オープンソースジオコーダ
geocoder.ja
マニュアル

V1.0
2006.7.1

株式会社オークニー

目次

１．システムの概要

このシステムは住所文字列を解析して、表記方法を統一したり、住所の緯度経度を取得したりするものです。単独のアプリケーションではなく、ライブラリとして提供されるので、様々な用途に適用できます。またオープンソースであるので、カスタマイズや高速化がユーザ側で可能です。

１．１　住所文字列の解析

一般に使われている住所は色々な表記方法があります。

例えば、「東京都千代田区丸の内一丁目２番３号」という住所は「東京都千代田区丸の内１－２－３」と簡略にしたり、「丸の内」が「丸ノ内」と表記されたりします。また全国に一つしかない市町村では都道府県名や郡名を省略したりします。

このような表記方法の違いは住所をコンピュータ処理する場合に問題となります。住所で検索する場合に検索できなかったり、住所の入力エラーと判断されたりします。そのため、何らかの方法で住所を統一する方法が必要となります。

その統一のため、このシステムでは、色々な表記方法の住所文字列を解析して、その住所が示す場所を一つに特定します。その特定した場所にコード番号を振り、システムの内部では住所をコード番号で管理します。

このシステムでは、そのコード番号を住所コードと呼びます。

１．２　住所文字列の正規化

特定された場所の住所コードは番号（実際には複数の番号の集まり）であり、最終的には、その番号を通常使う住所文字列に変換する必要があります。

この変換を行う際に、統一した形式で変換する事により、元は色々な表記の住所文字列を統一した住所文字列に変換できます。このようにして、住所を統一した形式に変換する事を住所の正規化と呼びます。

ここで、統一した形式とは、何らかの既定の標準形式がある訳ではありませんので、用途に応じた統一形式という事になります。

１．３　緯度経度の取得

住所文字列を住所コードに変換すると、その住所コードと結びつけられた色んなデータベースのデータを検索できます。

このシステムでは緯度経度データのデータベースを持っていますので、住所の代表点の緯度経度を取得する事ができます。

１．４　ライブラリ

上記の住所文字列の解析、正規化、緯度経度の取得などの機能はこのシステムの住所解析用ライブラリ（ジオコーダライブラリ）のAPI(Application Programing Interface)を通して利用できます。

ライブラリはLinuxとWindowsの２種類のOSで利用可能で、gccやVC++などの言語から直接呼び出す事ができます。

１．５　住所データベース

ジオコーダライブラリは、住所の市区町村名、町大字名などの住所の要素文字列を格納した住所データベースを参照します。解析する住所文字列を、住所データベースに登録されている要素文字列と比較する事で、元の住所文字列を県名、市区町村名、町大字名といった広い区域から狭い区域に順次分解していきます。このエリアを絞っていくプロセスが解析作業そのものであり、住所データベースの出来が解析そのものを決定付けます。

当システムでは、住所データベースそのものは提供せず、ユーザが用途に応じて元データ（住所基本データ）から変換するようになっています。

１．６ 住所基本データ

住所データベースを構築するためのベースとなる住所基本データとして、当システムでは次の２種類が利用できます。

１）郵便番号データ＋街区レベル位置参照情報

日本郵政公社から提供されている郵便番号データファイルと、国土交通省から提供されている街区レベル位置参照情報を元データとして、住所データベースを作成する事ができます。

この基本データは無料で利用できますが、丁目字レベル以下の情報が都市部だけになってしまいます。また、取得できる緯度経度に関しても、街区レベルまでですので、配送などのピンポイントの位置情報が必要な用途には向きません。ただ、顧客の位置分布を知るなどの用途には十分利用できます。

２）市販の住所データ

有料になりますが、インクリメントP株式会社から提供されている住所データは住居番号レベルまでの詳しさでピンポイントの位置特特定ができ、都市部だけでなく地方も含めた全国の区域で利用できます。但し用途による制限もあり、ライセンス契約が必要となります。

１．７ データベース作成プログラム

２種類の住所基本データは、それぞれの変換プログラムにより、検索に適した形式に変換され、検索用住所データベースが作成されます。プログラムはWindowsXPなどマイクロソフトのOS上で稼動致します。

１．８ オープンソース

当ライブラリはオープンソースですので、ユーザが目的に応じてカスタマイズしたり、処理の遅い部分を高速化したりする事ができます。

ソースはライブラリ部分はC言語で、データ作成ユーティリティはVC++で書かれています。

２．稼動環境

当システムは次の環境で稼動します。

２．１ ライブラリ

ジオコーダライブラリはLinuxとWindowsのOS上で利用できます。一般的なライブラリ以外に必要なライブラリと稼動を確認したOSのバージョンは次のとおりです。

