Barcode.wasm

ブラウザだけで、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 1.1 — 2026年2月

有限会社 パオ・アット・オフィス

https://www.pao.ac/

目次

---

1. はじめに

1.1 Barcode.wasm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.wasm は、そのバーコードを **ブラウザだけで生成** 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WebAssembly（WASM）にコンパイルしているため、サーバー通信は一切不要。JavaScript / TypeScript から呼び出すだけで、1次元・2次元あわせて **全18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。

たった数行のコードで、目の前にバーコードが現れる。 その手軽さを、ぜひ体感してみてください。

```
const barcode = new module.Jan13();
barcode.setShowText(true);
const result = barcode.draw('491234567890', 300, 100);
document.getElementById('barcode').src = result;
barcode.delete();
```

これだけで、おなじみの13桁商品バーコードが画面に表示されます。

1.2 特長

特長	説明
ブラウザ完結	サーバー通信不要。HTMLファイル1つで動作します
C++譲りの高速描画	WASMによるネイティブ級の処理速度。大量生成にも対応
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
JavaScript / TypeScript	どちらの環境にも対応。npm パッケージ・型定義ファイル付き
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
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	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

バーコード	クラス名	どんなところで使われている？
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
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ド
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2Dコード
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

2. できること

2.1 PNG画像出力 — そのまま表示できるBase64

`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。 `<img>` タグの `src` にそのまま渡すだけ。ファイル保存もサーバー通信も不要です。

```
const barcode = new module.Code128();
barcode.setShowText(true);
const base64 = barcode.draw('Hello123', 400, 100);

// これだけでバーコードが表示される
document.getElementById('barcode').src = base64;
barcode.delete();
```

PNGが向いている場面:

- 画面上でのプレビュー表示
- 固定解像度での画像出力
- あらゆるブラウザでの互換性を重視する場合

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。

```
const barcode = new module.Code128();
barcode.setShowText(true);
barcode.setOutputFormat('svg');
const svg = barcode.draw('Hello123', 400, 100);

// HTMLに直接挿入
document.getElementById('container').innerHTML = svg;
barcode.delete();
```

SVGが向いている場面:

- ラベル印刷（拡大しても劣化しない）
- CSSでバーコードの色やサイズを動的に変更したい場合
- 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 `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、ダウンロードは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。

```
// 紺色のバーにアイボリーの背景
barcode.setForegroundColor(0, 0, 128, 255);
barcode.setBackgroundColor(255, 255, 240, 255);

// 背景を透明にする（ロゴの上に重ねたいときなどに）
barcode.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。

```
barcode.setShowText(true);           // テキストを表示する
barcode.setTextFontScale(1.2);       // 少し大きめに
barcode.setTextGap(0.5);             // バーとの隙間を狭く
barcode.setTextEvenSpacing(true);    // 1文字ずつ均等に配置
```

> **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。

```
barcode.setPxAdjustBlack(-1);  // 黒バーを1px細く（にじみ対策）
barcode.setPxAdjustWhite(1);   // 白スペースを1px広く
```

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。

```
barcode.setFitWidth(true);   // 小数ピクセルを使って幅ぴったりに
barcode.setFitWidth(false);  // 整数ピクセルのみ（若干余白が出ることがある）
```

---

3. 導入方法

導入はとてもシンプルです。必要なのは2つのファイル（`barcode.js` と `barcode.wasm`）だけ。 お好みの方法で、プロジェクトに取り込んでください。

3.1 ダウンロード・インストール

ダウンロード

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

パッケージ	内容	こんな方に
HTML単体版	`barcode_wasm.html`, `barcode.js`, `barcode.wasm`	まずは動かしてみたい方
TypeScript + Vite 版	Viteプロジェクト一式	モダンな開発環境で使いたい方

npm

```
npm install @pao-at-office/barcode-wasm
```

パッケージURL: https://www.npmjs.com/package/@pao-at-office/barcode-wasm

---

3.2 HTML + JavaScript での導入

最もシンプルな方法です。HTMLファイル1つで動きます。

ファイル構成

