# Barcode.Rust

Rust 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/

# 1.1 Barcode.Rustとは

物流倉庫のピッキングリスト、医療現場の検体ラベル、ECサイトの出荷伝票——。

バーコードはあらゆる業務システムの「最後の1ミリ」を担っています。

Barcode.Rust は、そのバーコードを Pure Rust で生成するライブラリです。

unsafe不要、外部Cライブラリへの依存も一切不要。cargo　add　一発でインストールでき、1次元・2次元あわせて全18種 のバーコードを、PNG画像・JPEG画像・SVGベクター で出力できます。

Rust　のメモリ安全性とゼロコスト抽象化をそのまま活かせるため、axum などの非同期Webフレームワークに組み込めば、毎秒数千枚のバーコードを安全に生成することも可能です。

```
let mut qr = QR::new(FORMAT_PNG);
qr.draw("https://www.pao.ac/", 300)?;
let base64 = qr.base_2d.base.get_image_base64()?;
```

たった3行で、QRコードが Base64 文字列になって返ってきます。

# 1.2 特長

| 特長 | 説明 |
| --- | --- |
| Pure Rust | 外部依存は image + ab_glyph クレートのみ。unsafe不要、クロスコンパイルも自 |
| サーバーサイド特化 | axum と組み合わせるだけで REST API バーコードサーバーが完成 |
| PNG / JPEG / SVG | 画面表示にはPNG、印刷にはSVG、サムネイルにはJPEG。用途で使い分け可能 |
| 18種のバーコード | 1D・2D・GS1・郵便まで、業務で必要なバーコードを網羅 |
| thread-safe (Send + Sync) | 各インスタンスが独立しているため、マルチスレッド環境でもロックフリーで並行 |
| 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 |

# 1.3 対応バーコード一覧

## 1次元バーコード（11種類）

| バーコード | コンストラクタ | どんなところで使われている？ |
|---|---|---|
| Code39 | Code39::new() | 工場の部品ラベル、軍事規格（MIL-STD）にも採用 |
| Code93 | Code93::new() | Code39の高密度版。郵便・物流で活用 |
| Code128 | Code128::new() | 物流の標準。ASCII全文字をエンコード可能 |
| GS1-128 | GS1_128::new() | 医薬品・物流。ロット番号や有効期限をAIで管理 |
| NW-7 (Codabar) | NW7::new() | 宅配便の送り状、図書館の貸出管理でおなじみ |
| ITF | ITF::new() | 段ボール箱の集合包装用。数字ペアで高密度 |
| Matrix 2of5 | Matrix2of5::new() | 工業用途。数字のみのシンプルな構成 |
| NEC 2of5 | NEC2of5::new() | 日本の工業現場で使われるバリエーション |
| JAN-8 (EAN-8) | JAN8::new() | 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::new() | 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::new() | 北米の商品コード。12桁 |
| UPC-E | UPC_E::new() | 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 | コンストラクタ | どんなところで使われている？ |
|---|---|---|
| GS1 DataBar 標準型 | GS1DataBar14::new() | 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::new() | 小型商品向けのコンパクト版 |
| GS1 DataBar 拡張型 | GS1DataBarExpanded::new() | 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 | コンストラクタ | 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::new() | 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 | コンストラクタ | どんなところで使われている？ |
| --- | --- | --- |
| QRコード | QR::new() | URL、決済、名刺交換——。日本発、世界で最も普及した |
| DataMatrix | DataMatrix::new() | 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::new() | 運転免許証、搭乗券。大容量データを1本に集約 |

# 2.1 PNG画像出力 — Base64でそのまま返せる

draw() でバーコードを生成し、get_image_base64() 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。HTMLの <img> タグにそのまま埋め込めるので、REST APIの戻り値としてそのままクライアントに返せます。

```
let mut qr = QR::new(FORMAT_PNG);
qr.draw("Hello World", 300)?;
let base64 = qr.base_2d.base.get_image_base64()?;

// base64  "..."
//  JSON
```

**PNGが向いている場面:**

- REST APIでBase64文字列をクライアントに返す
- 画面上でのプレビュー表示
- 固定解像度での画像出力

# 2.2 SVGベクター出力 — 拡大しても美しい

出力形式を FORMAT_SVG にするだけで、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。

```
let mut bc = Code128::new(FORMAT_SVG);
bc.draw("Hello-2026", 400, 100)?;
let svg = bc.base_1d.base.get_svg()?;

// svg  "<svg xmlns="...">...</svg>"
// HTML
```

**SVGが向いている場面:**

- ラベル印刷（拡大しても劣化しない）
- PDF生成時の高品質バーコード埋め込み
- ファイルサイズを小さく抑えたい場合

> ヒント: 同じバーコードオブジェクトで set_output_format() 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、ダウンロードはSVG、という使い分けも簡単です。

