# 目次

# 改訂情報

| 変更年月日 | 変更内容 |
| --- | --- |
| 2014-08-01 | 初版 |
| 2017-12-01 | 第2版 下記を追加しました<br>■ 「*概要*」 - 「*モジュール*」を追加 |
| 2019-08-01 | 第3版 下記を追加しました<br>■ 「*郵便番号データのインポート*」 - 「*住所の郵便番号データインポート　実行パラメータ*」 のクレンジングパターンに no cleansing を追加<br>■ 「*郵便番号データのインポート*」 - 「*事業所の個別郵便番号データインポートの実行パラメータ*」 のクレンジングパターンに no cleansing を追加 |

# はじめに

## 本書の目的

本書では intra-mart Accel Platform で提供する郵便番号検索機能の仕様とそのプログラミング方法や注意点について説明します。

## 対象読者

本書は、以下の条件を満たす人を対象としています。

- intra-mart Accel Platform を理解している
- 郵便番号検索機能を利用したアプリケーションの開発者
- 郵便番号検索機能を利用したアプリケーションの利用者

## 本書の構成

本書は、以下のような内容で構成されています。

- *概要*

郵便番号検索について説明しています。

- *郵便番号データのインポート*

郵便番号データのインポート方法について説明します。

- *郵便番号検索の利用方法*

郵便番号検索機能を利用したプログラミング方法について説明します。

# 概要

項目



## 郵便番号検索機能とは

郵便番号検索機能とは、日本郵便株式会社の提供している郵便番号データを取り込み、郵便番号や住所の一部から住所を検索するための機能です。
取り込む郵便番号データについては以下を参照してください。

- 住所の郵便番号
- 事業所の個別郵便番号

取り込んだ郵便番号データの検索は、 intra-mart Accel Platform 上に構築したアプリケーションや外部アプリケーションから利用できます。

## 仕様

郵便番号検索機能を利用するには、はじめに日本郵便株式会社のサイトより郵便番号データをダウンロードし、 intra-mart Accel Platform へインポートする必要があります。
郵便番号データのインポートはインポート用のジョブから行います。詳しくは以下を参照してください。