```
my-project/
├── index.html
└── wasm/
    ├── barcode.js
    └── barcode.wasm
```

index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <title>Barcode.wasm サンプル</title>
```

```
</head>
<body>
    <h1>バーコード生成</h1>
    <img id="barcode" alt="バーコード">

    <script type="module">
        // WASMモジュールをインポート
        import createBarcodeModule from './wasm/barcode.js';

        // モジュールを初期化（非同期）
        const module = await createBarcodeModule({
            locateFile: (path) => {
                if (path.endsWith('.wasm')) {
                    return './wasm/barcode.wasm';
                }
                return path;
            }
        });

        // バーコード生成
        const barcode = new module.Code39();
        barcode.setShowText(true);
        const base64 = barcode.draw('HELLO123', 400, 100);
        document.getElementById('barcode').src = base64;

        // メモリ解放（重要！）
        barcode.delete();
    </script>
</body>
</html>
```

ローカルでの実行方法

> **注意:** HTMLファイルをダブルクリックで直接開くと、セキュリティ制限でWASMを読み込めません。ローカルサーバーを起動してください。

VS Code + Live Server（おすすめ）:

1. VS Codeで「Live Server」拡張機能をインストール
2. HTMLファイルを右クリック →「Open with Live Server」

Python 3:

```
python -m http.server 8080
# → http://localhost:8080
```

Node.js:

```
npx http-server -p 8080
# → http://localhost:8080
```

3.3 TypeScript + Vite での導入

TypeScriptの型安全を活かしたい方向けです。

1. プロジェクト作成

```
npm create vite@latest barcode-demo -- --template vanilla-ts
cd barcode-demo
npm install
```

2. WASMファイルの配置

```
barcode-demo/
├── src/
│   ├── wasm/
│   │   ├── barcode.js      ← ここに配置
│   │   └── barcode.wasm    ← ここに配置
│   └── main.ts
├── index.html
├── package.json
└── vite.config.ts
```

3. vite.config.ts

```
import { defineConfig } from 'vite'

export default defineConfig({
  optimizeDeps: {
    exclude: ['./src/wasm/barcode.js']
  },
  assetsInclude: ['**/*.wasm']
})
```

4. main.ts

```
interface BarcodeModule {
  Code39: new () => Barcode1D;
  // 他のバーコードクラスも同様に定義可能
}
```

```
interface Barcode1D {
  setShowText(show: boolean): void;
  setOutputFormat(format: string): void;
  setTextEvenSpacing(even: boolean): void;
  draw(data: string, width: number, height: number): string;
  delete(): void;
}

async function init(): Promise<void> {
  const Module = await import('./wasm/barcode.js');
  const module: BarcodeModule = await Module.default();

  const barcode = new module.Code39();
  barcode.setShowText(true);
  barcode.setTextEvenSpacing(true);
  barcode.setOutputFormat('png');

  const result = barcode.draw('HELLO123', 400, 100);

  const img = document.getElementById('barcode') as HTMLImageElement;
  img.src = result;

  barcode.delete();
}

document.addEventListener('DOMContentLoaded', init);
```

> **ヒント:** npm パッケージ版には型定義ファイル（`index.d.ts`）が含まれているため、上記のような手動の型定義は不要になります。

5. 開発サーバー起動

```
npm run dev
# → http://localhost:5173
```

3.4 npm パッケージでの導入

npm パッケージを使えば、型定義もついてきます。

```
npm install @pao-at-office/barcode-wasm
```

```
import { initBarcodeModule } from '@pao-at-office/barcode-wasm';

async function generateBarcode(): Promise<void> {
  const module = await initBarcodeModule({
```

```
    wasmPath: '/barcode.wasm'
  });

  const barcode = new module.Code39();
  barcode.setShowText(true);
  barcode.setTextEvenSpacing(true);
  const base64 = barcode.draw('HELLO123', 400, 100);

  document.getElementById('barcode')!.src = base64;

  barcode.delete();  // 忘れずに！
}
```