# 2.3 バイト列出力 — ファイル保存やHTTPレスポンスに

get_image_memory() を使えば、PNG/JPEG のバイト列（Vec<u8>）を直接取得できます。ファイル保存はもちろん、axum のレスポンスに直接返してバイナリレスポンスとして返すことも可能です。

```
let mut bc = Code39::new(FORMAT_PNG);
bc.draw("HELLO", 400, 100)?;
let image_bytes = bc.base_1d.base.get_image_memory()?;

//
std::fs::write("barcode.png", &image_bytes)?;

//  axum
// (StatusCode::OK, [("Content-Type", "image/png")], image_bytes)
```

# 2.4 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.base_1d.base.set_foreground_color(0, 0, 128, 255);
bc.base_1d.base.set_background_color(255, 255, 240, 255);

//
bc.base_1d.base.set_background_color(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.base_1d.show_text = true;                          //
bc.base_1d.base.set_text_font_scale(1.2);             //
bc.base_1d.base.set_text_vertical_offset_scale(0.5);  //
bc.base_1d.text_even_spacing = true;                   // 1
```

> ヒント: text_even_spacing = 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.base_1d.base.set_px_adjust_black(-1);  // 1px
bc.base_1d.base.set_px_adjust_white(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.base_1d.base.set_fit_width(true);   //
bc.base_1d.base.set_fit_width(false);  //
```

導入はとてもシンプルです。Cargo に対応しているので、cargo add 一発で完了します。

# 3.1 ダウンロード

**https://www.pao.ac/barcode.rust/#download**

| パッケージ | 内容 | こんな方に |
|---|---|---|
| Easy 2 Steps | QRコード生成の最小RESTサーバー | まずは動かしてみたい方 |
| All-in-One | 全18種対応のフル機能RESTサーバー | 本格的に評価したい方 |
| WASM版 | ブラウザで動くデモ | Rust以外の環境でも試したい方 |

# 3.2 cargo add でインストール

```
cargo add barcode-pao
```

これだけです。unsafe不要なので、安全性を確保したままプロジェクトに組み込めます。

# 3.3 ファイル構成

## Easy 2 Steps サンプル

```
barcode_rust_easy2steps/
├── Cargo.toml            #
├── src/
│   └── main.rs           # QR100
├── templates/
│   └── index.html        # Web
└── barcode-pao/          #
    ├── src/
    │   ├── lib.rs
    │   ├── base.rs
    │   ├── qr.rs
    │   └── ...
    └── Cargo.toml
```

## 起動方法

```
cd barcode_rust_easy2steps
cargo run
# → http://localhost:5720
```

ブラウザで開くと、QRコードの生成画面が表示されます。テキストを入力して「生成」ボタンを押すだけ。

> ヒント: サンプルは axum を使用した非同期Webサーバーで構成されています。Rust の async/await を活かした高性能なバーコード生成サーバーです。

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 QRコードをPNGで生成

```
use barcode_pao::{QR, FORMAT_PNG, QR_ECC_M};
use std::fs;

fn main() -> Result<(), Box<dyn std::error::Error>> {
    // Step 1: QR
    let mut qr = QR::new(FORMAT_PNG);
    qr.set_error_correction_level(QR_ECC_M); // : Medium

    // Step 2:
    qr.draw("https://www.pao.ac/", 300)?;

    // Base64 — HTML
    let base64 = qr.base_2d.base.get_image_base64()?;
    println!("Base64: {} ...", &base64[..50]);

    //
    let image_bytes = qr.base_2d.base.get_image_memory()?;
    fs::write("qr.png", &image_bytes)?;
    println!("Saved: qr.png");

    Ok(())
}
```

# 4.2 1Dバーコードを SVGで生成

```
use barcode_pao::{Code128, FORMAT_SVG};
use std::fs;

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let mut bc = Code128::new(FORMAT_SVG);
    bc.base_1d.show_text = true;
    bc.base_1d.text_even_spacing = true;

    bc.draw("Hello-2026", 400, 100)?;

    let svg = bc.base_1d.base.get_svg()?;
    fs::write("code128.svg", svg.as_bytes())?;
    println!("Saved: code128.svg");

    Ok(())
}
```

