Barcode.Swift

Swift バーコード生成ライブラリ

マニュアル

バージョン 1.0

有限会社 パオ・アット・オフィス

https://www.pao.ac/

目次

サーバーサイドで、18種のバーコードを自在に生成。

ユーザーズマニュアル

バージョン 1.0 — 2026年2月

有限会社 パオ・アット・オフィス

https://www.pao.ac/

はじめに

Barcode.Swiftとは

Barcode.Swift は、Swift で書かれた Pure Swift バーコード生成ライブラリです。C++やその他の外部依存を必要とせず、Swift Package Manager で導入するだけですぐに18種類のバーコードを生成できます。

サーバーサイド Swift（Vapor など）でバーコード画像を動的に生成し、Web アプリケーションや帳票出力に活用できます。

```
import BarcodePao

let qr = QRCode(outputFormat: BarcodeBase.FORMAT_PNG)
try qr.draw(code: "https://www.pao.ac/", size: 300)
let base64 = try qr.getImageBase64()
// → "..."  <img src="...">
```

特長

特長	説明
Pure Swift	C/C++ 依存なし。Swift Package Manager だけで完結
18種対応	1D/2D バーコード 18種をすべてサポート
3形式出力	PNG、JPEG、SVG いずれの形式でも出力可能
カスタマイズ	色、テキスト表示、均等配置など細かく制御
サーバー対応	Vapor、Hummingbird など主要フレームワークに対応
クロスプラットフォーム	macOS / Linux で動作

対応バーコード一覧

カテゴリ	バーコード
1D 工業用	Code39, Code93, Code128, NW-7 (Codabar)
1D 物流用	ITF, Matrix 2of5, NEC 2of5
GS1 系	GS1-128, GS1 DataBar 標準型/限定型/拡張型
商品コード	JAN-8 (EAN-8), JAN-13 (EAN-13), UPC-A, UPC-E
郵便	郵便カスタマバーコード
2D	QRコード, DataMatrix, PDF417

できること

PNG画像出力 — Base64でそのまま返せる

PNG 画像を Base64 エンコードした文字列（Data URI）で取得できます。HTML の <img> タグにそのまま埋め込むことが可能です。

```
let bc = Code128(outputFormat: BarcodeBase.FORMAT_PNG)
bc.showText = true
try bc.draw(code: "Hello-2026", width: 400, height: 100)
let dataUri = try bc.getImageBase64()
// → "..."
```

SVGベクター出力 — 拡大しても美しい

SVG 形式で出力すればベクターなので、拡大しても劣化しません。

```
let bc = Code39(outputFormat: BarcodeBase.FORMAT_SVG)
try bc.draw(code: "HELLO", width: 300, height: 80)
let svg = try bc.getSVG()
// → "<svg xmlns=..."
```

バイト列出力 — ファイル保存やHTTPレスポンスに

getImageMemory() でバイト列 [UInt8] を取得し、ファイルに保存したり HTTP レスポンスとして直接返したりできます。

```
let qr = QRCode(outputFormat: BarcodeBase.FORMAT_PNG)
try qr.draw(code: "Test", size: 200)
let bytes = try qr.getImageMemory()
try Data(bytes).write(to: URL(fileURLWithPath: "qr.png"))
```

カスタマイズ — 色もテキストも思いのままに

前景色・背景色の変更、テキスト表示のON/OFF、テキストの均等配置など、さまざまなカスタマイズが可能です。

```
let bc = Code128(outputFormat: BarcodeBase.FORMAT_PNG)
bc.showText = true
bc.textEvenSpacing = true
bc.setForegroundColor(0, 0, 128, 255)     //
bc.setBackgroundColor(255, 255, 240, 255)  //
try bc.draw(code: "Custom", width: 400, height: 120)
```

導入方法

ダウンロード

製品サイトからサンプルプロジェクトを含む ZIP ファイルをダウンロードしてください。

https://www.pao.ac/barcode.swift/

Swift Package Manager でインストール

Package.swift の dependencies に追加します：

```
dependencies: [
    .package(url: "https://github.com/pao-company/barcode-swift.git", from: "1.0.0"),
]
```

ターゲットの dependencies にも追加します：

```
.target(name: "YourApp", dependencies: [
    .product(name: "BarcodePao", package: "barcode-swift"),
]),
```

ファイル構成

```
barcode-pao/
├── Package.swift
├── Roboto-Regular.ttf        ←
└── Sources/BarcodePao/
    ├── BarcodeBase.swift      ←
    ├── BarcodeBase1D.swift    ← 1D
    ├── BarcodeBase2D.swift    ← 2D
    ├── RenderPNG.swift        ← PNG/JPEG
    ├── RenderSVG.swift        ← SVG
    ├── Font.swift             ← TTF
    ├── Code39.swift           ← Code39
    ├── QR.swift               ← QR
    └── ... (18)
```