１）Linux

必要なライブラリ　：　libiconv

動作確認済みバージョン：FedoraCore３

２）Windows

動作確認済みバージョン：Windows2000、WindowsXP

３）ハードウェア

両OSで必要とされるマシンのスペックは次のとおりです。

CPU:　２GHz　　メモリ：1GB

HD：　住所DBの範囲で変わります。全国の場合は５００MB　以上

２．２ 住所データベース作成プログラム

住所基本データから住所データベースを作成するプログラムはWindowsのOS上で利用できます。稼動を確認したOSのバージョンは次のとおりですが、作成プログラムは特別な処理はしていませんので、他のバージョンでも動くものと考えられます。

１）Windows

動作確認済みバージョン：Windows2000、WindowsXP

２）ハードウェア

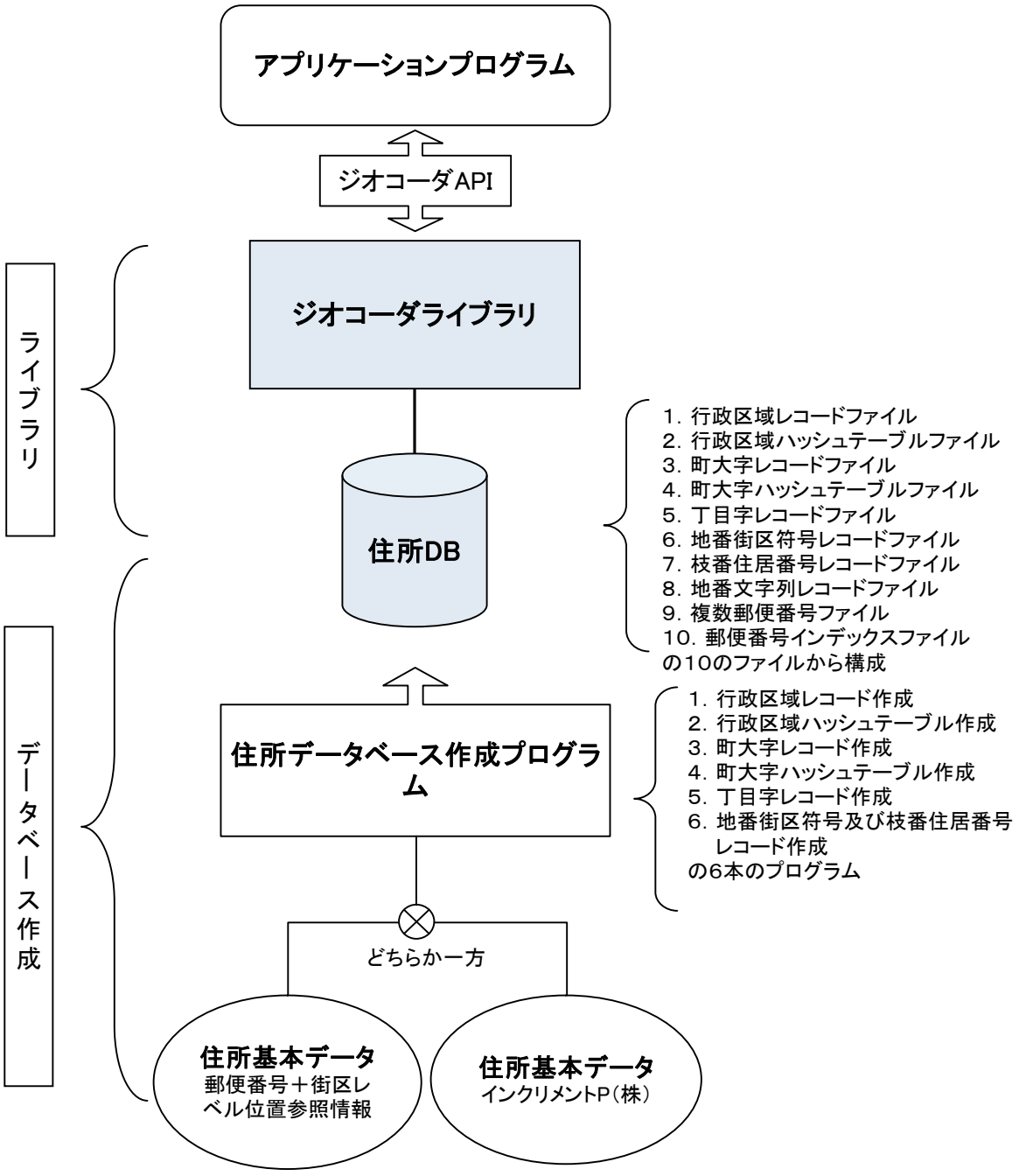
必要とされるマシンのスペックは次のとおりです。

CPU:　１GHz　　メモリ：５００MB

HD：　住所DBの作成範囲で変わります。全国の場合は２GB　以上

3. システム構成図

当システムの全体の構成を図で示すと以下のようになります。大きく分けてライブラリ部分と住所データベース作成部分の２つに分かれます。ライブラリ部分はライブラリのプログラム本体と住所データベースから構成されます。
住所データベース作成部分は、データベースの元となる住所基本データと作成プログラムで構成されます。作成プログラムは６つの段階に分かれていて、前の段階で作成されたデータを元にして次の段階での作成が行われます。住所データベースの作成は最初に行なった後は変更があるまで行う必要はありません。
住所データベースは９つのファイルから構成されています。これらのファイルはライブラリ起動時にメモリに読み込まれ、高速アクセスが可能となっています。



４．インストール

当システムはプログラムおよび必要なデータファイルが一つの圧縮ファイルにして配布されます。
インストールはその圧縮ファイルを解凍し、ライブラリを使用環境に応じたディレクトリにコピーすれば終わりです。

４．１ 配布形式

１）住所データベース作成プログラム

住所データベース作成プログラムはWindowsで稼動します。プログラムは次のファイル名の圧縮ファイルとなっています。

ファイル名：geocoderja_1_0.lzh

２）ライブラリ

ライブラリはLinuxおよびWindows上で稼動します。ライブラリはOSによって次のファイル名の圧縮ファイルとなっています。

ファイル名：

Linux　：　geocoderja_1_0.tar.gz
Windows：　geocoderja_1_0.lzh

４．２ ファイル一覧

圧縮ファイルを解凍すると出来るファイルは次のとおりです。

１．住所データベース作成プログラム

１）mkgyoseirec.exe　：行政区域レコード作成プログラム
２）mkgyoseihash.exe　：行政区域名ハッシュテーブル作成プログラム
３）mkchorec.exe　：町大字レコード作成プログラム