- *郵便番号データのインポート*

郵便番号データのインポートを行ったら、以下のURLへリクエストを送信すると検索結果を取得できます。検索結果はJSONで返却されます。

`http://<HOST>:<PORT>/<CONTEXT_PATH>/api/zipcode/search`

送信するリクエストのパラメータや、検索結果のJSON形式については以下を参照してください。

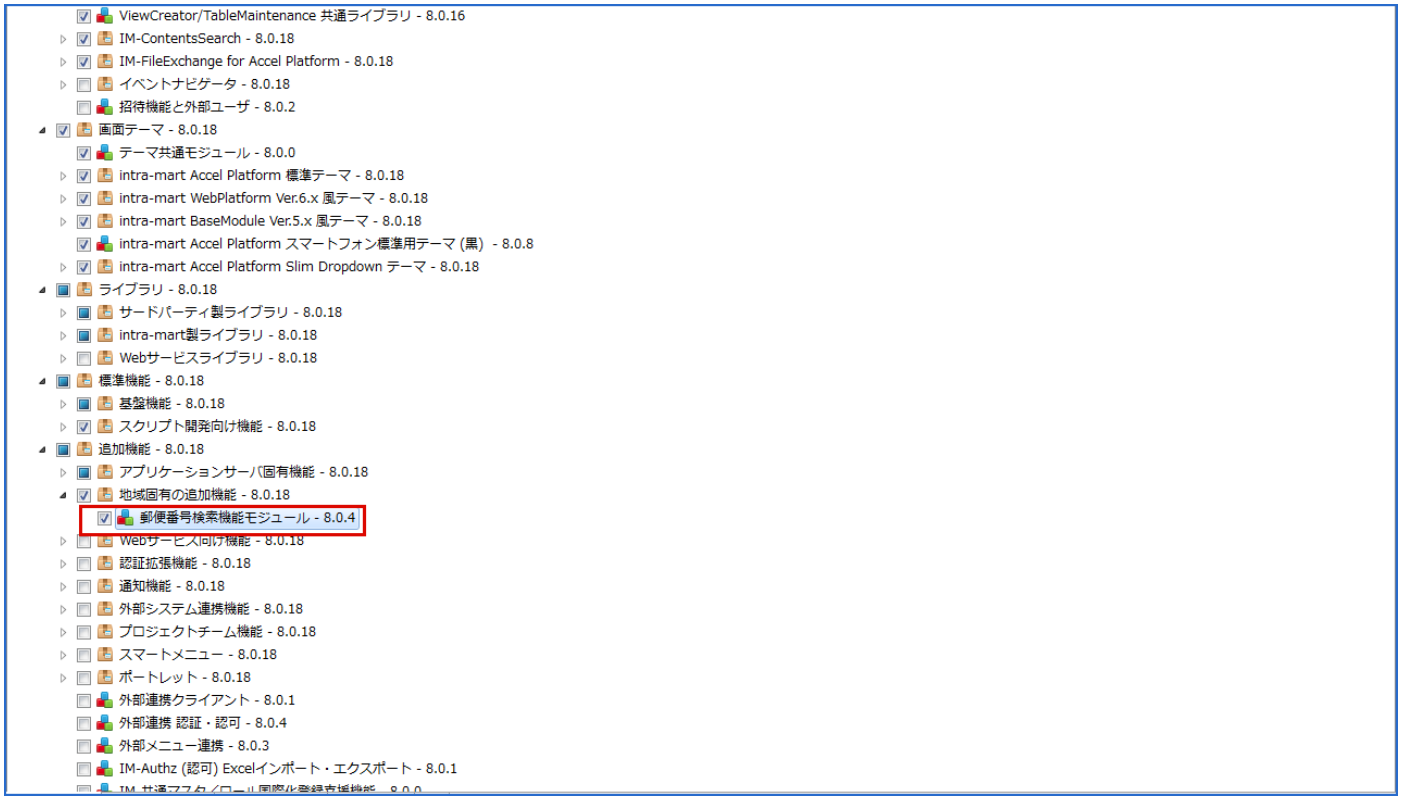
- *インポートした郵便番号データの検索方法*

郵便番号検索を利用できるユーザを制限したい場合は、認可設定を行う事で利用可能ユーザを設定できます。
対象のリソースは以下の通りです。

`画面・処理 - 郵便番号検索 - 郵便番号検索`

## モジュール

郵便番号検索機能を利用するには、下記のモジュールが必要です。




## 制限事項

制限事項はリリースノートに記載されています。必ず制限事項を確認してください。

# 郵便番号データのインポート

項目



## 住所の郵便番号データインポート

intra-mart Accel Platform に住所の郵便番号データをインポートする場合の手順は以下の通りです。

1. 郵便番号データ（ken_all.zip）をダウンロードし解凍する。
2. テナント管理者でログインしファイル操作画面より、解凍した郵便番号データファイル（KEN_ALL.CSV） をパブリックストレージへ配置する。
3. ジョブネット設定画面より「住所の郵便番号データインポート」を実行し、郵便番号データをインポートする。

### ステップ1：郵便番号データ（ken_all.zip）をダウンロードし解凍する。

1. 住所の郵便番号 ページより郵便番号データをダウンロードします。
2. 「都道府県一覧 - 全国一括」をクリックし「ken_all.zip」をダウンロードします。
   （特定の都道府県の郵便番号データのみ利用する場合は、利用する都道府県の郵便番号データをダウンロードしてください。）
3. ダウンロードした圧縮ファイル （ken_all.zip） を任意のフォルダに解凍します。