# 4.3 REST APIサーバーで提供

Rust の真骨頂——サーバーサイドでバーコードを生成し、APIとして提供する例です。

```
use axum::{extract::Query, routing::get, Json, Router};
use barcode_pao::{QR, FORMAT_PNG};
use serde::Deserialize;
use std::collections::HashMap;

#[derive(Deserialize)]
struct Params {
    code: Option<String>,
}

async fn handler(Query(p): Query<Params>) -> Json<HashMap<&'static str, String>> {
    let code = p.code.unwrap_or_else(|| "https://www.pao.ac/".to_string());

    let mut qr = QR::new(FORMAT_PNG);
    qr.draw(&code, 300).unwrap();
    let b64 = qr.base_2d.base.get_image_base64().unwrap();

    Json(HashMap::from([("base64", b64)]))
}

#[tokio::main]
async fn main() {
    let app = Router::new().route("/api/qr", get(handler));
    let listener = tokio::net::TcpListener::bind("0.0.0.0:5720").await.unwrap();
    println!("→ http://localhost:5720/api/qr?code=Hello");
    axum::serve(listener, app).await.unwrap();
}
```

curl http://localhost:5720/api/qr?code=Hello で JSON が返ってきます。フロントエンドから fetch するだけでバーコードが表示できます。

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。

# 5.1 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。

```
let mut bc = Code39::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.base_1d.show_start_stop = true;   // *HELLO123*
bc.draw("HELLO123", 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。

```
let mut bc = Code93::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.draw("Hello123", 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 CODE128_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。

```
let mut bc = Code128::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.code_mode = CODE128_AUTO;  // AUTO / A / B / C
bc.draw("Hello123", 400, 100)?;
```

| コードモード | 定数 | 説明 |
|---|---|---|
| AUTO | CODE128_AUTO | 自動で最短幅に最適化（おすすめ） |
| A | CODE128_CODE_A | 制御文字 + 数字 + 英大文字 |
| B | CODE128_CODE_B | 数字 + 英大文字 + 英小文字 + 記号 |
| C | CODE128_CODE_C | 数字のみ（2桁ずつ高密度エンコード） |

> ヒント: AUTO モードでは、データの内容を解析して CODE-A / B / C を動的に切り替え、最短幅になるよう自動最適化します。特別な理由がなければ AUTO のままで問題ありません。

## 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。

```
let mut bc = NW7::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.base_1d.show_start_stop = true;
bc.draw("A1234567A", 400, 100)?;
```

## ITF — 段ボール箱でよく見るバーコード

Interleaved 2 of 5。バーとスペースを交互に使って2桁ずつエンコードするため、高密度です。入力は偶数桁である必要があります。

```
let mut bc = ITF::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.draw("123456", 400, 100)?;
```

# 5.2 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。

```
let mut qr = QR::new(FORMAT_PNG);
qr.set_string_encoding("utf-8");
qr.set_error_correction_level(QR_ECC_M);  // L(7%) / M(15%) / Q(25%) / H(30%)
qr.set_version(0);                        // 0=
qr.draw("https://www.pao.ac/", 300)?;
```

| 誤り訂正レベル | 定数 | 復元能力 | こんなときに |
|---|---|---|---|
| L | QR_ECC_L | 約7% | データ量を最優先したい |
| M | QR_ECC_M | 約15% | 一般的な用途（おすすめ） |
| Q | QR_ECC_Q | 約25% | やや過酷な環境（汚れ・傷） |
| H | QR_ECC_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。

```
let mut dm = DataMatrix::new(FORMAT_PNG);
dm.set_string_encoding("utf-8");
dm.set_code_size(DX_SZ_AUTO);
dm.set_encode_scheme(DX_SCHEME_AUTO_BEST);
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。

```
let mut pdf = PDF417::new(FORMAT_PNG);
pdf.set_string_encoding("utf-8");
pdf.set_error_level(PDF417_ERROR_LEVEL_2);
pdf.set_columns(3);
pdf.set_aspect_ratio(3.0);
pdf.set_y_height(3);
pdf.draw("Hello World", 400, 100)?;
```

# 5.3 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。

```
let mut gs1 = GS1_128::new(FORMAT_PNG);
gs1.base_1d.show_text = true;
gs1.base_1d.text_even_spacing = true;
gs1.draw("[01]04912345123459[10]ABC123", 500, 120)?;
```

| 特殊文字 | 意味 |
|---|---|
| [AI] | AIを角括弧で表記（テキスト表示で括弧表示） |
| {FNC1} | ファンクション1（可変長フィールドの区切り） |

## 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。draw_convenience() で生成します。

```
let mut gs1 = GS1_128::new(FORMAT_PNG);
gs1.base_1d.show_text = true;
gs1.draw_convenience(
    "{FNC1}9191234500000000000000452087500401310029500", 500, 150,
)?;
```

## GS1 DataBar 標準型 — 青果・精肉売り場で活躍

重量や価格情報をコンパクトにエンコードできるバーコードです。