---

4. クイックスタート — 最初の1本を生成しよう

ここでは、PNG出力とSVG出力のそれぞれで、コピー＆ペーストですぐ動くサンプルを紹介します。

4.1 JavaScript + PNG出力

```
<img id="barcode" alt="Code39">

<script type="module">
import createBarcodeModule from './wasm/barcode.js';
const module = await createBarcodeModule();

const barcode = new module.Code39();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);

const base64 = barcode.draw('HELLO123', 400, 100);
document.getElementById('barcode').src = base64;

barcode.delete();
</script>
```

`draw()` の戻り値は `...` 形式の文字列です。 `<img>` の `src` にそのまま設定できます。

4.2 JavaScript + SVG出力

```
<div id="container"></div>
<a id="download" download="barcode.svg">SVGをダウンロード</a>

<script type="module">
import createBarcodeModule from './wasm/barcode.js';
const module = await createBarcodeModule();

const barcode = new module.Code39();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setOutputFormat('svg');

const svg = barcode.draw('HELLO123', 400, 100);

// SVGは直接HTMLに挿入
document.getElementById('container').innerHTML = svg;

// ダウンロードリンクも作れる
const blob = new Blob([svg], { type: 'image/svg+xml' });
document.getElementById('download').href = URL.createObjectURL(blob);
```

```
barcode.delete();
</script>
```

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

4.3 TypeScript + PNG出力

```typescript
interface BarcodeModule {
  Code39: new () => Barcode1D;
}

interface Barcode1D {
  setShowText(show: boolean): void;
  setOutputFormat(format: string): void;
  setTextEvenSpacing(even: boolean): void;
  draw(data: string, width: number, height: number): string;
  delete(): void;
}

async function generatePNG(): Promise<void> {
  const Module = await import('./wasm/barcode.js');
  const module: BarcodeModule = await Module.default();

  const barcode = new module.Code39();
  barcode.setShowText(true);
  barcode.setTextEvenSpacing(true);

  const base64: string = barcode.draw('HELLO123', 400, 100);

  const img = document.getElementById('barcode') as HTMLImageElement;
  img.src = base64;

  barcode.delete();
}

generatePNG();
```

4.4 TypeScript + SVG出力

```typescript
interface BarcodeModule {
  Code39: new () => Barcode1D;
}

interface Barcode1D {
  setShowText(show: boolean): void;
  setOutputFormat(format: string): void;
  setTextEvenSpacing(even: boolean): void;
  draw(data: string, width: number, height: number): string;
```

```
    delete(): void;
  }

  async function generateSVG(): Promise<void> {
    const Module = await import('./wasm/barcode.js');
    const module: BarcodeModule = await Module.default();

    const barcode = new module.Code39();
    barcode.setShowText(true);
    barcode.setTextEvenSpacing(true);
    barcode.setOutputFormat('svg');

    const svg: string = barcode.draw('HELLO123', 400, 100);

    const container = document.getElementById('container') as HTMLDivElement;
    container.innerHTML = svg;

    // ダウンロードリンクを作成
    const blob = new Blob([svg], { type: 'image/svg+xml' });
    const link = document.getElementById('download') as HTMLAnchorElement;
    link.href = URL.createObjectURL(blob);
    link.download = 'barcode.svg';

    barcode.delete();
  }

  generateSVG();
```

---

5. 実践サンプル集

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。

```
const barcode = new module.Code39();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setShowStartStop(true);     // *HELLO123* と表示
const result = barcode.draw('HELLO123', 400, 100);
barcode.delete();
```

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。

```
const barcode = new module.Code93();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
const result = barcode.draw('Hello123', 400, 100);
barcode.delete();
```

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。

```
const barcode = new module.Code128();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setCodeMode('AUTO');  // AUTO / A / B / C
const result = barcode.draw('Hello123', 400, 100);
barcode.delete();
```

コードモード	説明

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。

```
const barcode = new module.NW7();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setShowStartStop(true);
const result = barcode.draw('A1234567A', 400, 100);
barcode.delete();
```

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。

```
const qr = new module.QR();
qr.setStringEncoding('utf-8');
qr.setErrorCorrectionLevel('M');   // 復元能力: L(7%) / M(15%) / Q(25%) / H(30%)
qr.setVersion(0);                  // 0=自動（データに応じた最小サイズ）
const result = qr.draw('https://www.pao.ac/', 300);
qr.delete();
```