クイックスタート — 最初の1本を生成しよう

QRコードをPNGで生成

```
import BarcodePao

// 1:
let qr = QRCode(outputFormat: BarcodeBase.FORMAT_PNG)

// 2:  →
try qr.draw(code: "https://www.pao.ac/", size: 300)
let base64 = try qr.getImageBase64()

// HTML data URI
print("<img src=\"\(base64)\">")
```

1Dバーコードを SVGで生成

```
import BarcodePao

let c128 = Code128(outputFormat: BarcodeBase.FORMAT_SVG)
c128.showText = true
try c128.draw(code: "SWIFT-2026", width: 400, height: 100)
let svg = try c128.getSVG()
```

REST APIサーバーで提供

Vapor を使って QR コードを返す簡単なエンドポイント：

```
import Vapor
import BarcodePao

app.get("qr") { req -> Response in
    let code = req.query[String.self, at: "code"] ?? "Hello"
    let qr = QRCode(outputFormat: BarcodeBase.FORMAT_PNG)
    try qr.draw(code: code, size: 300)
    let bytes = try qr.getImageMemory()
    var headers = HTTPHeaders()
    headers.add(name: .contentType, value: "image/png")
    return Response(status: .ok, headers: headers,
                    body: .init(data: Data(bytes)))
}
```

実践サンプル集

1次元バーコード — 物流・工業の定番

```
// Code39 —  +
let c39 = Code39(outputFormat: BarcodeBase.FORMAT_PNG)
c39.showText = true
try c39.draw(code: "HELLO-123", width: 400, height: 100)

// Code93 — Code39
let c93 = Code93(outputFormat: BarcodeBase.FORMAT_PNG)
try c93.draw(code: "TEST-93", width: 400, height: 100)

// Code128 — ASCII
let c128 = Code128(outputFormat: BarcodeBase.FORMAT_PNG)
c128.showText = true
try c128.draw(code: "Hello-2026", width: 400, height: 100)

// NW-7 (Codabar)
let nw7 = NW7(outputFormat: BarcodeBase.FORMAT_PNG)
try nw7.draw(code: "A12345B", width: 400, height: 100)

// ITF (Interleaved 2 of 5) —
let itf = ITF(outputFormat: BarcodeBase.FORMAT_PNG)
try itf.draw(code: "1234567890", width: 400, height: 100)
```

2次元バーコード — 大容量データを小さな面積に

```
// QR
let qr = QRCode(outputFormat: BarcodeBase.FORMAT_PNG)
qr.errorCorrectionLevel = QR_ECC_H  //
try qr.draw(code: "https://www.pao.ac/", size: 300)

// DataMatrix —
let dm = DataMatrix(outputFormat: BarcodeBase.FORMAT_PNG)
try dm.draw(code: "Hello DataMatrix", size: 300)

// PDF417 —
let pdf = PDF417(outputFormat: BarcodeBase.FORMAT_PNG)
try pdf.draw(code: "Hello PDF417", width: 400, height: 200)
```

GS1系バーコード — 流通のインフラ

```
// GS1-128 — AI
let gs1 = GS1_128(outputFormat: BarcodeBase.FORMAT_PNG)
try gs1.draw(code: "(01)04912345123459(10)ABC123", width: 500, height: 100)

// GS1 DataBar
let db14 = GS1DataBar14(outputFormat: BarcodeBase.FORMAT_PNG)
try db14.draw(code: "0112345678901231", width: 400, height: 80)

// GS1 DataBar
let dbl = GS1DataBarLimited(outputFormat: BarcodeBase.FORMAT_PNG)
try dbl.draw(code: "0100012345678905", width: 400, height: 80)

// GS1 DataBar
let dbe = GS1DataBarExpanded(outputFormat: BarcodeBase.FORMAT_PNG)
try dbe.draw(code: "(01)00012345678905(10)ABC123", width: 500, height: 80)
```

商品・郵便バーコード — 身の回りのバーコード

```
// JAN-8 (EAN-8)
let j8 = JAN8(outputFormat: BarcodeBase.FORMAT_PNG)
j8.showText = true
try j8.draw(code: "1234567", width: 200, height: 100)

// JAN-13 (EAN-13) — POS
let j13 = JAN13(outputFormat: BarcodeBase.FORMAT_PNG)
j13.showText = true
try j13.draw(code: "490123456789", width: 300, height: 120)

// UPC-A —
let upca = UPC_A(outputFormat: BarcodeBase.FORMAT_PNG)
upca.showText = true
try upca.draw(code: "01234567890", width: 300, height: 120)

//  —
let yubin = YubinCustomer(outputFormat: BarcodeBase.FORMAT_PNG)
try yubin.draw(code: "10200091-13-2-3", height: 50)
```

APIリファレンス

共通メソッド（全バーコード）

メソッド	説明
init(outputFormat: String)	バーコードオブジェクトを作成
getImageBase64() throws -> String	Base64 Data URI を返す（PNG/JPEG）
getSVG() throws -> String	SVG 文字列を返す
getImageMemory() throws -> [UInt8]	バイト列を返す（PNG/JPEG）
setForegroundColor(_ r: Int, _ g: Int, _ b: Int, _ a: Int)	前景色を設定
setBackgroundColor(_ r: Int, _ g: Int, _ b: Int, _ a: Int)	背景色を設定

