Barcode.Rust (WASM)

C++ WASMエンジン Rust ラッパー

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

C++ バーコードエンジンを Rust から手軽に利用。

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 C++ WASM版とは

Barcode.Rust （C++ WASMエンジン） は、C++ で書かれた高速バーコードエンジンを Node.js 経由 で Rust から利用するラッパーライブラリです。

C++ バーコードエンジンは Emscripten で WebAssembly にコンパイルされており、Node.js をサブプロセスとして起動して WASM を実行します。Rust 側は JSON で命令を送り、生成されたバーコード（Base64 PNG や SVG 文字列）を受け取ります。

```
Rust  → JSON → node _barcode_runner.mjs → barcode.mjs → WASM → JSON → Rust
```

Pure Rust版と同じ18種のバーコードを生成でき、外部依存は serde と serde_json のみです。

```
use barcode_pao_wasm::{QR, FORMAT_PNG};

let qr = QR::new(FORMAT_PNG);
let result = qr.draw("https://www.pao.ac/", 300)?;
// result  "..."
```

1.2 特長

特長	説明
C++ 高速エンジン	実績あるC++バーコードエンジンをそのまま利用。高品質な出力
最小依存	serde / serde_json のみ。標準ライブラリの std::process::Command で Node.js を
Node.js ブリッジ	Emscripten WASM を Node.js 経由で実行。インストールは node のみ
シンプルAPI	draw() が直接 Base64/SVG 文字列を返す。2ステップで完結
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
PNG / JPEG / SVG	画面表示にはPNG、印刷にはSVG。用途で使い分け可能

1.3 対応バーコード一覧

1次元バーコード（15種類）

バーコード	コンストラクタ
Code39	Code39::new()
Code93	Code93::new()
Code128	Code128::new()
GS1-128	GS1128::new()
NW-7 (Codabar)	NW7::new()
ITF	ITF::new()
Matrix 2of5	Matrix2of5::new()
NEC 2of5	NEC2of5::new()
JAN-8 (EAN-8)	Jan8::new()
JAN-13 (EAN-13)	Jan13::new()
UPC-A	UPCA::new()
UPC-E	UPCE::new()
GS1 DataBar 標準型	GS1DataBar14::new()
GS1 DataBar 限定型	GS1DataBarLimited::new()
GS1 DataBar 拡張型	GS1DataBarExpanded::new()

特殊バーコード（1種類）

バーコード	コンストラクタ
郵便カスタマバーコード	YubinCustomer::new()

2次元バーコード（3種類）

バーコード	コンストラクタ
QRコード	QR::new()
DataMatrix	DataMatrix::new()
PDF417	PDF417::new()

2.1 動作要件

項目	要件
Rust	1.70 以降 (edition 2021)
Node.js	16 以降
OS	Windows / macOS / Linux

Node.js は WASM エンジンの実行に必要です。node コマンドが PATH に含まれている必要があります。

2.2 ダウンロード

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

C++ WASM版のダウンロードパッケージには、Easy 2 Steps と All-in-One の2つのサンプルが含まれています。

2.3 ファイル構成

```
barcode_rust_wasm_cpp/
├── easy2steps/
│   ├── Cargo.toml
│   ├── src/
│   │   └── main.rs            # QRaxum
│   └── templates/
│       └── index.html         # Web
├── allinone/
│   ├── Cargo.toml
│   ├── src/
│   │   └── main.rs            # 18
│   └── templates/
│       └── index.html
└── barcode_pao_wasm/          # WASM
    ├── Cargo.toml
    ├── src/
    │   ├── lib.rs
    │   └── wrapper.rs         # Rust → WASM
    └── wasm/
        ├── barcode.wasm       # C++ WASM
        ├── barcode.js         # EmscriptenJS
        ├── barcode.mjs        # ES
        └── _barcode_runner.mjs # Node.js
```

起動方法

```
cd barcode_rust_wasm_cpp/easy2steps
cargo run
# → http://localhost:5722
```

3.1 QRコードをPNGで生成

```
use barcode_pao_wasm::{QR, FORMAT_PNG};

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

    // Step 2:  → Base64
    let result = qr.draw("https://www.pao.ac/", 300)?;

    // result  "..."
    println!("Base64: {}...", &result[..50]);

    Ok(())
}
```

ポイント: Pure Rust版と異なり、draw() が直接 Base64 文字列を返します。get_image_base64() を呼ぶ必要はありません。

3.2 SVGベクター出力

```
use barcode_pao_wasm::{Code128, FORMAT_SVG};

let mut code = Code128::new(FORMAT_SVG);
code.base_1d.set_show_text(true);
code.base_1d.set_text_even_spacing(true);

let svg = code.draw("Hello-2026", 400, 100)?;
// svg  "<svg xmlns="...">...</svg>"
```

3.3 REST APIサーバー（axum）

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

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

async fn handle(Query(p): Query<Params>) -> Json<HashMap<&'static str, String>> {
    let code = p.code.unwrap_or("https://www.pao.ac/".into());
    let qr = QR::new(FORMAT_PNG);
    let b64 = qr.draw(&code, 300).unwrap();
    Json(HashMap::from([("base64", b64)]))
}