```
let mut db = GS1DataBar14::new(FORMAT_PNG, OMNIDIRECTIONAL);
db.base_1d.show_text = true;
db.draw("1234567890128", 200, 80)?;
```

| シンボルタイプ | 定数 | 説明 |
|---|---|---|
| 標準型 | OMNIDIRECTIONAL | どの方向からでも読み取り可能 |
| 二層型 | STACKED | 省スペース |
| 標準二層型 | STACKED_OMNIDIRECTIONAL | 二層かつ全方向対応 |

## GS1 DataBar 拡張型 — クーポンや特売情報も格納可能

可変長データに対応し、多層型（スタック）にも対応しています。

```
let mut db = GS1DataBarExpanded::new(FORMAT_PNG, UNSTACKED);
db.base_1d.show_text = true;
db.draw("[01]90012345678908[10]ABC123", 400, 80)?;

//
let mut db_stacked = GS1DataBarExpanded::new(FORMAT_PNG, STACKED_EXP);
db_stacked.set_no_of_columns(4);
db_stacked.draw("[01]90012345678908[10]ABC123", 300, 100)?;
```

# 5.4 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。

```
let mut yubin = YubinCustomer::new(FORMAT_PNG);
yubin.draw("27500263-29-2-401", 25)?;
```

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
let mut jan13 = JAN13::new(FORMAT_PNG);
jan13.base_1d.show_text = true;
jan13.base_1d.extended_guard = true;          //
jan13.base_1d.text_even_spacing = false;      //
jan13.draw("491234567890", 300, 100)?;

// JAN-8
let mut jan8 = JAN8::new(FORMAT_PNG);
jan8.base_1d.show_text = true;
jan8.base_1d.extended_guard = true;
jan8.base_1d.text_even_spacing = false;
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 extended_guard と text_even_spacing 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、extended_guard=true + text_even_spacing=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
let mut upc_a = UPC_A::new(FORMAT_PNG);
upc_a.base_1d.show_text = true;
upc_a.base_1d.extended_guard = true;
upc_a.base_1d.text_even_spacing = false;
upc_a.draw("01234567890", 300, 100)?;

// UPC-E8
let mut upc_e = UPC_E::new(FORMAT_PNG);
upc_e.base_1d.show_text = true;
upc_e.base_1d.extended_guard = true;
upc_e.base_1d.text_even_spacing = false;
upc_e.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。

# 6.1 共通メソッド（全バーコード）

すべてのバーコード型で使用できるメソッドです。

## set_output_format(format: &str)

出力形式を設定します。

| パラメータ | 型 | 説明 |
|---|---|---|
| format | &str | "png", "jpeg", "svg" |

```
bc.base_1d.base.set_output_format(FORMAT_PNG);   // PNGBase64—
bc.base_1d.base.set_output_format(FORMAT_JPEG);  // JPEG
bc.base_1d.base.set_output_format(FORMAT_SVG);   // SVG
```

デフォルト: "png"（コンストラクタで指定）

## set_foreground_color(r: u8, g: u8, b: u8, a: u8)

前景色（バーの色）を設定します。

| パラメータ | 型 | 説明 |
|---|---|---|
| r | u8 | 赤（0〜255） |
| g | u8 | 緑（0〜255） |
| b | u8 | 青（0〜255） |
| a | u8 | 透明度（0=透明 〜 255=不透明） |

```
bc.base_1d.base.set_foreground_color(0, 0, 0, 255);      //
bc.base_1d.base.set_foreground_color(0, 0, 128, 255);    //
bc.base_1d.base.set_foreground_color(255, 0, 0, 128);    //
```

## set_background_color(r: u8, g: u8, b: u8, a: u8)

背景色を設定します。

```
bc.base_1d.base.set_background_color(255, 255, 255, 255);  //
bc.base_1d.base.set_background_color(255, 255, 240, 255);  //
bc.base_1d.base.set_background_color(0, 0, 0, 0);          //
```

## set_px_adjust_black(adj: i32) / set_px_adjust_white(adj: i32)

黒バー / 白スペースの幅を微調整します。印刷時のにじみ補正に使います。

```
bc.base_1d.base.set_px_adjust_black(-1);  // 1px
bc.base_1d.base.set_px_adjust_white(1);   // 1px
```

デフォルト: 0

## set_fit_width(fit: bool)

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

> 仕組み: true の場合、バーの幅に小数ピクセルを使用して指定幅にぴったり収めます。false の場合は整数ピクセルのみ使用するため、指定幅より若干小さくなることがあります。

## get_image_base64() -> Result<String>

Base64エンコードされたデータURIを返します。

**戻り値:**

- PNG: "..." 形式
- JPEG: "..." 形式
- SVG: エラー（SVGモードでは get_svg() を使用）

## get_svg() -> Result<String>

SVG文字列を返します（SVGモード時のみ）。

戻り値: "<svg xmlns="...">...</svg>" 形式

## get_image_memory() -> Result<Vec<u8>>

PNG/JPEGのバイト列を返します。ファイル保存や HTTP レスポンスに直接使用できます。

# 6.2 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フィールドとメソッドです。

## draw(code: &str, width: i32, height: i32) -> Result<()>

バーコードを生成します。

| パラメータ | 型 | 説明 |
|---|---|---|
| code | &str | エンコードするデータ |
| width | i32 | 画像の幅（px） |
| height | i32 | 画像の高さ（px） |

