
pdf.setErrorLevel(3);
pdf.setColumns(4);
pdf.setAspectRatio(3.0);
pdf.setYHeight(3);

final result = pdf.draw('PDF417 Sample', 400);
```

6.3 GS1系バーコード — 流通のインフラ

GS1-128 — AI（アプリケーション識別子）で情報を構造化

ロット番号、有効期限、重量——さまざまな情報をAIコードで構造化して格納します。

```
final bc = GS1_128();
bc.setShowText(true);
bc.setTextEvenSpacing(true);

final result = bc.draw(
  '{FNC1}0100012345678905{AI}10ABC123', 500, 120
);
```

特殊文字	意味
{FNC1}	ファンクション1（可変長フィールドの区切り）
{AI}	AI括弧表示（テキストでAIを括弧で囲んで表示）

コンビニバーコード（標準料金代理収納）

公共料金の払込票に印字されているあのバーコードです。専用メソッド drawConvenience() で生成します。

```
final bc = GS1_128();
bc.setShowText(true);

final result = bc.drawConvenience(
  '{FNC1}9191234500000000000000452087500401310029500', 500, 150
);
```

GS1 DataBar 標準型 — 青果・精肉売り場で活躍

```
final bc = GS1DataBar14();
bc.setShowText(true);
bc.setSymbolType('NORMAL');          //
// bc.setSymbolType('STACKED');          //
// bc.setSymbolType('STACKED_OMNI');     //

final result = bc.draw('4912345678904', 300, 100);
```

GS1 DataBar 限定型 — 小型商品向けコンパクト版

```
final bc = GS1DataBarLimited();
bc.setShowText(true);

final result = bc.draw('0123456789012', 200, 60);
```

入力: 13桁の数字。先頭桁は 0 または 1 のみ。

GS1 DataBar 拡張型 — クーポンや特売情報も格納可能

```
final bc = GS1DataBarExpanded();
bc.setShowText(true);

//
bc.setSymbolType('NORMAL');
final result1 = bc.draw('0100012345678905{AI}10ABC123', 400, 80);

//
bc.setSymbolType('STACKED');
bc.setNoOfColumns(4);
final result2 = bc.draw('0100012345678905{AI}10ABC123', 300, 150);
```

6.4 商品・郵便バーコード — 身の回りのバーコード

郵便カスタマバーコード — 郵便物を高速仕分け

長さの異なる4種類のバー（ロング・セミアッパー・セミロウワー・タイミング）で住所情報を表現します。

幅はバーの本数から自動計算されるため、高さだけを指定します。

```
final yubin = YubinCustomer();

//
final result = yubin.draw('264-0025-1-2-503', 60);

//
final result2 = yubin.drawWithWidth('264-0025-1-2-503', 400, 60);
```

入力形式: 郵便番号7桁 + 住所表示番号（ハイフン区切り可）

JAN/EAN バーコード — スーパーのレジで毎日活躍

```
// JAN-13
final jan13 = Jan13();
jan13.setShowText(true);
jan13.setExtendedGuard(true);        //
jan13.setTextEvenSpacing(false);     //

final result = jan13.draw('490123456789', 300, 100);

// JAN-8
final jan8 = Jan8();
jan8.setShowText(true);
jan8.setExtendedGuard(true);
jan8.setTextEvenSpacing(false);

final result2 = jan8.draw('4901234', 200, 100);
```

チェックディジットは自動計算されるため、JAN-13なら12桁、JAN-8なら7桁を入力すればOKです。

> ヒント: JAN/UPCバーコードでは setExtendedGuard() と setTextEvenSpacing() の組み合わせで見た目が変わります。商品バーコードらしい標準的な見た目にするには、extendedGuard=true + textEvenSpacing=false の組み合わせがおすすめです。

UPC バーコード — 北米の商品コード

```
// UPC-A12
final upcA = UPC_A();
upcA.setShowText(true);
upcA.setExtendedGuard(true);
upcA.setTextEvenSpacing(false);

final result = upcA.draw('01234567890', 300, 100);