４）mkchohash.exe　：町大字名ハッシュテーブル作成プログラム
５）mkazarec.exe　：丁目字レコード作成プログラム
６）mkgaikurec.exe　：地番街区符号レコード作成プログラム
７）gcj_gyoseimei.csv：行政区域名リストファイル
８）gcj_variation.csv：表記バリエーションファイル
９）mkgeodb.exe　：住所データベース作成プログラム

２．ライブラリ

Linux　：　geocoderja.so, geocoderja.a
Windows：　geocoderja.dll

３．ソースコード

上記各バイナリファイルのソースコード

５．住所データベースの作成

ジオコーダライブラリを利用するためには、まず住所データベースを作る必要があります。
その作成手順は住所基本データの種類によって違います。

５．１　郵便番号＋街区レベル位置参照情報を使用する場合

郵便番号データと街区レベル位置参照情報データを各ホームページよりダウンロードし、圧縮ファイルを解凍します。そして、住所データベース作成プログラムでデータベースを作成します。

５．１．１　郵便番号データのダウンロード

日本郵政公社のホームページから郵便番号データをダウンロードします。
ホームページ：
タイトル：住所の郵便番号のダウンロードサービス
「読み仮名データの促音・拗音を小書きで表記したもの:」
URL　：　http://www.post.japanpost.jp/zipcode/dl/kogaki.html
郵便番号データには、読み仮名の種類によって２種類ありますが、「促音・拗音を小書きで表記したもの:」の方をダウンロードします。また、地域は全国一括のファイルをダウンロードします。
ダウンロード後、ファイルを解凍すると、KEN_ALL.CSVというファイルが出来ますので、適当なフォルダに格納します。

５．１．２　街区レベル位置参照情報のダウンロード

国土交通省のホームページから街区レベル位置参照情報データをダウンロードします。
ホームページ：
タイトル：街区レベル位置参照情報ダウンロードサービス
URL: http://nlftp.mlit.go.jp/isj/index.html
このホームページの”ダウンロード　データのダウンロードはこちらから。”のリンクをクリックすると、データのダウンロードのページが表示されます。
位置参照情報は市区町村単位か県単位かの選択ができますが、当システムでは県単位のデータのみに対応しています。必要な範囲をカバーする県を選択してダウンロードして下さい。また、データの作成年度の選択もできますが、最新のものを選択します。古い年度のものは郵便番号データと合わない市区町村が多くなります。もし古い年度のものが必要な場合は郵便番号データもできるだけ同じ時期のものをダウンロードして下さい。
ダウンロード後、ファイルを解凍すると、NN_YYYY.CSV（NNは都道府県コード、YYYYは作成年度）というファイルが出来ますので、適当なフォルダに格納します。解凍した際、他にもファイルが出来ますが、当システムでは使用しません。