```
bc.draw("HELLO123", 400, 100)?;
```

## show_text: bool

バーコード下部のテキスト表示を切り替えます。

デフォルト: true

## text_even_spacing: bool

テキストの均等割付を設定します。

```
bc.base_1d.text_even_spacing = true;   // 1
bc.base_1d.text_even_spacing = false;  //
```

デフォルト: true

> 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 extended_guard = 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。

## set_text_font_scale(scale: f64)

テキストのフォントサイズ倍率を設定します。

デフォルト: 1.0

## set_text_vertical_offset_scale(scale: f64)

テキストの垂直オフセット倍率を設定します。値を小さくするとバーとテキストの間隔が狭くなります。

デフォルト: 1.0

## set_min_line_width(width: i32)

最小線幅を設定します（ITF, Matrix2of5, NEC2of5 向け）。

デフォルト: 1

# 6.3 Code39

型: Code39 — 工業用途の定番バーコード

コンストラクタ: Code39::new(output_format: &str) -> Code39

入力可能文字: 0-9, A-Z, - . $ / + %, スペース

## 固有フィールド

### show_start_stop: bool

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: true

## 使用例

```
let mut bc = Code39::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.base_1d.show_start_stop = true;
bc.draw("HELLO123", 400, 100)?;
```

# 6.4 Code93

型: Code93 — Code39の高密度版

コンストラクタ: Code93::new(output_format: &str) -> Code93

入力可能文字: ASCII全文字（0x00〜0x7F）

固有フィールド: なし（共通フィールドのみ）

## 使用例

```
let mut bc = Code93::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.draw("Hello123!@#", 400, 100)?;
```

# 6.5 Code128

型: Code128 — 物流の標準バーコード

コンストラクタ: Code128::new(output_format: &str) -> Code128

入力可能文字: ASCII全文字（0x00〜0x7F）

## 固有フィールド

### code_mode: i32

| 定数 | 対応文字 |
| --- | --- |
| CODE128_AUTO | 自動で最短幅に最適化（おすすめ） |
| CODE128_CODE_A | 制御文字 + 数字 + 英大文字 + 一部記号 |
| CODE128_CODE_B | 数字 + 英大文字 + 英小文字 + 記号 |
| CODE128_CODE_C | 数字のみ（2桁ずつ高密度エンコード） |

デフォルト: CODE128_AUTO

## 使用例

```
let mut bc = Code128::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.code_mode = CODE128_AUTO;
bc.draw("Hello123", 400, 100)?;
```

# 6.6 GS1-128

型: GS1_128 — GS1標準準拠。物流・医療分野で使用

コンストラクタ: GS1_128::new(output_format: &str) -> GS1_128

入力形式: AI（アプリケーション識別子）とデータの組み合わせ

## 固有メソッド

### draw(code: &str, width: i32, height: i32) -> Result<()>

通常のGS1-128バーコードを生成します。

### draw_convenience(code: &str, width: i32, height: i32) -> Result<()>

標準料金代理収納用（コンビニバーコード）を生成します。

## 使用例

```
// GS1-128
let mut gs1 = GS1_128::new(FORMAT_PNG);
gs1.base_1d.show_text = true;
gs1.base_1d.text_even_spacing = true;
gs1.draw("[01]04912345123459[10]ABC123", 500, 120)?;

//
let mut gs1c = GS1_128::new(FORMAT_PNG);
gs1c.base_1d.show_text = true;
gs1c.draw_convenience(
    "{FNC1}9191234500000000000000452087500401310029500", 500, 150,
)?;
```

# 6.7 NW-7（Codabar）

型: NW7 — 宅配便・図書館で使用

コンストラクタ: NW7::new(output_format: &str) -> NW7

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

## 固有フィールド

### show_start_stop: bool

デフォルト: true

## 使用例

```
let mut bc = NW7::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.base_1d.show_start_stop = true;
bc.draw("A1234567A", 400, 100)?;
```

# 6.8 ITF（Interleaved 2 of 5）

型: ITF — 集合包装用バーコード

コンストラクタ: ITF::new(output_format: &str) -> ITF

入力可能文字: 0-9 のみ（偶数桁必須）

固有フィールド: なし

## 使用例

```
let mut bc = ITF::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.draw("123456", 400, 100)?;
```

# 6.9 Matrix 2of5

型: Matrix2of5 — 工業用の数字専用バーコード

コンストラクタ: Matrix2of5::new(output_format: &str) -> Matrix2of5

入力可能文字: 0-9 のみ

固有フィールド: なし

## 使用例