// UPC-E8
final upcE = UPC_E();
upcE.setShowText(true);
upcE.setExtendedGuard(true);
upcE.setTextEvenSpacing(false);

final result2 = upcE.draw('0123456', 200, 100);
```

ここからは、全メソッドの詳細なリファレンスです。

各メソッドのパラメータ、戻り値、デフォルト値を網羅しています。

7.1 共通メソッド（全バーコード）

すべてのバーコードクラスで使用できるメソッドです。

setOutputFormat(String format)

出力形式を設定します。

パラメータ	型	説明
format	String	'png', 'jpg', 'gif', 'svg'

```
bc.setOutputFormat('png');    // PNGBase64—
bc.setOutputFormat('svg');    // SVG
```

デフォルト: 'png'

setForegroundColor(int r, int g, int b, [int a = 255])

前景色（バーの色）を設定します。

パラメータ	型	説明
r	int	赤（0〜255）
g	int	緑（0〜255）
b	int	青（0〜255）
a	int	透明度（0=透明 〜 255=不透明）

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

setBackgroundColor(int r, int g, int b, [int a = 255])

背景色を設定します。

```
bc.setBackgroundColor(255, 255, 255, 255);    //
bc.setBackgroundColor(0, 0, 0, 0);            //
```

描画メソッド

引数の形式はバーコードの種類によって異なります：

カテゴリ	draw
1次元	draw(String code, int width, int height) → String
2次元	draw(String code, int size) → String
郵便	draw(String code, int height) → String

戻り値:

- PNG/JPEG/GIF: "..." 形式の Data URI 文字列
- SVG: "<svg xmlns=...>...</svg>" 形式の文字列
- エラー時: 空文字列 ""

FFI版の追加メソッド

メソッド	戻り値	説明
drawBytes(...)	Uint8List	生画像バイト列
drawRect(...)	String	長方形出力（2Dバーコード）
getBase64()	String	Base64 Data URI 文字列
getSvg()	String	SVG文字列
getImageData()	Uint8List	生画像バイト列
isSvgOutput()	bool	現在のフォーマットがSVGか
dispose()	—	ネイティブリソースを解放。必ず呼ぶこと

> なぜ dispose() が必要？
FFI版では、バーコードオブジェクトがC++のメモリを保持しています。Dartのガベージコレクション対象外なので、dispose() を呼ばないとメモリリークの原因になります。try-finally パターンを推奨します。

7.2 1次元バーコード共通メソッド

1次元バーコード（郵便カスタマバーコードを除く）で共通して使用できるメソッドです。

setShowText(bool show)

バーコード下部のテキスト表示を切り替えます。

パラメータ	型	説明
show	bool	true: 表示 / false: 非表示

デフォルト: true

setTextFontScale(double scale)

テキストのフォントサイズ倍率を設定します。

パラメータ	型	説明
scale	double	倍率（0.5〜3.0 推奨）

デフォルト: 1.0

setTextGap(double gap)

バーコードとテキストの間隔を調整します。

パラメータ	型	説明
gap	double	倍率（0.0〜3.0 推奨）

デフォルト: 1.0

setTextEvenSpacing(bool even)

テキストの均等割付を設定します。

パラメータ	型	説明
even	bool	true: 均等割付 / false: 中央寄せ

デフォルト:

- JAN/UPC以外: true（均等割付）
- JAN-8, JAN-13, UPC-A, UPC-E: false（セクション間配置）

> 使い分けのコツ: 一般的な1Dバーコード（Code39, Code128など）では

true（均等割付）にすると、各文字がバーの真下に揃って読みやすくなります。一方、JAN/UPCバーコードでは　false にして setExtendedGuard(true) と組み合わせるのが、商品バーコードとしての標準的な見た目です。

setFitWidth(bool fit)

指定した幅にぴったり収めるかどうかを設定します。

デフォルト: false

setPxAdjustBlack(int px) / setPxAdjustWhite(int px)

バー幅の微調整です。印刷時のにじみ補正に使います。

デフォルト: 0

7.3 Code39

クラス: Code39 — 工業用途の定番バーコード

入力可能文字: 0-9, A-Z, - . $ / + %, スペース