５．１．３　住所データベースの作成

郵便番号データと街区レベル位置参照情報データのダウンロードが完了すると、次はプログラムにより、住所データベースを作成します。
作成は次の６段階に分かれます。
１）行政区域レコードの作成
２）行政区域名ハッシュテーブルの作成

３）町大字レコードの作成
４）町大字名ハッシュテーブルの作成
５）丁目字レコードの作成
６）地番街区符号レコードの作成

これらの各段階での作成後、住所データベースとして次のファイルが得られます。

１）gcj_gyosei.rec　　：行政区域レコードファイル
２）gcj_gyosei.hsh　　：行政区域ハッシュテーブルファイル
３）gcj_cho.rec　　：町大字レコードファイル
４）gcj_cho.hsh　　：町大字ハッシュテーブルファイル
５）gcj_aza.rec　　：丁目字レコードファイル
６）gcj_gaiku.rec　　：地番街区符号レコードファイル
７）gcj_jukyo.rec　　：枝番住居番号レコードファイル
８）gcj_string.rec　　：地番文字列レコードファイル
９）gcj_multizip.rec　　：複数郵便番号ファイル
１０）gcj_post.idx　　：郵便番号インデックスファイル

作成の各段階で起動するプログラムと、その際に必要なファイル、結果として作成されるファイルは、次のとおりです。この各段階にわたって、順次作成していきます。作成の際に指定するdbDirName（住所DBファイルが保存されているディレクトリ名）にはそのプログラムの実行に必要な入力ファイルが存在しなければなりません。また、出力ファイルはdbDirNameのディレクトリに作成されます。

作成段階		
説明		
プログラム名	入力ファイル	出力ファイル
プログラムの起動方法		
1. 行政区域レコードの作成		
県名、支庁名、郡政令市名、市区町村名の各名称と、読み仮名、コード番号などのレコードを作成します。		
mkgyoseirec.exe	gcj_gyoseimei.csv	gcj_gyosei.rec
起動方法： mkgyoseirec　dbDirName dbDirName ： 住所DBファイルが保存されているディレクトリ名 ディレクトリには必要な入力ファイルが存在していなければなりません（以下同じ）。		
2. 行政区域ハッシュテーブルの作成		
県名、支庁名、郡政令市名、市区町村名の頭２文字のハッシュ値を計算しファイル化します。		
mkgyoseihash.exe	gcj_gyoseimei.csv gcj_variation.csv	gcj_gyosei.hsh
起動方法： mkgyoseihash　dbDirName dbDirName ： 住所DBファイルが保存されているディレクトリ名		

3. 町大字レコードの作成

郵便番号データを元にして、町大字の各名称と、読み仮名、コード番号などのレコードを作成します。

mkchorecord.exe	gcj_gyosei.rec gcj_gyosei.hsh 郵便番号データken_all.csv	gcj_cho.rec gcj_multizip.rec gcj_post.idx

起動方法： mkchorecord dbDirName sourceDirName
dbDirName ： 住所DBファイルが保存されているディレクトリ名
sourceDirName: 郵便番号データが格納されているディレクトリ名

4. 町大字名ハッシュテーブルの作成

町大字名の頭２文字のハッシュ値を計算しファイル化します。

mkchohash.exe	gcj_cho.rec	gcj_cho.hsh

起動方法： mkchohash dbDirPath
dbDirName ： 住所DBファイルが保存されているディレクトリ名

5. 丁目字レコードの作成

街区レベル位置参照情報データを元にして、丁目字の各名称と、読み仮名、コード番号、緯度経度などのレコードを作成します。

mkazarec.exe	gcj_gyosei.rec gcj_gyosei.hsh gcj_cho.rec gcj_cho.hsh 街区レベルデータ（CSV）	gcj_aza.rec

起動方法： mkazarec gcjDbDir gaikuDirName
dbDirName ： 住所DBファイルが保存されているディレクトリ名
gaikuDirName: 街区レベルCSVファイルが格納されているディレクトリ

6. 地番街区符号レコードの作成

街区レベル位置参照情報データを元にして、地番または街区符号の緯度経度、コード番号などのレコードを作成します。一部、枝番がある場合は枝番レコードも作成します。

mkgaiku.exe	gcj_gyosei.rec gcj_gyosei.hsh gcj_cho.rec gcj_cho.hsh gcj_aza.rec 街区レベルデータ（CSV）	gcj_gaiku.rec gcj_jukyo.rec gcj_string.rec

