

目次

[1. はじめに](#)

[2. 動作環境の構築方法](#)

[2-1. 前提条件の確認と作業フォルダの作成](#)

[2-2. ポート 8080 に法人 API サーバを起動する](#)

[インストール](#)

[サーバの起動と終了](#)

[動作確認](#)

[2-3. ポート 3474 にデモ用の HTTP サーバを起動する](#)

[インストール](#)

[サーバの起動と終了](#)

[動作確認](#)

[3. 動作確認用サンプルの利用方法](#)

[3-1. 基本的な利用方法](#)

[Step1: 初期画面](#)

[Step2: 正常系サンプル](#)

[Step3: 異常系サンプル](#)

[Step4: 法人情報のロード](#)

[3-2. 応用的トピック](#)

[スキーマの切り替え](#)

[詳細なエラー](#)

1. はじめに

本ドキュメントは「GIF コンポーネントツールの動作確認用サンプル」の動作環境の構築方法および利用方法を説明するものです。

対象読者はコマンドラインを通じて動作環境を構築する構築担当者、 および、ブラウザを通じて動作確認用サンプルを体験する利用者です。

対象とする動作環境は Linux Bash および Windows 10/11 PowerShell です。

2. 動作環境の構築方法

2-1. 前提条件の確認と作業フォルダの作成

それぞれの動作環境で以下の前提条件が満たされていることを確認します。

(Bash)

- Node.js (v18 以上) および npm がインストールされていること
- PC の 3474 ポート、8080 ポートが使用可能であること
- 適当な作業フォルダ `${WORK}` を用意しておくこと
- `${SOMEWHERE}` フォルダに以下の 2 ファイルが格納されていること

```
dev-gif-component-tools-demo-1.0.0.tgz
dev-gif-component-tools-legal-entity-1.0.0.tgz
```

(PowerShell)

- PowerShell 7.x がインストールされていること [参考 1]
- Windows 用の Node.js (v18 以上) および npm がインストールされていること [参考 2]
- PC の 3474 ポート、8080 ポートが使用可能であること
- 作業フォルダ `c:\demo` を用意しておくこと (フォルダ名は任意です、適宜読み替えてください)
- `c:\demo` フォルダに以下の 2 ファイルが格納されていること

```
dev-gif-component-tools-demo-1.0.0.tgz
dev-gif-component-tools-legal-entity-1.0.0.tgz
```

※ 参考 1: **Windows への PowerShell のインストール - PowerShell | Microsoft Learn**, <https://learn.microsoft.com/ja-jp/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.5>, 2025-01-29 閲覧

※ 参考 2: **Node.js — Node.js® をダウンロードする**, <https://nodejs.org/ja/download>, 2025-01-29 閲覧

2-2. ポート 8080 に法人 API サーバを起動する

インストール

それぞれの動作環境で以下の手順で環境構築します。

(Bash)

```
$ cd ${WORK}
$ tar xvfz ${SOMEWHERE}/dev-gif-component-tools-legal-entity-1.0.0.tgz
$ cd dev-gif-component-tools-legal-entity-1.0.0
$ npm install
$ npm run setup
$
```

(PowerShell)

```
PS C:\Users\John> cd C:\demo
PS C:\demo> tar xvfz .\dev-gif-component-tools-legal-entity-1.0.0.tgz
PS C:\demo> cd .\dev-gif-component-tools-legal-entity-1.0.0
PS C:\demo\dev-gif-component-tools-legal-entity-1.0.0> npm install
PS C:\demo\dev-gif-component-tools-legal-entity-1.0.0> Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
PS C:\demo\dev-gif-component-tools-legal-entity-1.0.0> .\tools\setup.ps1
PS C:\demo\dev-gif-component-tools-legal-entity-1.0.0>
```

- `npm run setup` の代わりに、`.\tools\setup.ps1` を実行するのが相違点です
- 通常 PowerShell ではローカルの `*.ps1` ファイルを実行できないように権限が制限されており、これを緩和するために `Set-ExecutionPolicy -Scope CurrentUser RemoteSigned` を実行しています

サーバの起動と終了

それぞれの動作環境で以下の手順でサーバを起動します。

(Bash)

```
$ cd ${WORK}/dev-gif-component-tools-legal-entity-1.0.0
$ npm start

> @dev-gif-component-tools/legal-entity@1.0.0 start
> node bin/server.js 8080

@gif-component-tools/legal-entity server is running on port 8080
```