固有メソッド

setShowStartStop(bool show)

テキスト表示時にスタート/ストップコード（*）を表示するかどうか。

デフォルト: false

7.4 Code93

クラス: Code93 — Code39の高密度版

入力可能文字: ASCII全文字（0x00〜0x7F）

固有メソッド: なし

7.5 Code128

クラス: Code128 — 物流の標準バーコード

入力可能文字: ASCII全文字（0x00〜0x7F）。制御文字は {CR}, {LF}, {TAB}, {FNC1} 等で入力。

固有メソッド

setCodeMode(String mode)

モード	対応文字
'AUTO'	自動で最短幅に最適化（おすすめ）
'A'	制御文字 + 数字 + 英大文字 + 一部記号
'B'	数字 + 英大文字 + 英小文字 + 記号
'C'	数字のみ（2桁ずつ高密度エンコード）

デフォルト: 'AUTO'

7.6 GS1-128

クラス: GS1_128 — GS1標準準拠。物流・医療分野で使用

特殊文字	意味
{FNC1}	ファンクション1（可変長AIの区切り）
{AI}	AI括弧表示（テキスト表示時にAIを括弧で囲む）

固有メソッド

drawConvenience(String code, int width, int height)

標準料金代理収納用（コンビニバーコード）を生成します。

コンビニバーコードのデータ形式:

位置	桁数	内容
1-2	2	AI: 91 固定
3-8	6	企業コード
9-43	35	収納データ（金額・期限等）
44	1	チェックディジット（モジュラス10ウエイト3、自動計算

7.7 NW-7（Codabar）

クラス: NW7 — 宅配便・図書館で使用

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

固有メソッド

setShowStartStop(bool show)

デフォルト: false

7.8 ITF

クラス: ITF — 段ボール箱の物流管理バーコード

入力可能文字: 0-9 のみ。偶数桁 が必要。

固有メソッド: なし

7.9 Matrix 2of5

クラス: Matrix2of5 — 工業用の数字専用バーコード

入力可能文字: 0-9 のみ

固有メソッド: なし

7.10 NEC 2of5

クラス: NEC2of5 — 日本の工業用途向け

入力可能文字: 0-9 のみ

固有メソッド: なし

7.11 JAN-8（EAN-8）

クラス: Jan8 — 小型商品用の8桁バーコード

入力: 数字7桁（チェックディジットは自動計算）

固有メソッド

setExtendedGuard(bool extended)

ガードバー（先頭・中央・末尾の区切りバー）を拡張するかどうかを設定します。

true
にすると、ガードバーがテキスト領域まで長く伸び、テキストは左右のセクションに分かれて配置されます。商品バーコードとしての標準的な外観になります。

false にすると、全バーが同じ高さのフラットな外観になります。

デフォルト: true

テキスト表示パターン

JAN/UPC バーコードでは、setExtendedGuard() と setTextEvenSpacing() の組み合わせで、4種類の見た目を選べます。

extendedGuard	textEvenSpacing	見た目
true	false	商品バーコードの標準スタイル。 ガードバーが長く伸び、
true	true	ガードバーが長く伸び、テキストは均等割付
false	false	フラットバー + テキスト中央寄せ
false	true	フラットバー + テキスト均等割付

7.12 JAN-13（EAN-13）

クラス: Jan13 — 日本の標準的な商品バーコード（13桁）

入力: 数字12桁（チェックディジットは自動計算）

固有メソッド

setExtendedGuard(bool extended)

7.11 JAN-8 と同じです。

デフォルト: true

> JAN-13の特徴:
拡張ガードバー有効時、先頭1桁がバーコード左側にプレフィックスとして表示されます。日本の商品バーコードの「49」や「45」で始まるおなじみの見た目です。

7.13 UPC-A

クラス: UPC_A — 北米の商品コード（12桁）

入力: 数字11桁（チェックディジットは自動計算）

固有メソッド

setExtendedGuard(bool extended)

7.11 JAN-8 と同じです。

デフォルト: true

> UPC-Aの特徴:
拡張ガードバー有効時、先頭1桁（ナンバーシステム）が左側に、末尾1桁（チェックディジット）が右側に、それぞれ小さく表示されます。

7.14 UPC-E

クラス: UPC_E — UPC-Aの短縮版（8桁）。小型商品用

入力: 数字7桁（チェックディジットは自動計算）。UPC-Aのゼロ圧縮形式。

固有メソッド

setExtendedGuard(bool extended)

7.11 JAN-8 と同じです。

デフォルト: true

7.15 GS1 DataBar 標準型

クラス: GS1DataBar14 — 生鮮食品向けのコンパクトバーコード

入力: 数字 8〜13桁（チェックディジットは自動計算）

固有メソッド

setSymbolType(String type)

値	説明
'NORMAL'	標準型（どの方向からでも読み取り可能）
'STACKED'	二層型（省スペース）