### ステップ2：郵便番号データファイル（KEN_ALL.CSV） をパブリックストレージへ配置する。

1. intra-mart Accel Platform のログイン画面を開き、 テナント管理者 でログインします。
2. サイトマップより「テナント管理 - ファイル操作」をクリックします。
3. ファイル操作画面よりルート・ディレクトリ（テナントIDのディレクトリ）に解凍したCSVファイル （KEN_ALL.CSV） をアップロードします。

### ステップ3：住所の郵便番号データインポートを実行する。

1. サイトマップより「テナント管理 - ジョブ管理 - ジョブネット設定」をクリックします。
2. ジョブネット管理画面より「郵便番号データインポート - 住所の郵便番号データインポート」をクリックします。
3. ジョブネット情報の「このジョブネットを編集する」ボタンをクリックして編集画面を表示します。
4. トリガ設定 「繰り返し指定」トリガの「新規登録」ボタンをクリックし、「1回だけ実行する」を選択して「決定」ボタンをクリックします。
5. スケジュールが登録されたら、「有効」のチェックボックスにチェックをいれ、「この内容でジョブネットを更新する」ボタンをクリックします。

> **注意**
>
> このジョブは、実行する時に住所の郵便番号データを一旦クリアします。

## 住所の郵便番号データインポート　実行パラメータ

住所の郵便番号データインポートでは、以下の実行パラメータを使用できます。

| パラメータ名 | 必須項目 | デフォルト値 | 備考 |
|---|---|---|---|
| plugin_id | ○ | jp.co.intra_mart.import.ZipCodeImporter | インポータのIDです。（このパラメータは変更しないでください。） |
| file | ○ | なし | インポートファイルのファイル名を指定します。 |
| encoding | × | SHIFT-JIS | インポートファイルのエンコーディングを指定します。 |
| cleansing | × | none | インポートデータのクレンジングパターンを指定します。 |

クレンジングパターンは標準では以下のパターンが提供されています。

- none

  半角カナを全角カナへ変換します。それ以外のデータの加工は行いません。

- no cleansing

  データの加工を行いません。半角カナのままインポートします。

- simple

  半角カナを全角カナへ変換後、「以下に掲載がない場合」等の特定の文字列を除きます。
  このクレンジングパターンで除かれる文字列は以下のファイルで定義されています。

  %PUBLIC_STORAGE_PATH%/im_zip_code/simple_cleansing.txt

- split

  simpleパターンまでの変換を行った後、カンマ区切りでまとめられている町域名を分割します。
  例えば、「北海道美唄市上美唄町（協和、南）」の場合、「（協和、南）」が分割され、「北海道美唄市上美唄町協和」と「北海道美唄市上美唄町南」の2件のデータが登録されます。

- omit_parentheses

  simpleパターンまでの変換を行った後、町域名の括弧「（」以降を省きます。
  例えば、「北海道美唄市上美唄町（協和、南）」の場合、「北海道美唄市上美唄町」に書き換えます。

クレンジング処理の拡張方法については、 *住所の郵便番号データインポートのクレンジング処理拡張方法* を参照してください。

## 事業所の個別郵便番号データインポート

intra-mart Accel Platform に事業所の個別郵便番号データをインポートする場合の手順は以下の通りです。

1. 郵便番号データ（jigyosyo.zip）をダウンロードし解凍する。
2. テナント管理者でログインしファイル操作画面より、解凍した郵便番号データファイル（JIGYOSYO.CSV） をパブリックストレージへ配置する。
3. ジョブネット設定画面より、「住所の個別郵便番号データインポート」を実行し、郵便番号データをインポートする。

### ステップ1：郵便番号データ（jigyosyo.zip）をダウンロードし解凍する。

1. 事業所の個別郵便番号 ページより郵便番号データをダウンロードします。
2. 「最新データのダウンロード」をクリックし「jigyosyo.zip」をダウンロードします。
3. ダウンロードした圧縮ファイル （jigyosyo.zip） を任意のフォルダに解凍します。