```
let mut bc = Matrix2of5::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.draw("1234567890", 400, 100)?;
```

# 6.10 NEC 2of5

型: NEC2of5 — 日本の工業用途向け

コンストラクタ: NEC2of5::new(output_format: &str) -> NEC2of5

入力可能文字: 0-9 のみ

固有フィールド: なし

## 使用例

```
let mut bc = NEC2of5::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.text_even_spacing = true;
bc.draw("1234567890", 400, 100)?;
```

# 6.11 JAN-8（EAN-8）

型: JAN8 — 小型商品用の8桁バーコード

コンストラクタ: JAN8::new(output_format: &str) -> JAN8

入力: 数字7桁（チェックディジットは自動計算）

## 固有フィールド

### extended_guard: bool

ガードバーの拡張。true にすると商品バーコードとしての標準的な外観になります。

デフォルト: true

## テキスト表示パターン

| extended_guard | text_even_spacing | 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 |

## 使用例

```
let mut bc = JAN8::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.extended_guard = true;
bc.base_1d.text_even_spacing = false;
bc.draw("4901234", 200, 100)?;
```

# 6.12 JAN-13（EAN-13）

型: JAN13 — 日本の標準的な商品バーコード（13桁）

コンストラクタ: JAN13::new(output_format: &str) -> JAN13

入力: 数字12桁（チェックディジットは自動計算）

## 固有フィールド

### extended_guard: bool

6.11 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。

## 使用例

```
let mut bc = JAN13::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.extended_guard = true;
bc.base_1d.text_even_spacing = false;
bc.draw("491234567890", 300, 100)?;
```

# 6.13 UPC-A

型: UPC_A — 北米の商品コード（12桁）

コンストラクタ: UPC_A::new(output_format: &str) -> UPC_A

入力: 数字11桁（チェックディジットは自動計算）

## 固有フィールド

### extended_guard: bool

6.11 JAN-8 と同じです。デフォルト: true

## 使用例

```
let mut bc = UPC_A::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.extended_guard = true;
bc.base_1d.text_even_spacing = false;
bc.draw("01234567890", 300, 100)?;
```

# 6.14 UPC-E

型: UPC_E — UPC-Aの短縮版（8桁）。小型商品用

コンストラクタ: UPC_E::new(output_format: &str) -> UPC_E

入力: 数字6桁（チェックディジットは自動計算）

## 固有フィールド

### extended_guard: bool

6.11 JAN-8 と同じです。デフォルト: true

## 使用例

```
let mut bc = UPC_E::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.base_1d.extended_guard = true;
bc.base_1d.text_even_spacing = false;
bc.draw("0123456", 200, 100)?;
```

# 6.15 GS1 DataBar 標準型

型: GS1DataBar14 — 生鮮食品向けのコンパクトバーコード

コンストラクタ: GS1DataBar14::new(output_format: &str, symbol_type: SymbolType14) -> GS1DataBar14

入力: 数字 8〜13桁（チェックディジットは自動計算）

## シンボルタイプ

| 定数 | 説明 |
|---|---|
| OMNIDIRECTIONAL | 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_OMNIDIRECTIONAL | 標準二層型 |

## 使用例

```
let mut bc = GS1DataBar14::new(FORMAT_PNG, OMNIDIRECTIONAL);
bc.base_1d.show_text = true;
bc.draw("1234567890128", 200, 80)?;
```

# 6.16 GS1 DataBar 限定型

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

コンストラクタ: GS1DataBarLimited::new(output_format: &str) -> GS1DataBarLimited

入力: 数字 8〜13桁（先頭桁は0または1のみ）

## 使用例

```
let mut bc = GS1DataBarLimited::new(FORMAT_PNG);
bc.base_1d.show_text = true;
bc.draw("0123456789012", 200, 60)?;
```

# 6.17 GS1 DataBar 拡張型

型: GS1DataBarExpanded — 可変長データ対応

コンストラクタ: GS1DataBarExpanded::new(output_format: &str, symbol_type: ExpandedSymbolType) -> GS1DataBarExpanded

入力: AI + データの組み合わせ

## シンボルタイプ

| 定数 | 説明 |
|---|---|
| UNSTACKED | 一層型 |
| STACKED_EXP | 多層型（スペースが限られる場合に） |

## 固有メソッド

### set_no_of_columns(columns: i32)

多層型のセグメント数（列数）を設定します。偶数推奨。デフォルト: 2

## 使用例

```
//
let mut db = GS1DataBarExpanded::new(FORMAT_PNG, UNSTACKED);
db.base_1d.show_text = true;
db.draw("[01]90012345678908[10]ABC123", 400, 80)?;