'STACKED_OMNI'	標準二層型

デフォルト: 'NORMAL'

FFI版の追加メソッド

メソッド	説明
getSymbolType()	現在のシンボルタイプを取得
encode(String content)	事前エンコード。成功で true
static calculateCheckDigit(String src)	チェックディジット計算

7.16 GS1 DataBar 限定型

クラス: GS1DataBarLimited — 先頭桁が0または1に限定されたコンパクト版

入力: 数字 13桁（先頭桁は0または1のみ）

固有メソッド: なし

7.17 GS1 DataBar 拡張型

クラス: GS1DataBarExpanded — 可変長データ対応

入力: AI + データの組み合わせ（{FNC1}, {AI} 使用可能）

固有メソッド

setSymbolType(String type)

値	説明
'NORMAL'	一層型
'STACKED'	多層型

デフォルト: 'NORMAL'

setNoOfColumns(int columns)

多層型のセグメント数（列数）。偶数推奨。 デフォルト: 2

FFI版の追加メソッド

drawStacked(String code, int width, int height)

Stacked専用の描画メソッドです。

7.18 郵便カスタマバーコード

クラス: YubinCustomer — 日本郵便の住所バーコード

入力: 郵便番号7桁 + 住所表示番号（ハイフン区切り可）

固有メソッド

draw(String code, int height)

他のバーコードと異なり、幅は自動計算されるため高さのみ指定します。

drawWithWidth(String code, int width, int height)

幅も指定したい場合はこちらを使います。

> 注意: テキスト関連メソッド（setShowText(), setTextFontScale(), setTextEvenSpacing()等）は使用できません。setForegroundColor(), setBackgroundColor(), setPxAdjustBlack(), setPxAdjustWhite()は使用可能です。

入力例

入力データ	意味
'264-0025-1-2-503'	〒264-0025 1丁目2番地503号
'1050001-13-6'	〒105-0001 13番6号
'2750026-3-29-2-401'	〒275-0026 3丁目29番2号401

7.19 QRコード

クラス: QR — 日本発、世界で最も普及している2次元バーコード

入力: 数字、英数字、バイナリ、漢字（Shift-JIS）

固有メソッド

draw(String code, int size)

パラメータ	型	説明
code	String	エンコードするデータ
size	int	画像サイズ（px、正方形）

setStringEncoding(String encoding)

'UTF-8'（デフォルト）または 'Shift_JIS'

setErrorCorrectionLevel(String level)

レベル	復元能力	こんなときに
'L'	約7%	データ量優先
'M'	約15%	一般的な用途（おすすめ）
'Q'	約25%	汚れ・傷への耐性が必要
'H'	約30%	最高品質。ロゴ重ね時にも

デフォルト: 'M'

setVersion(int version)

0（自動）〜 40。 デフォルト: 0

setEncodeMode(String mode)

値	説明
'AUTO'	自動選択
'NUMERIC'	数字のみ（最高効率）
'ALPHANUMERIC'	英数字
'BYTE'	バイトデータ
'KANJI'	漢字（Shift-JIS）

デフォルト: 'AUTO'

setFitWidth(bool fit)

デフォルト: false

QRコードバージョンと最大容量（誤り訂正レベルM）

バージョン	モジュール数	数字	英数字	バイト
1	21x21	34	20	14
5	37x37	202	122	84
10	57x57	652	395	271
20	97x97	2,061	1,249	858
40	177x177	7,089	4,296	2,953

7.20 DataMatrix

クラス: DataMatrix — 超小型マーキングの世界標準

入力: ASCII文字、バイナリデータ、GS1データ（{FNC1} で開始）

固有メソッド

draw(String code, int size)

setStringEncoding(String encoding)

'UTF-8'（デフォルト）または 'Shift_JIS'

setCodeSize(String size)

主な値	説明
'AUTO'	自動（おすすめ）
'10x10' ～ '144x144'	正方形
'8x18', '8x32' 等	矩形

デフォルト: 'AUTO'

setEncodeScheme(String scheme)

値	説明
'AUTO'	自動選択（おすすめ）
'ASCII'	ASCII
'C40'	英数字
'TEXT'	テキスト（小文字優先）
'X12'	ANSI X12 EDI
'EDIFACT'	EDIFACT
'BASE256'	バイナリ

デフォルト: 'AUTO'

setFitWidth(bool fit)

デフォルト: false

GS1-DataMatrix

GS1データを格納する場合は、先頭に {FNC1} を付けます。