起動方法： mkgaiku gcjDbDir gaikuDirName
dbDirName ： 住所DBファイルが保存されているディレクトリ名
gaikuDirName: 街区レベルCSVファイルが格納されているディレクトリ

上記の各段階のプログラムの起動をより簡単に行うためのツールも用意されています。詳しくは「ジオコーダ用住所データベース作成プログラム.pdf」を参照して下さい。

５．２ 市販の住所基本データを使用する場合

インクリメントP(株) より販売されています住所基本データを使用する場合は、次の手順で住所データベースを作成します。

データに同梱されているmkincp.exeという実行ファイルを起動します。起動後、次の各項目のパラメータを入力します。

１．製品のCDROMをセットしたドライブ
２．住所データベースを作成する区域
３．住所データベースを格納するディレクトリ

その後、”作成”ボタンを押すと、住所データベースが作成されます。

６．ライブラリの利用

住所データベースが作成された後はライブラリが利用できるようになります。ライブラリの使用方法は次のとおりです。なお、詳細は各関数のリファレンスを参照して下さい。

６．１　住所データベースのメモリへの格納

ライブラリを使用する最初に、住所データベースをメモリにロードします。

(例)

```
path = "/data/address_data";        // 住所データベースの格納ディレクトリ
loadLevel = 4;                      // 地番街区符号のデータまでをロード
retCode =  gcjDbLoad( path, loadLevel );
```

６．２　文字コードの設定

次に、関数で受け渡しする住所文字列の文字コードを設定します。シフトJISの場合は設定しなくても構いません。

(例)

```
retCode = gcjSetArgEncoding( "EUC-JP" );
```

６．３　住所を緯度経度に変換

住所を緯度経度に変換する場合は、次の２つのステップで行います。

１）住所を住所コードに変換する。

２）住所コードを緯度経度に変換する。

(例)

```
char *address;
ADR_CODE adrCode;
double latitude,longitude;
address = "東京都千代田区丸の内一丁目２番３号"

//  住所を住所コードに変換
retCode = gcjAdrStr2Code( address, &adrCode );

// 住所コードを緯度経度に変換
retCode = gcjAdrCode2Point( &adrCode, &latitude, &longitude);
```

６．４　住所を正規化

住所を統一した書式に変換（正規化）する場合は、次の２つのステップで行います。

１）住所を住所コードに変換する。

２）住所コードを住所文字列に変換する。

(例)

```
char *address,kanji[256];
ADR_CODE adrCode;
double latitude,longitude;
address = "東京都千代田区丸ノ内1-2-3"
```

```
//  住所を住所コードに変換
retCode = gcjAdrStr2Code( address, &adrCode );

// 住所コードを住所文字列に変換
retCode = gcjAdrCode2Str( &adrCode, kanji, 256, NULL, 0 );

// kanji[]には"東京都千代田区丸の内一丁目２番３号"が返ります。
```

出力形式を変更したい場合はgcjSetAdrStringFormat()で形式を設定します。

（例）丁目の数字を算用数字で出力する場合

```
retCode = gcjSetAdrStringFormat( 21, 0 );
retCode = gcjAdrCode2Str( &adrCode, kanji, 256, NULL, 0 );
// kanji[]には"東京都千代田区丸の内１丁目２番３号"が返ります
```

６．５ ライブラリの利用終了処理

ライブラリの使用が終了した場合は、住所データベースなどライブラリ用に確保したメモリを解放します。この処理は必ず行って下さい。

（例）

```
retCode = jcjDbEnd();
```

６．６ 住所コード

ライブラリの関数で、住所を緯度経度へ変換したり、正規化するだけであれば、ライブラリで使われている住所コードに関して知る必要はありません。しかし、住所データベースの内容に直接アクセスして、より柔軟な対応を行うといった場合は、住所コードについての知識が必要です。以下に住所コードについて解説いたします。

住所コードは以下の構造体です。

```
typedef struct {
    int codeLevel; /* コードが有効なレベル  */
    int cityCode;  /* 市区町村コード(JIS) 5桁      : レベル１～４  */
    int choCode;   /* 町大字コード（独自）         : レベル５  */
    int azaCode;   /* 丁目字コード（独自）         : レベル６  */
    int gaikuCode; /* 地番街区符号コード（独自）   : レベル７  */
    int jukyoCode; /* 枝番住居番号コード（独自）   : レベル８  */
} ADR_CODE ;
```