サーバを停止する場合には `Ctrl-C` を押下します。

(PowerShell)

```
PS C:\Users\John> cd C:\demo\dev-gif-component-tools-legal-entity-1.0.0
PS C:\demo\dev-gif-component-tools-legal-entity-1.0.0> npm start

> @dev-gif-component-tools/legal-entity@1.0.0 start
> node bin/server.js 8080

@gif-component-tools/legal-entity server is running on port 8080
```

サーバを停止する場合には Ctrl-C を押下します。

動作確認

サーバが起動した状態でローカル PC のブラウザで <http://localhost:8080/> にアクセスし、 以下のような画面が表示されれば成功です。



2-3. ポート 3474 にデモ用の HTTP サーバを起動する

インストール

それぞれの動作環境で以下の手順で環境構築します。

(Bash)

```
$ cd ${WORK}
$ tar xvfz ${SOMEWHERE}/dev-gif-component-tools-demo-1.0.0.tgz
$ cd dev-gif-component-tools-demo-1.0.0
$ npm install
$
```

(PowerShell)

```
PS C:\Users\John> cd C:\demo
PS C:\demo> tar xvfz .\dev-gif-component-tools-demo-1.0.0.tgz
PS C:\demo> cd .\dev-gif-component-tools-demo-1.0.0
PS C:\demo\dev-gif-component-tools-demo-1.0.0> npm install
PS C:\demo\dev-gif-component-tools-demo-1.0.0>
```

サーバの起動と終了

それぞれの動作環境で以下の手順でサーバを起動します。

(Bash)

```
$ cd ${WORK}/dev-gif-component-tools-demo-1.0.0
$ npm start

> @dev-gif-component-tools/demo@1.0.0 start
> superstatic dist

Superstatic started.
Visit http://localhost:3474 to view your app.
```

サーバを停止する場合には `Ctrl-C` を押下します。

(PowerShell)

```
PS C:\Users\John> cd C:\demo\dev-gif-component-tools-demo-1.0.0
PS C:\demo\dev-gif-component-tools-demo-1.0.0> npm start

> @dev-gif-component-tools/demo@1.0.0 start
> superstatic dist

Superstatic started.
Visit http://localhost:3474 to view your app.
```

サーバを停止する場合には `Ctrl-C` を押下します。

動作確認

サーバが起動した状態でローカル PC のブラウザで <http://localhost:3474/> にアクセスし、 以下のような画面が表示されれば成功です。

GIF コンポーネントツール動作確認デモ

スキーマ ----

JSON Schema [all.semantic.schema.json](#) の準備が完了しました 5346[msec]

Input

サンプル ---- 法人番号

1430001000005 読み込み

ここに JSON を入力します。サンプルや法人番号から読み込むこともできます。

検証する 正規化する 整形する

Output

1 | {}

3. 動作確認用サンプルの利用方法

3-1. 基本的な利用方法

Step1: 初期画面

ウェブブラウザで <http://localhost:3474/> を開くことで、 以下のような GUI が表示されます。

GIF コンポーネントツール動作確認デモ

JSON Schema [all.semantic.schema.json](#) の準備が完了しました 5346[msec]

スキーマ -----

Input

サンプル ----- 法人番号 1430001000005 読み込み

ここに JSON を入力します。サンプルや法人番号から読み込むこともできます。

検証する 正規化する 整形する

Output

1 | {}

※ 実行環境によっては起動～ GUI 表示までに時間がかかる場合があります

Input 欄に任意の GIF (JSON) を入力し、【検証する】【正規化する】ボタンを押すと Output 欄に検証・正規化の結果が表示される、というのが基本動作です。エラーが発生する場合には Output 欄の下部に **詳細なエラー** が表示されます。

Step2: 正常系サンプル

【サンプル】プルダウンからさまざまなサンプルデータをロードすることができます。

以下は先頭のサンプルを選択した場合です。Input 欄に JSON が転記されます。

GIF コンポーネントツール動作確認デモ

JSON Schema [all.semantic.schema.json](#) の準備が完了しました 5346[msec]

Input

```
{  "idGroup": {    "idType": "職員番号",    "id": "12334567"  },  "familyName": "山田",  "givenName": "花子",  "familyNameKana": "ヤマダ",  "givenNameKana": "ハナコ",  "familyNameEn": "Yamada",  "givenNameEn": "Hanako"}
```

サンプル

文字種 (Person - 正常系)