#[tokio::main]
async fn main() {
    let app = Router::new().route("/api/qr", get(handle));
    let listener = tokio::net::TcpListener::bind("0.0.0.0:5722").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

4.1 共通メソッド

すべてのバーコード型の base（BarcodeWasmBase）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（コンストラクタで指定済みの場合は不要）。

定数	説明
FORMAT_PNG	PNG画像（Base64）
FORMAT_JPEG	JPEG画像（Base64）
FORMAT_SVG	SVGベクター

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

前景色（バーの色）を RGBA で設定します。

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

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

背景色を RGBA で設定します。

```
code.base_1d.base.set_background_color(255, 255, 240, 255);
```

4.2 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）の base_1d（Barcode1DBase）で使用できるメソッドです。

draw(code: &str, width: u32, height: u32) -> Result<String, String>

バーコードを生成し、Base64 または SVG 文字列を返します。

```
let result = code.draw("HELLO123", 400, 100)?;
```

set_show_text(show: bool)

バーコード下部のテキスト表示を設定します。

set_text_even_spacing(even: bool)

テキストの均等割付を設定します。

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。

set_fit_width(fit: bool)

指定幅にぴったり収めるかどうかを設定します。

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。

4.3 各バーコード型

Code39

```
use barcode_pao_wasm::{Code39, FORMAT_PNG};

let mut code = Code39::new(FORMAT_PNG);
code.base_1d.set_show_text(true);
code.set_show_start_stop(true);
let result = code.draw("HELLO123", 400, 100)?;
```

固有メソッド: set_show_start_stop(show: bool)

Code128

```
let mut code = Code128::new(FORMAT_PNG);
code.base_1d.set_show_text(true);
code.set_code_mode("AUTO");  // AUTO / A / B / C
let result = code.draw("Hello123", 400, 100)?;
```

固有メソッド: set_code_mode(mode: &str)

GS1-128

```
let mut gs1 = GS1128::new(FORMAT_PNG);
gs1.base_1d.set_show_text(true);
let result = gs1.draw("[01]04912345123459[10]ABC123", 500, 120)?;
```

NW-7 (Codabar)

```
let mut code = NW7::new(FORMAT_PNG);
code.set_show_start_stop(true);
let result = code.draw("A1234567A", 400, 100)?;
```

JAN-13 / JAN-8

```
let mut jan = Jan13::new(FORMAT_PNG);
jan.base_1d.set_show_text(true);
jan.set_extended_guard(true);
jan.base_1d.set_text_even_spacing(false);
let result = jan.draw("491234567890", 300, 100)?;
```

固有メソッド: set_extended_guard(ext: bool)

GS1 DataBar 標準型

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

固有メソッド: set_symbol_type(t: &str) — "OMNIDIRECTIONAL", "STACKED", "STACKED_OMNIDIRECTIONAL"

GS1 DataBar 拡張型

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

固有メソッド: set_symbol_type(t: &str), set_no_of_columns(cols: u32)

郵便カスタマバーコード

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

高さのみ指定（幅は自動計算）。

QRコード

```
let mut qr = QR::new(FORMAT_PNG);
qr.set_error_correction_level("M");  // L / M / Q / H
qr.set_version(0);                    // 0=
let result = qr.draw("https://www.pao.ac/", 300)?;
```

固有メソッド: set_error_correction_level(level: &str), set_version(v: i32), set_encode_mode(mode: &str)

DataMatrix

```
let mut dm = DataMatrix::new(FORMAT_PNG);
dm.base_2d.set_string_encoding("utf-8");
let result = dm.draw("Hello World", 200)?;
```

固有メソッド: set_code_size(size: &str), set_encode_scheme(scheme: &str)

PDF417

```
let mut pdf = PDF417::new(FORMAT_PNG);
pdf.set_error_level(2);
pdf.set_columns(4);
let result = pdf.draw("Hello World", 400, 100)?;
```

固有メソッド: set_error_level(level: i32), set_columns(cols: i32), set_rows(rows: i32), set_aspect_ratio(ratio: f64), set_y_height(h: i32)

PDF417 の draw() は幅と高さの両方を指定します。

項目	Pure Rust版	C++ WASM版
エンジン	Pure Rust 実装	C++ (Emscripten WASM)
依存	image, ab_glyph	serde, serde_json, Node.js
API方式	フィールド設定 + draw + get_image_base64	セッターメソッド + draw（結果を直接返す）
出力取得	get_image_base64(), get_svg(), get_image_mem	draw() が直接返す
提供形態	Pure Rust ライブラリ	WASM バイナリ
価格	22,000円	11,000円

API の主な違い

Pure Rust版:

```
let mut qr = QR::new(FORMAT_PNG);
qr.draw("Hello", 300)?;
let b64 = qr.base_2d.base.get_image_base64()?;
```

C++ WASM版:

```
let qr = QR::new(FORMAT_PNG);
let b64 = qr.draw("Hello", 300)?;
```

項目	要件
Rust	1.70 以降 (edition 2021)
Node.js	16 以降
OS	Windows / macOS / Linux

依存クレート: serde, serde_json（Cargo.toml に記載済み、自動取得）。

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。

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

Barcode.Rust (C++ WASMエンジン) ユーザーズマニュアル

バージョン 1.0 — 2026年2月