### ステップ2：郵便番号データファイル（jigyosyo.zip） をパブリックストレージへ配置する。

1. intra-mart Accel Platform のログイン画面を開き、 テナント管理者 でログインします。
2. サイトマップより「テナント管理 - ファイル操作」をクリックします。
3. ファイル操作画面よりルート・ディレクトリ（テナントIDのディレクトリ）に解凍したCSVファイル （JIGYOSYO.CSV） をアップロードします。

### ステップ3：事業所の個別郵便番号データインポートを実行する。

1. サイトマップより「テナント管理 - ジョブ管理 - ジョブネット設定」をクリックします。
2. ジョブネット管理画面より「郵便番号データインポート - 事業所の個別郵便番号データインポート」をクリックします。
3. ジョブネット情報の「このジョブネットを編集する」ボタンをクリックして編集画面を表示します。
4. トリガ設定 「繰り返し指定」トリガの「新規登録」ボタンをクリックし、「1回だけ実行する」を選択して「決定」ボタンをクリックします。
5. スケジュールが登録されたら、「有効」のチェックボックスにチェックをいれ、「この内容でジョブネットを更新する」ボタンをクリックします。

> **注意**
>
> このジョブは、実行する時に事業所の個別郵便番号データを一旦クリアします。

## 事業所の個別郵便番号データインポートの実行パラメータ

住所の郵便番号データインポートでは、以下の実行パラメータを使用できます。

| パラメータ名 | 必須項目 | デフォルト値 | 備考 |
|---|---|---|---|
| plugin_id | ○ | jp.co.intra_mart.import.OfficeZipCodeImporter | インポータのIDです。（このパラメータは変更しないでください。） |
| file | ○ | なし | インポートファイルのファイル名を指定します。 |
| encoding | × | SHIFT-JIS | インポートファイルのエンコーディングを指定します。 |
| cleansing | × | none | インポートデータのクレンジングパターンを指定します。 |

クレンジングパターンは標準では以下のパターンが提供されています。

- none

  半角カナを全角カナへ変換します。それ以外のデータの加工は行いません。

- no cleansing

  データの加工を行いません。半角カナのままインポートします。

クレンジング処理の拡張方法については、 *事業所の個別郵便番号データインポートのクレンジング処理拡張方法* を参照してください。

# 郵便番号検索の利用方法

項目



## インポートした郵便番号データの検索方法

郵便番号検索を行うには、以下のURLにGETリクエストを送信します。

`http://<HOST>:<PORT>/<CONTEXT_PATH>/api/zipcode/search`

検索URLへのクエリとして次のパラメータを付与します。

| パラメータ名 | 必須項目 | 形式 | 備考 |
|---|---|---|---|
| zipcode | △ | 数値 | 検索する郵便番号を3桁から7桁で指定します。<br>このパラメータとwordのいずれかを指定する必要があります。 |
| word | △ | 文字列 | 検索する住所の一部を指定します。<br>このパラメータとzipcodeのいずれかを指定する必要があります。 |
| start | × | 数値 | 取得開始位置を指定します。<br>指定されていない場合は検索結果の1件目から返却します。 |
| length | × | 数値 | 取得件数を指定します。<br>指定されていない場合は検索結果を全件返却します。 |
| callback | × | 文字列 | コールバック用の関数の名前を指定します。<br>このパラメータはクロスドメインの場合に利用します。 |

リクエストを送信すると以下のようにJSONでエンコードされたレスポンスが返却されます。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
   "error": false,
   "total": 53,
   "data": [{
      "city": "港区",
      "cityKana": "ミナトク",
      "jisCode": "13103",
      "officeName": "",
      "officeNameKana": "",
      "oldZipCode": "107  ",
      "prefecture": "東京都",
      "prefectureKana": "トウキョウト",
      "street": "",
      "town": "赤坂",
      "townKana": "アカサカ",
      "zipCode": "1070052"
   }]
}
```

パラメータに誤りがある場合等、検索時にエラーが発生した場合は以下のようなレスポンスが返却されます。

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
   "error": true,
   "errorMessage": "郵便番号または住所のいずれかを指定してください。",
   "total": 0,
   "data" : []
}
```