ここで、「コードが有効なレベル」とは、この構造体に格納されているコードのうち、上位レベルからどのコードまで有効な値が入っているかを示す値です。例えば、codeLevelの値が６の場合は丁目字コードまで有効な値として利用できる事を示します。

また、コード番号が下位のレベルまで有効な値が入っていても、より上位のレベルのコード番号を関数に渡したい場合はcodeLevelを渡したいレベルの値にして使用するという使い方もできます。

市区町村コードはレベル１～４となっていますが、これは
都道府県レベル　：　レベル１
支庁レベル　　：　レベル２
郡政令市レベル　：　レベル３
市区町村レベル　：　レベル４
の４つのレベルをまとめています。これは５桁の市区町村コードの番号を解析する事で上位のコード番号を求める事ができるからです。

６．７ 住所データベースへの直接アクセス

住所データベースのレコードを直接読み出す場合は次のようにします。

１．読み出したいデータの住所コードが分かっている場合
　gcjGetAdrRecord()で直接読み出す事ができます。

２．ある市の町大字名を全て読み出したいといった場合
　１）読み出したいレベルの住所データのレコード数を取得します。
　２）そのレコード数のintの配列を確保します。
　３）その配列にレコードのコード番号のリストを読み込みます。
　４）配列のコード番号を指定してレコードを取得します。

（例）東京都新宿区の町大字レコードの最初のレコードを取得する場合

```
ADR_CODE adrCode;
int *codeNums;
CHO_RECORD choRecord;
adrCode.codeLevel = 5;          // 町大字区域を表すコードは5
adrCode.cityCode = 13104;     // 新宿区の市区町村コードは13104

// numReordsに町の数が返されます。
numRecords =  gcjGetNumAdrRecords( &adrCode );

// 町の数分のintの配列を確保します。
codeNums  = (int*) malloc( numRecords * sizeof(int) );

// codeNumsの配列にコード番号が格納されます。
numCodes = gcjGetAdrRecordList( &adrCode, codeNums, numRecords );

// 最初のレコードがrecordに格納されます。
adrCode.choCode = codeNums[0];
result = gcjGetAdrRecord( &adrCode, (void*) &choRecord, sizeof(record)
);
```

7. 住所データベースのファイルフォーマット

住所データベースは複数のファイルから構成されています。
住所は都道府県から始まって、地番、号のレベルまで階層構造になっていますが、その階層に応じた住所データ別にファイル化されています。
このライブラリでは、高速化のために、メモリに住所データベースをロードします。ファイルはそのロードしたメモリイメージの形式になっています。
ファイルの種類は次の10種類です。

１．行政区域レコードファイル
２．行政区域ハッシュテーブルファイル
３．町大字レコードファイル
４．町大字ハッシュテーブルファイル
５．丁目字レコードファイル
６．地番街区符号レコードファイル
７．枝番住居番号レコードファイル
８．地番文字列レコードファイル
９．複数郵便番号ファイル
１０．郵便番号インデックスファイル

各ファイルの詳細は次のとおりです。

ファイルの記録形式

文字コード：シフトJIS
バイト順序：リトルエンディアン
レコード番号：特に断りが無い場合、レコード番号やテーブル番号などのシリアル番号は１から始まります。また、レコード配列のインデックスとシリアル番号は一致します。即ち配列の最初のレコードは通常のレコードとしては使用していません。
かな：半角カタカナ

名称 **行政区域レコードファイル**

ファイル名 gcj_gyosei.rec

説明 都道府県、支庁、郡政令市、市区町村の各区域に関するレコードを格納したファイルです。

内容

ファイル識別子	８バイト文字列　″gyorecd¥0″
バージョン	４バイト整数
予備	８バイト文字列
レコード総数	４バイト整数
レコード番号テーブル	
{	
レコード番号[48000]	４バイト整数 x 48000 キーは行政区域コード
}	
行政区域レコード[]	
{	
種別	４バイト整数（１：都道府県　２：支庁 3:郡　4:市区町村）
県コード（JIS）	４バイト整数（1..47）
支庁コード（JIS）	４バイト整数（５桁コード）
郡政令市コード（JIS）	４バイト整数（５桁コード）
市区町村コード（JIS）	４バイト整数（５桁コード）
廃止フラグ	４バイト整数（0:存在　1:廃止）
新市区町村コード（JIS）１	４バイト整数
新市区町村コード（JIS）２	４バイト整数
新市区町村コード（JIS）３	４バイト整数
行政名称	２４バイト文字列
読み仮名	１６バイト文字列（半角カナ）
緯度	４バイト整数（ミリ秒）
経度	４バイト整数（ミリ秒）
}	