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。

```
const dm = new module.DataMatrix();
dm.setStringEncoding('utf-8');
dm.setCodeSize('AUTO');
dm.setEncodeScheme('AUTO');
const result = dm.draw('Hello World', 200);
dm.delete();
```

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。

```
const pdf = new module.PDF417();
pdf.setStringEncoding('utf-8');
pdf.setErrorLevel(2);
pdf.setColumns(3);
pdf.setAspectRatio(3.0);
pdf.setYHeight(3);
const result = pdf.draw('Hello World', 400);
pdf.delete();
```

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。

```
const gs1 = new module.GS1_128();
gs1.setShowText(true);
gs1.setTextEvenSpacing(true);
const result = gs1.draw('{FNC1}0100012345678905{AI}10ABC123', 500, 120);
gs1.delete();
```

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。

```
const gs1 = new module.GS1_128();
gs1.setShowText(true);
const result = gs1.drawConvenience(
```

```
  '{FNC1}9191234500000000000000452087500401310029500', 500, 150
);
gs1.delete();
```

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。

```
const db = new module.GS1DataBar14();
db.setShowText(true);
db.setSymbolType('OMNIDIRECTIONAL');  // 標準型
// db.setSymbolType('STACKED');           // 二層型
// db.setSymbolType('STACKED_OMNIDIRECTIONAL'); // 標準二層型
const result = db.draw('1234567890128', 200, 80);
db.delete();
```

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。

```
// 一層型
const db = new module.GS1DataBarExpanded();
db.setShowText(true);
db.setSymbolType('UNSTACKED');
const result1 = db.draw('0100012345678905{AI}10ABC123', 400, 80);

// 多層型（スペースが限られる場合）
db.setSymbolType('STACKED');
db.setNoOfColumns(4);
const result2 = db.drawStacked('0100012345678905{AI}10ABC123', 300, 100);

db.delete();
```

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。

```
const yubin = new module.YubinCustomer();
const result = yubin.draw('27500263-29-2-401', 25);
yubin.delete();
```

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（日本の標準的な商品バーコード）
const jan13 = new module.Jan13();
jan13.setShowText(true);
jan13.setExtendedGuard(true);       // ガードバーを長く伸ばす
jan13.setTextEvenSpacing(false);    // セクション別のテキスト配置
const result = jan13.draw('491234567890', 300, 100);
jan13.delete();

// JAN-8（小型商品向け）
const jan8 = new module.Jan8();
jan8.setShowText(true);
jan8.setExtendedGuard(true);
jan8.setTextEvenSpacing(false);
const result2 = jan8.draw('4901234', 200, 100);
jan8.delete();
```

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-A（12桁）
const upcA = new module.UPC_A();
upcA.setShowText(true);
upcA.setExtendedGuard(true);
upcA.setTextEvenSpacing(false);
const result = upcA.draw('01234567890', 300, 100);
upcA.delete();

// UPC-E（8桁、ゼロ圧縮版）
const upcE = new module.UPC_E();
upcE.setShowText(true);
upcE.setExtendedGuard(true);
upcE.setTextEvenSpacing(false);
const result2 = upcE.draw('425261', 200, 100);
upcE.delete();
```

---

6. APIリファレンス

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クラスで使用できるメソッドです。

---

setOutputFormat(format)

出力形式を設定します。

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

```
barcode.setOutputFormat('png');  // PNG（Base64）– デフォルト
barcode.setOutputFormat('svg');  // SVGベクター
```

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	number	赤（0〜255）
g	number	緑（0〜255）
b	number	青（0〜255）
a	number	透明度（0=透明 〜 255=不透明）

```
barcode.setForegroundColor(0, 0, 0, 255);      // 黒（デフォルト）
barcode.setForegroundColor(0, 0, 128, 255);    // 紺色
barcode.setForegroundColor(255, 0, 0, 128);    // 半透明の赤
```

---

setBackgroundColor(r, g, b, a)

背景色を設定します。

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

```
barcode.setBackgroundColor(255, 255, 255, 255);  // 白（デフォルト）
barcode.setBackgroundColor(255, 255, 240, 255);  // アイボリー
barcode.setBackgroundColor(0, 0, 0, 0);          // 透明
```