## 検索ダイアログを使用した検索画面の作成方法

intra-mart Accel Platform 上に画面を作成する場合、クライアントJavaScript APIを使用することで検索ダイアログを利用できます。
検索ダイアログの使用方法は以下の通りです。

1. 外部JSファイル「im_zipcode/csjs/im_zipcode.js」を読み込む。
   <imart type="head"> 内に、スクリプトタグを実装し、外部JSファイルを読み込みます。

   ```
   <script type="text/javascript" src="im_zipcode/csjs/im_zipcode.js"></script>
   ```

2. クライアントJavaScript API 「imuiZipcodeDialog」を呼び出す。
   任意のオペレーションでダイアログを呼び出すスクリプトを実装します。

   ```
   $('#button').click(function() {
      $('#dialog').imuiZipcodeDialog();
   });
   ```

3. ダイアログで選択された住所を受け取った際の動作を実装します。

   ```
   function callback(e, data) {
      $('#zip-code').val(data.zipCode);
      var address = data.prefecture + data.city + data.town;
      $('#address').val(address);
   };
   ```

imuiZipcodeDialog の詳しい利用方法は クライアントサイド JavaScript を参照してください。

### サンプルプログラム

検索ダイアログを利用して郵便番号を取得するサンプルは以下のとおりです。

```
<imart type="head">
  <title>郵便番号検索サンプル</title>
  <script type="text/javascript" src="im_zipcode/csjs/im_zipcode.js"></script>
  <script type="text/javascript">
  $(function() {
    $('#search-button').click(function() {
      // ダイアログを呼び出します。
      $('#zipcode-dialog').imuiZipcodeDialog();
    });
  });
  function callback(e, data) {
    // ここに検索結果を受け取った際の動作を実装します。
    $('#zip-code').val(data.zipCode);
    $('#address').val(data.prefecture + data.city + data.town);

    // ダイアログを自動で閉じる場合は、 close を実行します。
    $(this).imuiZipcodeDialog('close');
  }
  </script>
</imart>
<div class="imui-form-container">
  <input type="text" id="zip-code"/>
  <input type="text" id="address"/><br/>
  <input type="button" id="search-button" value="検索" class="imui-small-button">
</div>
<div id="zipcode-dialog"></div>
```

## 住所の郵便番号データインポートのクレンジング処理拡張方法

郵便番号データをインポートする際に、任意の加工を行いたい場合には、データクレンジング処理を拡張できます。
拡張方法は以下のとおりです。

1. サービス構成ファイルを作成し、拡張クラスが有効になるようにします。
   サービス構成ファイルは以下のファイルを作成し、拡張クラスのクラス名をフルパッケージで指定します。

   `META-INF/services/jp.co.intra_mart.foundation.zip_code.io.cleanser.ZipCodeDataCleanser`

2. 実測クラスを作成します。
   データクレンジングの実装クラスは以下のインタフェースの実装クラスとして作成します。

   `jp.co.intra_mart.foundation.zip_code.io.cleanser.ZipCodeDataCleanser`

3. 作成したクラスの `isSupport` メソッドにこの実装クラスが有効になる条件を実装します。

4. 作成したクラスの `cleansing` メソッドに任意のデータ加工処理を実装します。

実装例については、サンプルプログラムを参照してください。

### サンプルプログラム

クレンジング処理のサンプルプログラムは以下の通りです。

- サービス構成ファイル 「`META-INF/services/jp.co.intra_mart.foundation.zip_code.io.cleanser.ZipCodeDataCleanser`」

  ```
  sample.SampleZipCodeCleansing
  ```