・使用しない、または意味のないコードは0が入っています。

・支庁、郡コードでは一部JISコードでなく独自コードの場合があります。

・合併などで廃止、変更のあった行政区域の場合は、廃止フラグを立てて、変更後の市区町村コードを新市区町村コードへ記入。
複数の市区町村になった場合は新市区町村コード２以降に記入。

名称　行政区域ハッシュテーブルファイル

ファイル名　gcj_gyosei.hsh

説明

都道府県名、支庁名、郡政令市名、市区町村名の各名称の頭２文字についてのハッシュ値を記録したファイルです。

内容

ファイル識別子	８バイト文字列　″gyohash¥0″
バージョン	４バイト整数
予備	４バイト文字列
名称レコード数	４バイト整数
ハッシュサイズ:n	４バイト整数
レコード番号テーブル	
{	
名称レコードの番号[n]	４バイト整数×n
}	
名称レコード[]	
{	
検索用行政名称	２４バイト文字列
行政区域コード	４バイト整数
種別	４バイト整数（１：都道府県　２：支庁 3:郡　4:市区町村）
次の名称レコード番号	４バイト整数；（同じハッシュ値のものが無い時は0）
}	

・名称レコードの番号[]の配列のインデックスがハッシュ値となっています。例えば、ハッシュ値が125とすると、125x4=500バイト目にハッシュ値125に対応する名称レコードの番号が格納されています。
・頭２文字のハッシュ値が一致する名称について名称レコードのリストを探索して一致する名称を取り出します。

名称　町大字レコードファイル

ファイル名　gcj_cho.rec

説明

町大字に関するレコードを格納したファイルです。

内容

```
ファイル識別子                    ８バイト文字列　″chorecd¥0″
バージョン                        ４バイト整数
予備                              ８バイト文字列
総レコード数                      ４バイト整数
１レコードのバイト数              ４バイト整数
レコード先頭のバイト位置          ４バイト整数
テーブルリスト
 {
    テーブル位置レコード[48000]　（キーは市区町村コード）
    {
        レコード数                          ４バイト整数
        テーブル先頭の町大字レコード番号    ４バイト整数
    }
 }
町大字テーブル[]
 {
     町大字レコード[]
     {
        町・大字コード          ４バイト整数　町大字コードは１から
                                    始まる（注２）
        システム用              ４バイト整数
        町・大字名              ３１バイト文字列
        読み仮名                ２９バイト文字列　（半角カナ）
        郵便番号                ４バイト整数（注１）
        丁目字テーブルの番号    ４バイト整数　無い時は0
        地番、街区符号テーブルの番号　４バイト整数
                                        （丁目字が無い時に指定）
        緯度                    ４バイト整数（ミリ秒）
        経度                    ４バイト整数（ミリ秒）
    }
 }
```

注１：一つの町大字に複数ある場合、この番号にはマイナスの値が入っています。実際の郵便番号は複数郵便番号ファイルに記載されていて、その参照用番号をマイナスの値として記入してあります。

注２：実際の町大字コードは2から始まります。1は字が直接始まる場合の

仮想の大字（名無し）とします。

・町大字レコードは市区町村毎にまとまって格納されていて、その市区町村毎のレコードの集まりをテーブルと読んでいます。フォーマット上はテーブルの境界は特に無く、全市区町村のレコードが連続して並んでいます。

名称　町大字名ハッシュテーブルファイル

ファイル名　gcj_cho.hsh

説明

町大字名の名称の頭２文字についてのハッシュ値を記録したファイルです。

内容

ファイル識別子	８バイト文字列　″chohash¥0″
バージョン	４バイト整数
予備	１６バイト文字列
ハッシュテーブルの総バイト数	４バイト整数
名称レコード総数	４バイト整数
ハッシュテーブル位置[48000]	４バイト整数 x 48000 （キーは市区町村コード）

市区町村別町大字ハッシュテーブル[]

{

市区町村コード	４バイト整数
ハッシュサイズ:n	４バイト整数
名称レコード位置[n]	４バイト整数 x n

}

名称レコード[]

{

検索用町大字名称	３２バイト文字列
町大字コード	４バイト整数
次の名称レコード番号	４バイト整数　（無い場合は０）

}

・名称レコード位置配列のインデックスがハッシュ値となっています。
・頭２文字のハッシュ値が一致する名称について名称レコードのリストを探索して一致する名称を取り出します。