//
let mut db_stacked = GS1DataBarExpanded::new(FORMAT_PNG, STACKED_EXP);
db_stacked.set_no_of_columns(4);
db_stacked.draw("[01]90012345678908[10]ABC123", 300, 100)?;
```

# 6.18 郵便カスタマバーコード

型: YubinCustomer — 日本郵便の住所バーコード

コンストラクタ: YubinCustomer::new(output_format: &str) -> YubinCustomer

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(code: &str, height: i32) -> Result<()>

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。

> 注意: テキスト関連フィールド（show_text, text_even_spacing 等）は使用できません。set_foreground_color(), set_background_color() は使用可能です。

## 使用例

```
let mut bc = YubinCustomer::new(FORMAT_PNG);
bc.draw("27500263-29-2-401", 25)?;
```

# 6.19 QRコード

型: QR — 日本発、世界で最も普及している2次元バーコード

コンストラクタ: QR::new(output_format: &str) -> QR

入力: 数字、英数字、バイナリ、漢字（Shift-JIS）

## 固有メソッド

### draw(code: &str, size: i32) -> Result<()>

| パラメータ | 型 | 説明 |
|---|---|---|
| code | &str | エンコードするデータ |
| size | i32 | 画像サイズ（px、正方形） |

### set_string_encoding(encoding: &str)

| 値 | 説明 |
|---|---|
| "utf-8" | UTF-8（おすすめ） |
| "shift-jis" | Shift-JIS（レガシー環境との互換性が必要な場合） |

デフォルト: "utf-8"

### set_error_correction_level(level: i32)

| 定数 | 復元能力 | こんなときに |
|---|---|---|
| QR_ECC_L | 約7% | データ量優先 |
| QR_ECC_M | 約15% | 一般的な用途（おすすめ） |
| QR_ECC_Q | 約25% | 汚れ・傷への耐性が必要 |
| QR_ECC_H | 約30% | 最高品質。ロゴ重ね時にも |

デフォルト: QR_ECC_M

### set_version(version: i32)

QRコードのバージョン（セルの数）を指定します。0（自動）〜 40。

デフォルト: 0（データに応じた最小バージョンを自動選択）

### set_encode_mode(mode: &str)

| 定数 | 説明 |
| --- | --- |
| QR_MODE_BINARY | バイトデータ（デフォルト、おすすめ） |
| QR_MODE_NUMERIC | 数字のみ（最高効率） |
| QR_MODE_ALPHA_NUMERIC | 英数字 |
| QR_MODE_KANJI | 漢字（Shift-JIS） |

デフォルト: QR_MODE_BINARY

## 使用例

```
let mut qr = QR::new(FORMAT_SVG);
qr.set_string_encoding("utf-8");
qr.set_error_correction_level(QR_ECC_M);
qr.set_version(0);
qr.base_2d.base.set_fit_width(true);
qr.draw("https://www.pao.ac/", 300)?;
let svg = qr.base_2d.base.get_svg()?;
```

# 6.20 DataMatrix

型: DataMatrix — 超小型マーキングの世界標準

コンストラクタ: DataMatrix::new(output_format: &str) -> DataMatrix

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(code: &str, size: i32) -> Result<()>

| パラメータ | 型 | 説明 |
|---|---|---|
| code | &str | エンコードするデータ |
| size | i32 | 画像サイズ（px、正方形） |

### set_string_encoding(encoding: &str)

"utf-8"（デフォルト）または "shift-jis"

### set_code_size(size: i32)

| 主な定数 | 説明 |
|---|---|
| DX_SZ_AUTO | 自動（おすすめ） |
| DX_SZ_10X10 ～ DX_SZ_144X144 | 正方形 |
| DX_SZ_8X18, DX_SZ_8X32 等 | 矩形 |

デフォルト: DX_SZ_AUTO

### set_encode_scheme(scheme: i32)

| 定数 | 説明 |
|---|---|
| DX_SCHEME_AUTO_BEST | 自動選択（おすすめ） |
| DX_SCHEME_ASCII | ASCII |
| DX_SCHEME_C40 | 英数字 |
| DX_SCHEME_TEXT | テキスト（小文字優先） |
| DX_SCHEME_X12 | ANSI X12 EDI |
| DX_SCHEME_EDIFACT | EDIFACT |
| DX_SCHEME_BASE256 | バイナリ |

デフォルト: DX_SCHEME_AUTO_BEST

## 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)?;
```

## 使用例

```
let mut dm = DataMatrix::new(FORMAT_SVG);
dm.set_string_encoding("utf-8");
dm.set_code_size(DX_SZ_AUTO);
dm.set_encode_scheme(DX_SCHEME_AUTO_BEST);
dm.base_2d.base.set_fit_width(true);
dm.draw("Hello World", 200)?;
let svg = dm.base_2d.base.get_svg()?;
```