- クレンジング処理実装 「`sample.SampleZipCodeCleansing.java`」

```
package sample;

import jp.co.intra_mart.foundation.zip_code.io.data.ZipCodeData;
import jp.co.intra_mart.foundation.zip_code.io.cleanser.ZipCodeDataCleanser;

/**
 * 事業所の個別郵便番号データのクレンジング処理サンプルです。
 * @author INTRAMART
 * @version 8.0.0
 */
public class SampleZipCodeCleansing implements ZipCodeDataCleanser {

   /**
    * クレンジングパターンに「sample」が指定された場合に、このクレンジング処理が有効になります。
    * @see jp.co.intra_mart.system.zip_code.io.cleanser.ZipCodeDataCleanser#isSupport(java.lang.String)
    */
   @Override
   public boolean isSupport(String cleansingType) {
      return cleansingType.equals("sample");
   }

   @Override
   public ZipCodeData[] cleansing(ZipCodeData info) {
      // TODO ここにデータの加工処理を実装します。
      // 複数のデータに分割して登録を行う場合はZipCodeDataオブジェクトを複数件返却します。
      // 受け渡されたデータを登録しない場合は null を返却します。
      return new ZipCodeData[] { info };
   }

}
```

## 事業所の個別郵便番号データインポートのクレンジング処理拡張方法

郵便番号データをインポートする際に、任意の加工を行いたい場合には、データクレンジング処理を拡張できます。
拡張方法は以下のとおりです。

1. サービス構成ファイルを作成し、拡張クラスが有効になるようにします。
   サービス構成ファイルは以下のファイルを作成し、拡張クラスのクラス名をフルパッケージで指定します。

   `META-INF/services/jp.co.intra_mart.foundation.zip_code.io.cleanser.OfficeZipCodeDataCleanser`

2. 実測クラスを作成します。
   データクレンジングの実装クラスは以下のインタフェースの実装クラスとして作成します。

   `jp.co.intra_mart.foundation.zip_code.io.cleanser.OfficeZipCodeDataCleanser`

3. 作成したクラスの `isSupport` メソッドにこの実装クラスが有効になる条件を実装します。
4. 作成したクラスの `cleansing` メソッドに任意のデータ加工処理を実装します。

実装例については、サンプルプログラムを参照してください。

### サンプルプログラム

クレンジング処理のサンプルプログラムは以下の通りです。

- サービス構成ファイル 「`META-INF/services/jp.co.intra_mart.foundation.zip_code.io.cleanser.OfficeZipCodeDataCleanser`」

```
sample.SampleOfficeZipCodeCleansing
```

- クレンジング処理実装 「`sample.SampleOfficeZipCodeCleansing.java`」

```
package sample;

import jp.co.intra_mart.foundation.zip_code.io.data.OfficeZipCodeData;
import jp.co.intra_mart.foundation.zip_code.io.cleanser.OfficeZipCodeDataCleanser;

/**
 * 事業所の個別郵便番号データのクレンジング処理サンプルです。
 * @author INTRAMART
 * @version 8.0.0
 */
public class SampleOfficeZipCodeCleansing implements OfficeZipCodeDataCleanser {

    /**
     * クレンジングパターンに「sample」が指定された場合に、このクレンジング処理が有効になります。
     * @see jp.co.intra_mart.system.zip_code.io.cleanser.OfficeZipCodeDataCleanser#isSupport(java.lang.String)
     */
    @Override
    public boolean isSupport(String cleansingType) {
        return cleansingType.equals("sample");
    }

    @Override
    public OfficeZipCodeData[] cleansing(OfficeZipCodeData info) {
        // TODO ここにデータの加工処理を実装します。
        // 複数のデータに分割して登録を行う場合はOfficeZipCodeDataオブジェクトを複数件返却します。
        // 受け渡されたデータを登録しない場合は null を返却します。
        return new OfficeZipCodeData[] { info };
    }

}
```