出力フォーマット定数:

定数	値
BarcodeBase.FORMAT_PNG	"png"
BarcodeBase.FORMAT_JPEG	"jpeg"
BarcodeBase.FORMAT_SVG	"svg"

1次元バーコード共通プロパティ

プロパティ	型	説明
showText	Bool	テキスト表示 ON/OFF（デフォルト: false）
textEvenSpacing	Bool	テキストの均等配置（デフォルト: false）

Code39

```
let bc = Code39(outputFormat: format)
bc.showStartStop = true  // /
try bc.draw(code: "HELLO-123", width: 400, height: 100)
```

使用可能文字: 0-9, A-Z, -, ., , $, /, +, %

Code93

```
let bc = Code93(outputFormat: format)
try bc.draw(code: "TEST-93", width: 400, height: 100)
```

Code128

```
let bc = Code128(outputFormat: format)
bc.showText = true
try bc.draw(code: "Hello-2026", width: 400, height: 100)
```

ASCII 全域（0x00-0x7F）に対応。

GS1-128

```
let bc = GS1_128(outputFormat: format)
try bc.draw(code: "(01)04912345123459(10)ABC123", width: 500, height: 100)
```

AI（Application Identifier）を括弧で囲んで指定。

NW-7（Codabar）

```
let bc = NW7(outputFormat: format)
try bc.draw(code: "A12345B", width: 400, height: 100)
```

スタート/ストップ文字: A, B, C, D

ITF（Interleaved 2 of 5）

```
let bc = ITF(outputFormat: format)
try bc.draw(code: "1234567890", width: 400, height: 100)
```

数字のみ、偶数桁。

Matrix 2of5

```
let bc = Matrix2of5(outputFormat: format)
try bc.draw(code: "12345", width: 400, height: 100)
```

NEC 2of5

```
let bc = NEC2of5(outputFormat: format)
try bc.draw(code: "12345", width: 400, height: 100)
```

JAN-8（EAN-8）

```
let bc = JAN8(outputFormat: format)
bc.showText = true
try bc.draw(code: "1234567", width: 200, height: 100)
```

7桁 + チェックデジット自動計算。

JAN-13（EAN-13）

```
let bc = JAN13(outputFormat: format)
bc.showText = true
try bc.draw(code: "490123456789", width: 300, height: 120)
```

12桁 + チェックデジット自動計算。

UPC-A

```
let bc = UPC_A(outputFormat: format)
bc.showText = true
try bc.draw(code: "01234567890", width: 300, height: 120)
```

UPC-E

```
let bc = UPC_E(outputFormat: format)
bc.showText = true
try bc.draw(code: "0123456", width: 200, height: 100)
```

GS1 DataBar 標準型

```
let bc = GS1DataBar14(outputFormat: format)
try bc.draw(code: "0112345678901231", width: 400, height: 80)
```

GS1 DataBar 限定型

```
let bc = GS1DataBarLimited(outputFormat: format)
try bc.draw(code: "0100012345678905", width: 400, height: 80)
```

GS1 DataBar 拡張型

```
let bc = GS1DataBarExpanded(outputFormat: format)
try bc.draw(code: "(01)00012345678905(10)ABC123", width: 500, height: 80)
```

郵便カスタマバーコード

```
let bc = YubinCustomer(outputFormat: format)
try bc.draw(code: "10200091-13-2-3", height: 50)
```

幅は自動計算されます。

QRコード

```
let qr = QRCode(outputFormat: format)
qr.errorCorrectionLevel = QR_ECC_M  // L, M, Q, H
qr.version = 0       // 0 = , 1-40
try qr.draw(code: "Hello", size: 300)
```

定数	値	誤り訂正能力
QR_ECC_L	0	約7%
QR_ECC_M	1	約15%
QR_ECC_Q	2	約25%
QR_ECC_H	3	約30%

DataMatrix

```
let dm = DataMatrix(outputFormat: format)
dm.stringEncoding = "gs1"  // GS1 DataMatrix
try dm.draw(code: "Hello DataMatrix", size: 300)
```

PDF417

```
let pdf = PDF417(outputFormat: format)
try pdf.draw(code: "Hello PDF417", width: 400, height: 200)
```

動作環境

項目	要件
Swift	5.10 以上
OS	macOS 14+, Linux (Ubuntu 22.04+, CentOS Stream 9+)
依存	swift-png (tayloraswift)

ライセンス・お問い合わせ

使用許諾

本ソフトウェアは商用ライセンスです。ご購入いただいたライセンスに基づき、お客様の開発プロジェクトでご利用いただけます。

ライセンス

プラン	価格（税込）	内容
通常ライセンス	¥22,000	1開発者、無制限プロジェクト
3年サポート	¥9,900	メールサポート + アップデート
5年サポート	¥13,750	メールサポート + アップデート

お問い合わせ:

- Web: https://www.pao.ac/
- Email: info@pao.ac