```
dm.draw('{FNC1}0100012345678905{FNC1}10ABC123', 200);
```

7.21 PDF417

クラス: PDF417 — 大容量2次元バーコード。運転免許証・搭乗券に使用

入力: テキスト、数字、バイナリ

固有メソッド

draw(String code, int width)

パラメータ	型	説明
code	String	エンコードするデータ
width	int	画像の幅（px）。高さは自動計算

setStringEncoding(String encoding)

'UTF-8'（デフォルト）または 'Shift_JIS'

setErrorLevel(int level)

レベル	訂正能力
-1	自動（おすすめ）
0	最小
1〜2	標準
3〜8	高〜最大

デフォルト: -1（自動）

setColumns(int columns)

列数。0=自動、1〜30で指定。 デフォルト: 0

setRows(int rows)

行数。0=自動、3〜90で指定。 デフォルト: 0

setAspectRatio(double ratio)

縦横比。1.0〜10.0。 デフォルト: 3.0

setYHeight(int height)

Y方向の高さ係数。1〜10。 デフォルト: 3

setFitWidth(bool fit)

デフォルト: false

Barcode.Flutter C++ Import版には WASM版 と FFI版 の2つのパッケージがあります。

バージョン1.1からWASM版が全プラットフォームに対応したため、基本的に WASM版だけでOK です。

比較表

項目	WASM版（おすすめ）	FFI版
パッケージ名	barcode_pao_wasm	barcode_pao
対応プラットフォーム	全プラットフォーム（Web + ネイティブ）	ネイティブのみ（Windows / macOS / Linux / iOS / And
Web での技術	WebAssembly + JS interop	—
ネイティブでの技術	dart:ffi（barcode_pao 経由）	dart:ffi + C++ DLL/SO
初期化	Web: await initBarcodeWasm() が必要 ネイ	不要
dispose()	Web: 不要 ネイティブ: 不要（内部で管理）	必須（メモリリーク防止）
drawBytes()	なし	あり（Uint8List 直接取得）
drawRect()	なし	あり（2D長方形出力）
速度	Web: 高速（最速クラス） ネイティブ: FFI版	最高速（C++ 直接実行）

選び方

迷ったらWASM版一択です。

- WASM版（barcode_pao_wasm） — Web でもネイティブでも同じコードで動く。これ1つで全プラットフォーム対応
- FFI版（barcode_pao） — drawBytes() や drawRect() が必要な場合のみ