文字種 (Person - 正常系)
文字種 (Person - 異常系 - 英字の半角化・カナの全角化)
文字種 (Person - 異常系 - ひらがなのカナ化)
文字種 (Person - 異常系 - 結合文字)
文字種 (Person - 異常系 - 複合系)
日付 (Person - 正常系 - yyyy-mm-dd)
日付 (Person - 異常系 - yyyy年m月d日)
日付 (Person - 異常系 - yyyy-mm-dd 全角)
日付 (Person - 異常系 - yyyy年m月d日 全角)
日付 (Person - 異常系 - 和暦)
日付 (Person - 異常系 - 和暦全角)
電話番号 (ContactPoint - 国内固定電話:正常系)
電話番号 (ContactPoint - 国内固定電話:非正規化形式1)
電話番号 (ContactPoint - 国内固定電話:非正規化形式2)
電話番号 (ContactPoint - 国内固定電話:全角数字)
電話番号 (ContactPoint - グローバル:複合)
電話番号 (ContactPoint - 内線番号あり:複合)
電話番号 (ContactPoint - その他)
コード類 (Person - 正常系:性別・婚姻・有無・該当区分・国コード)

法人番号

読み込み

検証

Output

1 | {}

この状態で【検証する】ボタンを押すことで Output 欄に検証結果 この JSON は valid です が表示されます。

GIF コンポーネントツール動作確認デモ

JSON Schema [all.semantic.schema.json](#) の準備が完了しました 5346[msec]

Input

```
{  "idGroup": {    "idType": "職員番号",    "id": "12334567"  },  "familyName": "山田",  "givenName": "花子",  "familyNameKana": "ヤマダ",  "givenNameKana": "ハナコ",  "familyNameEn": "Yamada",  "givenNameEn": "Hanako"}
```

検証する

正規化する

整形する

Output

```
1 // この JSON は valid です
2 {
3   "idGroup": {
4     "idType": "職員番号",
5     "id": "12334567"
6   },
7   "familyName": "山田",
8   "givenName": "花子",
9   "familyNameKana": "ヤマダ",
10  "givenNameKana": "ハナコ",
```

Step3: 異常系サンプル

【サンプル】から異常系サンプルを選択し、【検証する】ボタンを押すと Output 欄に検証結果 この JSON は valid ではありません が表示されます。また、Output の下部には 詳細なエラー が表示されます。

GIF コンポーネントツール動作確認デモ

JSON Schema [all.semantic.schema.json](#) の準備が完了しました 5346[msec]

スキーマ -----

Input

サンプル ----- 法人番号 1430001000005 読み込み

```
{
  "legalEntityNumber": "1430001000005",
  "tradeName": "株式会社 0 0 0 5",
  "categoryOfOrganization": "301",
  "contactPointInformation": {
    "contactPointUrl": "https://www.houjin-bangou.nta.go.jp/henkorireki-johoto.html?selHouzinNo=1430001000005"
  },
  "positionOfOrganizationTypeInName": "前株",
  "registeredAddress_MP04": {
    "localGovernmentCode": "011011",
    "prefecture": "北海道",
    "cityAndCounty": "札幌市中央区",
    "streetAddressAndCityBlock": "北一条西 2 丁目 3 番 3 2 号"
  }
}
```

検証する 正規化する 整形する

Output

```
1 // この JSON は valid です
2 {
3   "legalEntityNumber": "1430001000005",
4   "tradeName": "株式会社 0 0 0 5",
5   "categoryOfOrganization": "301",
6   "contactPointInformation": {
7     "contactPointUrl": "https://www.houjin-bangou.nta.go.jp/henkorireki-johoto.html?selHouzinNo=1430001000005"
8   },
9 }
```

なお、ここでロードされる法人情報は <http://localhost:8080/> で起動されている LegalEntity WebAPI から入手可能なものに限りです。

3-2. 応用的トピック

スキーマの切り替え

画面右上【スキーマ】プルダウンでは、使用するスキーマを切り替えることができます。

デフォルトは all ですが、これは GIF のすべてのクラスの JSON を検査することができる JSON Schema [参考 3] です。汎用性は高いですがサイズが大きく、ウェブブラウザでロードした後に操作できるようになるまで時間がかかる場合があります。

Event や Person といったスキーマは、そのクラス専用に整備されたスキーマです。サイズは最小限で比較的ロードに時間がかかりません。検証対象のクラスが事前に決まっているユースケースでは有効です。