---

delete()

バーコードオブジェクトのメモリを解放します。**使い終わったら必ず呼んでください。**

```
const barcode = new module.Code39();
// ... バーコード生成 ...
barcode.delete();  // 必須！
```

> **なぜ必要？** WASMオブジェクトはJavaScriptのガベージコレクション対象外です。`delete()` を呼ばないとメモリリークの原因になります。ループで大量生成する場合は特に注意してください。

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メソッドです。

---

draw(data, width, height)

バーコードを生成します。

パラメータ	型	説明
data	string	エンコードするデータ
width	number	画像の幅（px）
height	number	画像の高さ（px）

```
const result = barcode.draw('HELLO123', 400, 100);
```

戻り値:

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

---

setShowText(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	boolean	`true`: 表示 / `false`: 非表示

デフォルト: `true`

---

setTextFontScale(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	number	倍率（0.5〜3.0 推奨）

```
barcode.setTextFontScale(1.0);  // 標準
barcode.setTextFontScale(1.5);  // 1.5倍
barcode.setTextFontScale(0.8);  // 小さめ
```

デフォルト: `1.0`

---

setTextGap(scale)

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
scale	number	倍率（0.0〜3.0 推奨）

```
barcode.setTextGap(1.0);  // 標準
barcode.setTextGap(0.5);  // 狭め
barcode.setTextGap(2.0);  // 広め
```

デフォルト: `1.0`

---

setTextEvenSpacing(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	boolean	`true`: 均等割付 / `false`: 中央寄せ

```
barcode.setTextEvenSpacing(true);   // 1文字ずつ等間隔に配置
barcode.setTextEvenSpacing(false);  // テキストを中央にまとめて配置
```

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 `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(fit)

指定した幅にぴったり収めるかどうかを設定します。

パラメータ	型	説明
fit	boolean	`true`: ぴったり収める / `false`: 整数ピクセルで描画

デフォルト: `true`

> **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。

---

setPxAdjustBlack(px)

黒バーの幅を微調整します。印刷時のにじみ補正に使います。

パラメータ	型	説明
px	number	調整値（ピクセル）

```
barcode.setPxAdjustBlack(0);   // 調整なし（デフォルト）
barcode.setPxAdjustBlack(-1);  // 1px細く（にじみ対策）
barcode.setPxAdjustBlack(1);   // 1px太く
```

デフォルト: `0`

---

setPxAdjustWhite(px)

白スペースの幅を微調整します。

パラメータ	型	説明

パラメータ	型	説明
px	number	調整値（ピクセル）

```
barcode.setPxAdjustWhite(0);   // 調整なし（デフォルト）
barcode.setPxAdjustWhite(1);   // 1px広く
barcode.setPxAdjustWhite(-1);  // 1px狭く
```

デフォルト: `0`

6.3 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(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。

パラメータ	型	説明
show	boolean	`true`: `*HELLO*` / `false`: `HELLO`

デフォルト: `true`

使用例

```
const barcode = new module.Code39();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setShowStartStop(true);
const result = barcode.draw('HELLO123', 400, 100);
barcode.delete();
```

6.4 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（共通メソッドのみ）

使用例

```
const barcode = new module.Code93();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
const result = barcode.draw('Hello123!@#', 400, 100);
barcode.delete();
```

6.5 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(mode)

パラメータ	型	説明
mode	string	`'AUTO'`, `'A'`, `'B'`, `'C'`

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'`

> **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。

使用例

```
const barcode = new module.Code128();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setCodeMode('AUTO');
const result = barcode.draw('Hello123', 400, 100);
barcode.delete();
```

6.6 GS1-128

クラス: `GS1_128` — GS1標準準拠。物流・医療分野で使用

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

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

draw(data, width, height)

通常のGS1-128バーコードを生成します。

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

drawConvenience(data, width, 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。