```
// WASM —
import 'package:barcode_pao_wasm/barcode_pao_wasm.dart';

// FFIdrawBytes()
import 'package:barcode_pao/barcode_pao.dart';
```

> 技術ノート: WASM版は内部で Conditional Import を使い、プラットフォームに応じて自動的にバックエンドを切り替えます。Webでは WebAssembly + JS interop、ネイティブでは barcode_pao プラグインの DLL を dart:ffi で呼び出します。ユーザーが意識する必要はなく、import 文1つで全プラットフォームで動作します。

WASM版の初期化（Flutter Web のみ）

Flutter Web で使う場合は、アプリ起動時に 1回だけ initBarcodeWasm() を呼ぶ必要があります。ネイティブプラットフォームでは初期化不要です：

```
import 'package:flutter/foundation.dart' show kIsWeb;

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  if (kIsWeb) {
    await initBarcodeWasm();  // Web
  }
  runApp(const MyApp());
}
```

FFI版のリソース管理

FFI版を直接使う場合は、C++のメモリを保持しているため、使い終わったら 必ず dispose() を呼んでください：

```
final bc = Code128();
try {
  final result = bc.draw('HELLO', 400, 100);
  // ... result  ...
} finally {
  bc.dispose();
}
```

> ヒント: WASM版経由でネイティブ実行する場合、dispose() は内部で管理されるため呼ぶ必要はありません。

WASM版 (barcode_pao_wasm) — おすすめ

項目	要件
Flutter	3.0 以上
Dart	3.0 以上
プラットフォーム	全プラットフォーム（Web + Windows + macOS + Linux + iOS + Android）
Web ブラウザ	WebAssembly 対応ブラウザ（Chrome, Firefox, Safari, Edge）
ネイティブ追加依存	barcode_pao プラグイン（DLL/SO 提供用）

FFI版 (barcode_pao)

項目	要件
Flutter	3.0 以上
Dart	3.0 以上
プラットフォーム	Windows / macOS / Linux / iOS / Android
ネイティブライブラリ	プラットフォーム毎のビルド済みバイナリが必要

プラットフォームとパッケージの対応

プラットフォーム	WASM版	FFI版
Flutter Web	対応	非対応
Windows	対応	対応
macOS	対応	対応
Linux	対応	対応
iOS	対応	対応
Android	対応	対応

> 注意: WASM版をネイティブプラットフォームで使用する場合、内部的に barcode_pao プラグインの C++ ネイティブライブラリを使用します。そのため、pubspec.yaml に barcode_pao も追加してください（3.1節参照）。

使用許諾

Barcode.Flutter
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 Barcode.Flutter 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 Barcode.Flutter 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

Barcode.Flutter 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

Barcode.Flutter
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

Barcode.Flutter
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 Barcode.Flutter 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 Barcode.Flutter を使用した日より有効です。

ライセンス

Barcode.Flutter C++ Import版は有限会社パオ・アット・オフィスの製品です。

項目	内容
1開発ライセンス	11,000円（税込） / 10,000円（税抜）
必要ライセンス数	Barcode.Flutter を利用して開発するパソコンの台数分
ランタイムライセンス	無償（開発環境にのみライセンスが必要）
パッケージ共通	1ライセンスでWASM版・FFI版の両方を利用可能

試用版:
生成されるバーコードに「SAMPLE」の透かしが表示されます。機能制限はありません。すべてのバーコード種類・設定を自由にお試しいただけます。

製品版: 透かしなしでバーコードを生成できます。

お問い合わせ

有限会社 パオ・アット・オフィス

Webサイト	https://www.pao.ac/
製品ページ	https://www.pao.ac/barcode.flutter/
メール	info@pao.ac

保守・保証につきましては、保守・保証に関する規定をご覧ください。

関連製品

製品	対応環境
Barcode.Flutter Pure Dart版	Flutter / Dart（CanvasRenderer / PdfRenderer 連携）
Barcode.Python Pure Python版（ソースコード付）	Python（ReportLab / Pillow 連携）
Barcode.Python C++ Import版	Python（Native / WASM）
Barcode.wasm	JavaScript / TypeScript
Barcode.net	.NET（C#, VB.NET）
Barcode.jar	Java
Barcode.php	PHP

Barcode.Flutter C++ Import版 ユーザーズマニュアル

バージョン 1.1 — 2026年2月