# 6.21 PDF417

型: PDF417 — 大容量2次元バーコード。運転免許証・搭乗券に使用

コンストラクタ: PDF417::new(output_format: &str) -> PDF417

入力: テキスト、数字、バイナリ

## 固有メソッド

### draw(code: &str, width: i32, height: i32) -> Result<()>

| パラメータ | 型 | 説明 |
|---|---|---|
| code | &str | エンコードするデータ |
| width | i32 | 画像の幅（px） |
| height | i32 | 画像の高さ（px） |

### set_string_encoding(encoding: &str)

"utf-8"（デフォルト）または "shift-jis"

### set_error_level(level: i32)

| 定数 | 訂正能力 |
|---|---|
| PDF417_ERROR_LEVEL_0 | 最小 |
| PDF417_ERROR_LEVEL_1 | 低 |
| PDF417_ERROR_LEVEL_2 | 標準（おすすめ） |
| PDF417_ERROR_LEVEL_3 〜 PDF417_ERROR_LEVEL_8 | 高〜最大 |

デフォルト: PDF417_ERROR_LEVEL_2

### set_columns(columns: i32)

列数。0=自動、1〜30で指定。デフォルト: 0

### set_rows(rows: i32)

行数。0=自動、3〜90で指定。デフォルト: 0

### set_aspect_ratio(ratio: f64)

縦横比。1.0〜10.0。デフォルト: 3.0

### set_y_height(height: i32)

Y方向の高さ係数。1〜10。デフォルト: 3

## 使用例

```
let mut pdf = PDF417::new(FORMAT_SVG);
pdf.set_string_encoding("utf-8");
pdf.set_error_level(PDF417_ERROR_LEVEL_2);
pdf.set_columns(4);
pdf.set_rows(0);
pdf.set_aspect_ratio(3.0);
pdf.set_y_height(3);
pdf.base_2d.base.set_fit_width(true);
pdf.draw("Hello World", 400, 100)?;
let svg = pdf.base_2d.base.get_svg()?;
```

Barcode.Rust は wasm-pack / wasm-bindgen を使って WebAssembly にコンパイルすることで、ブラウザ上でも動作します。Rust の WASM バイナリは約 850KB と軽量で、高速に読み込めます。

## ビルド方法

```
wasm-pack build --target web --release
```

## 必要ファイル

```
pkg/
├── barcode_pao_bg.wasm    ← Rust WASM 850KB
├── barcode_pao.js         ← wasm-bindgen  JS
└── barcode_pao.d.ts       ← TypeScript
```

## 使い方

```
<script type="module">
import init, { drawQR } from './pkg/barcode_pao.js';

async function run() {
    await init();
    // JavaScript  Rust
    const base64 = drawQR("https://www.pao.ac/", 300);
    document.getElementById("barcode").src = base64;
}
run();
</script>
```

> ヒント: WASM版のダウンロードパッケージには、すぐに試せるデモ HTML が含まれています。Rust の WASM バイナリは Go 版（約5MB）と比べて大幅に小さく、読み込み速度に優れています。

# Rust バージョン

| 項目 | 要件 |
| --- | --- |
| Rust | 1.70 以降 |

# 依存クレート

| クレート | 用途 |
|---|---|
| image | PNG / JPEG 画像生成 |
| ab_glyph | TrueType フォント描画（テキスト表示） |

上記以外はすべて Rust 標準ライブラリのみを使用しています。unsafe は不要です。

# 対応OS（WASM版除く）

Rust がサポートするすべてのOS / アーキテクチャで動作します。

| OS | アーキテクチャ |
|---|---|
| Windows | x86_64, aarch64 |
| macOS | x86_64 (Intel), aarch64 (Apple Silicon) |
| Linux | x86_64, aarch64, arm |

# WASM版の対応ブラウザ

| ブラウザ | 対応バージョン |
|---|---|
| Google Chrome | 57 以降 |
| Mozilla Firefox | 53 以降 |
| Safari | 11 以降 |
| Microsoft Edge | 16 以降 |

# 使用許諾

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

# ライセンス

Barcode.Rust は有限会社パオ・アット・オフィスの製品です。

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。ライセンスの詳細は弊社Webサイトをご確認ください。

# 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.rust/ |
| メール | info@pao.ac |

# 関連製品

| 製品 | 対応環境 |
|---|---|
| Barcode.net | .NET（C#, VB.NET） |
| Barcode.jar | Java |
| Barcode.php | PHP |
| Barcode.wasm | JavaScript / TypeScript（ブラウザ） |
| Barcode.py | Python |
| Barcode.Flutter | Flutter / Dart |
| Barcode.Go | Go |

**Barcode.Rust ユーザーズマニュアル**

バージョン 1.0 — 2026年2月