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

使用例

```
// 通常のGS1-128
const gs1 = new module.GS1_128();
gs1.setShowText(true);
gs1.setTextEvenSpacing(true);
const result = gs1.draw('{FNC1}0100012345678905{AI}10ABC123', 500, 120);
gs1.delete();

// コンビニバーコード
const gs1c = new module.GS1_128();
gs1c.setShowText(true);
const result2 = gs1c.drawConvenience(
  '{FNC1}9191234500000000000000452087500401310029500', 500, 150
);
gs1c.delete();
```

6.7 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`

データ形式: 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 → `A1234567A`

固有メソッド

setShowStartStop(show)

パラメータ	型	説明
show	boolean	`true`: `A1234567A` / `false`: `1234567`

デフォルト: `true`

使用例

```
const barcode = new module.NW7();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
barcode.setShowStartStop(true);
const result = barcode.draw('A1234567A', 400, 100);
barcode.delete();
```

6.8 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

使用例

```
const barcode = new module.Matrix2of5();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
const result = barcode.draw('1234567890', 400, 100);
barcode.delete();
```

6.9 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

使用例

```
const barcode = new module.NEC2of5();
barcode.setShowText(true);
barcode.setTextEvenSpacing(true);
const result = barcode.draw('1234567890', 400, 100);
barcode.delete();
```

6.10 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(extend)

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。

パラメータ	型	説明
extend	boolean	`true`: 拡張（標準外観） / `false`: フラット

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、テキストはセクション別に配置
`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

```
const barcode = new module.Jan8();
barcode.setShowText(true);
barcode.setExtendedGuard(true);
barcode.setTextEvenSpacing(false);
const result = barcode.draw('4901234', 200, 100);
barcode.delete();
```

6.11 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(extend)

6.10 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

```
const barcode = new module.Jan13();
barcode.setShowText(true);
barcode.setExtendedGuard(true);
barcode.setTextEvenSpacing(false);
const result = barcode.draw('491234567890', 300, 100);
barcode.delete();
```

6.12 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(extend)

6.10 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。

使用例

```
const barcode = new module.UPC_A();
barcode.setShowText(true);
barcode.setExtendedGuard(true);
barcode.setTextEvenSpacing(false);
const result = barcode.draw('01234567890', 300, 100);
barcode.delete();
```

6.13 UPC-E

クラス: `UPC_E` — UPC-Aの短縮版（8桁）。小型商品用

入力: 数字6桁（チェックディジットは自動計算）

固有メソッド

setExtendedGuard(extend)

6.10 JAN-8 と同じです。

デフォルト: `true`

> **UPC-Eの特徴:** UPC-Aの「ゼロ圧縮」版です。拡張ガードバー有効時、先頭のナンバーシステム（0）とチェックディジットが左右に表示されます。

使用例

```
const barcode = new module.UPC_E();
barcode.setShowText(true);
barcode.setExtendedGuard(true);
barcode.setTextEvenSpacing(false);
const result = barcode.draw('425261', 200, 100);
barcode.delete();
```

6.14 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(type)

値	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

デフォルト: `'OMNIDIRECTIONAL'`

getSymbolType()

現在のシンボルタイプを数値で取得します（0: Omni, 1: Stacked）。

encode(data)

データをエンコードします。戻り値: `boolean`（成功/失敗）

calculateCheckDigit(data) — 静的メソッド

13桁のデータからチェックディジットを計算し、14桁のデータを返します。

```
const dataWithCheck = module.GS1DataBar14.calculateCheckDigit('0123456789012');
```

使用例

```
const barcode = new module.GS1DataBar14();
barcode.setShowText(true);
barcode.setSymbolType('OMNIDIRECTIONAL');
const result = barcode.draw('1234567890128', 200, 80);
barcode.delete();
```

6.15 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 8〜13桁（**先頭桁は0または1のみ**）

固有メソッド: なし

使用例

```
const barcode = new module.GS1DataBarLimited();
barcode.setShowText(true);
```

```
const result = barcode.draw('0123456789012', 200, 60);
barcode.delete();
```

6.16 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(type)

値	説明
`'UNSTACKED'`	一層型
`'STACKED'`	多層型（スペースが限られる場合に）

デフォルト: `'UNSTACKED'`

setNoOfColumns(columns)

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`

drawStacked(data, width, height)