※ 参考 3: JSON Schema, <https://json-schema.org/>, 2025-01-29 閲覧

詳細なエラー

この動作確認用サンプルは与えられた JSON を JSON Schema および Ajv JSON schema validator [参考 4] で判定し、その結果に応じて判定結果の提示や正規化を行う仕組みです。与えられた JSON が JSON Schema に準拠しない場合には Ajv がエラー群を生成します。

※ 参考 4: **Ajv JSON schema validator**, <https://ajv.js.org/>, 2025-01-29 閲覧

以下は 本来は全角カナが求められる箇所に半角カナを入力した場合 のエラーメッセージの抜粋です。

```
[
  {
    "instancePath": "/familyNameKana",
    "keyword": "format",
    "params": {
      "format": "familyNameKana"
    },
    "message": "¥"familyNameKana¥形式に揃えなければならない",
    "error": "半角カナは使用できません"
  }
]
```

- Ajv error-objects [参考 5] をベースとしています
- ajv-18n [参考 6] を用いて message が日本語化されています
- error は本コンポーネントの機能によって付与された独自の詳細なエラーメッセージです

※ 参考 5: **API Reference | Ajv JSON schema validator # Error Objects**,
<https://ajv.js.org/api.html#error-objects>, 2025-01-29 閲覧

※ 参考 6: **ajv-i18n | Ajv JSON schema validator**, <https://ajv.js.org/packages/ajv-i18n.html/>, 2025-01-29 閲覧

以下は 本来は全角カナが求められる箇所に半角カナを入力し、正規化を有効にした場合 のエラーメッセージの抜粋です。

```
[
  {
    "instancePath": "/familyNameKana",
    "schemaPath": "#/anyOf/0/properties/familyNameKana/format",
    "keyword": "format",
    "params": {
      "format": "familyNameKana"
    },
    "message": "¥"familyNameKana¥形式に揃えなければならない",
    "error": "半角カナは使用できません",
    "prev": "ヤマダ",
    "next": "ヤマダ"
  }
]
```

- prev および next は本コンポーネントの機能によって付与された正規化前後の文字列です
- 半角カナから全角カナへの変換は本コンポーネントの正規化機能によって実現されています

なお、詳細なエラーは人間可読のためのものではなく、機械可読性のためのものであることに注意してください。人間視点では冗長・過剰なエラーが提示される傾向があります。

たとえば以下はサンプル バリデーション (CodeInformationModel - 浅い - 異常系 : 余計なプロパティ) ですが、これは CodeInformationModel としては description が余計であるためエラーとなります。

```
{
  "codeType": "NDC",
  "code": "3",
  "description": "社会科学"
}
```

しかし、これを all スキーマによって検査するとバリデーションエラーは実に 81 件に上ります。

```
// 81 件のエラーを表示しています
[
  {
    "instancePath": "",
    "keyword": "additionalProperties",
    "params": {
      "additionalProperty": "codeType"
    },
    "message": "追加してはいけない"
  },
  {
    "instancePath": "",
    "keyword": "additionalProperties",
    "params": {
      "additionalProperty": "code"
    },
    "message": "追加してはいけない"
  },
  {
    "instancePath": "",
    "keyword": "additionalProperties",
    "params": {
      "additionalProperty": "description"
    },
    "message": "追加してはいけない"
  },
  (中略)
  {
    "instancePath": "",
    "keyword": "anyOf",
    "params": {},
    "message": "¥"anyOf¥のスキーマとマッチしなくてはならない"
  }
]
```

- `CodeInformationModel` としては `description` が余計なのでエラー
- その他のすべてのクラス (たとえば `Person` や `LegalEntity`) としては `code` や `codeType` が余計、加えてそのクラスの必須プロパティの不足によりエラー
- 全体としてはいずれの分岐でも成功する条件がないのでエラー

このような判定によりエラーが蓄積された上でレポートされる仕組みです。なお、**スキーマの切り替え**によって `CodeInformationModel` 専用のスキーマがロードされている場合には、以下のようにコンパクトなエラーが報告されます。

```
// 1 件のエラーを表示しています
[
  {
    "instancePath": "",
    "schemaPath": "#/definitions/pd:CodeInformationModel/additionalProperties",
    "keyword": "additionalProperties",
    "params": {
      "additionalProperty": "description"
    },
    "message": "追加してはいけない"
  }
]
```

(以上)