多層型バーコードを生成します。

使用例

```
// 一層型
const db = new module.GS1DataBarExpanded();
db.setShowText(true);
db.setSymbolType('UNSTACKED');
const result1 = db.draw('0100012345678905{AI}10ABC123', 400, 80);

// 多層型
db.setSymbolType('STACKED');
db.setNoOfColumns(4);
const result2 = db.drawStacked('0100012345678905{AI}10ABC123', 300, 100);

db.delete();
```

6.17 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(data, 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。

パラメータ	型	説明
data	string	郵便番号 + 住所表示番号
height	number	画像の高さ（px）

> **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()` は使用可能です。

使用例

```
const barcode = new module.YubinCustomer();
const result = barcode.draw('27500263-29-2-401', 25);
barcode.delete();
```

6.18 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(data, size)

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

setStringEncoding(encoding)

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'`

setErrorCorrectionLevel(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(version)

QRコードのバージョン（セルの数 = サイズ）を指定します。

パラメータ	型	説明
version	number	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）

setEncodeMode(mode)

値	説明
`'AUTO'`	データ内容に応じて自動選択
`'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(fit)

指定サイズにフィットさせるかどうか。

デフォルト: `false`

使用例

```
const qr = new module.QR();
qr.setStringEncoding('utf-8');
qr.setErrorCorrectionLevel('M');
qr.setVersion(0);
qr.setFitWidth(true);
```

```
qr.setOutputFormat('svg');
const result = qr.draw('https://www.pao.ac/', 300);
qr.delete();
```

6.19 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(data, size)

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

setStringEncoding(encoding)

`'utf-8'`（デフォルト）または `'shift-jis'`

setCodeSize(size)

シンボルのセル数を指定します。

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(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

値	説明
`'EDIFACT'`	EDIFACT
`'BASE256'`	バイナリ

デフォルト: `'AUTO'`

setFitWidth(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);
```

使用例

```
const dm = new module.DataMatrix();
dm.setStringEncoding('utf-8');
dm.setCodeSize('AUTO');
dm.setEncodeScheme('AUTO');
dm.setFitWidth(true);
dm.setOutputFormat('svg');
const result = dm.draw('Hello World', 200);
dm.delete();
```

6.20 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(data, width)

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

setStringEncoding(encoding)

`'utf-8'`（デフォルト）または `'shift-jis'`

setErrorLevel(level)

レベル	訂正能力
0	最小
1	低
2	標準（おすすめ）
3〜8	高〜最大

デフォルト: `2`

setColumns(columns)

列数。0=自動、1〜30で指定。 **デフォルト:** `0`

setRows(rows)

行数。0=自動、3〜90で指定。 **デフォルト:** `0`

setAspectRatio(ratio)

縦横比。1.0〜10.0。 **デフォルト:** `3.0`

setYHeight(height)

Y方向の高さ係数。1〜10。 **デフォルト:** `3`

setFitWidth(fit)

デフォルト: `false`

使用例

```
const pdf = new module.PDF417();
pdf.setStringEncoding('utf-8');
pdf.setErrorLevel(2);
pdf.setColumns(4);
pdf.setRows(0);
pdf.setAspectRatio(3.0);
pdf.setYHeight(3);
pdf.setFitWidth(true);
pdf.setOutputFormat('svg');
const result = pdf.draw('Hello World', 400);
pdf.delete();
```

7. 動作環境

対応ブラウザ

WebAssemblyに対応したモダンブラウザで動作します。

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
Opera	44 以降

> 2024年時点で、世界のブラウザの97%以上がWebAssemblyに対応しています。

開発環境（TypeScript版）

項目	要件
Node.js	16 以降
npm	8 以降
バンドラー	Vite（推奨）、Webpack 等

ファイルサイズ

ファイル	サイズ（目安）
barcode.wasm	約1.5MB
barcode.js	約50KB

> **ヒント:** `.wasm` ファイルはgzip圧縮と相性が良く、転送時のサイズはさらに小さくなります。

---

8. ライセンス・お問い合わせ

ライセンス

Barcode.wasm 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.wasm/
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 ユーザーズマニュアル

バージョン 1.1 — 2026年2月