名称　丁目字レコードファイル

ファイル名　gcj_aza.rec

説明

丁目字に関するレコードを格納したファイルです。

内容

ファイル識別子	８バイト文字列　″azarecd¥0″	
バージョン	４バイト整数	
予備	８バイト文字列	
テーブル最大数：n	４バイト整数	
現在のテーブル数	４バイト整数	
レコード総数	４バイト整数	
テーブルリスト[n]		
{		
５桁市区町村コード	４バイト整数	
町大字コード	４バイト整数	
丁目字レコード数	４バイト整数	
連続丁目の数	４バイト整数	（注３）
テーブル先頭のレコード番号	４バイト整数	（注２）
}		
丁目字テーブル[]		
{		
丁目字レコード[]		
{		
町大字コード＋丁目字コード	４バイト整数	（注１）
システム用	４バイト整数	
丁目字名称	３１バイト文字列	
丁目字かな	２９バイト文字列	
地番、街区符号テーブルの番号	４バイト	（無い時は0）
緯度	４バイト整数（ミリ秒）	
経度	４バイト整数（ミリ秒）	
}		
}		

注１：上位２バイトを町大字コード、下位２バイトを丁目字コードとします。

注２：丁目字レコード番号はファイル内一意で、１から順に付けます。

注３：丁目が１から連続している最大の丁目数。現バージョンでは使用していません。

・ｎは丁目字を持つ町名の数ですが、現バージョンでは固定にしてあり、150,000となっています。

名称　**地番街区符号レコードファイル**

ファイル名　gcj_gaiku.rec

説明　地番および街区符号に関するレコードを格納したファイルです。

内容

ファイル識別子	８バイト文字列	″gaikurc¥0″
バージョン	４バイト整数	
予備	８バイト文字列	
テーブル最大数	４バイト整数	（未使用）
現在のテーブル数：n	４バイト整数	
レコード総数	４バイト整数	
テーブルリスト[n]		
{		
５桁市区町村コード	４バイト整数	
町大字コード	４バイト整数	
丁目字コード	４バイト整数	
地番街区符号レコード数	４バイト整数	
テーブル先頭のレコード番号	４バイト整数	（注１）
住居表示フラグ	４バイト整数	（住居表示の場合１）
}		
地番街区符号テーブル[]		
{		
地番・街区符号レコード[]		
{		
地番、街区符号コード	４バイト整数	負数は文字列番号（注２）
枝番、住居番号テーブルの番号	４バイト	（無い時は0）
緯度	４バイト整数	（ミリ秒）
経度	４バイト整数	（ミリ秒）
}		
}		

注１：地番街区符号レコード番号はファイル内一意で、１から順に付けます。

注２：地番街区符号が数字のみの場合はその数字が地番街区符号コードとなります。文字列の場合は、その文字列は文字列ファイルに記録され、その文字列番号が負数として、このフィールドに記録されます。

名称　枝番住居番号レコードファイル

ファイル名　gcj_jukyo.rec

説明

枝番および住居番号に関するレコードを格納したファイルです。

内容

ファイル識別子	８バイト文字列　″jukyorc¥0″
バージョン	４バイト整数
予備	８バイト文字列
テーブル最大数	４バイト整数（未使用）
現在のテーブル数：n	４バイト整数
レコード総数	４バイト整数

テーブルリスト[n]
 {
　枝番住居番号レコード数およびチェックサム　４バイト整数（注１）
　テーブル先頭のレコード番号 ４バイト整数　（注２）
 }
枝番住居番号テーブル[]
 {
　枝番住居番号レコード[]
　{

枝番住居番号コード	４バイト整数 負数は文字列番号（注３）
緯度	４バイト整数 （ミリ秒）
経度	４バイト整数 （ミリ秒）

　}
 }

注１：レコード数は下位16ビットに入ります。当該テーブルの市区町村コード＋町大字コード＋丁目字コード＋地番街区符号コードの合計値が上位16ビットに格納されます。
注２：枝番住居番号レコード番号はファイル内一意で、１から順に付けます。
注３：枝番住居番号が数字のみの場合はその数字が枝番住居番号コードとなります。文字列の場合は、その文字列は文字列ファイルに記録され、その文字列番号が負数として、このフィールドに記録されます。

名称　**文字列ファイル**

ファイル名　gcj_string.rec

説明

地番、街区符号、枝番などが数字でなく文字列の場合、このファイルにその文字列が記録されています。

内容

ファイル識別子	８バイト文字列　″stringr¥0″
バージョン	４バイト整数
予備	１２イト文字列
１レコードのバイト数	４バイト整数
文字列レコード数	４バイト整数

文字列テーブル
{
　文字列[]　　１６バイト文字列
}

・文字列番号は１から始まるシリアル番号です。

名称　複数郵便番号ファイル

ファイル名　gcj_multipost.rec

説明　一つの町大字に複数の郵便番号がある場合、その郵便番号はこのファイルに格納されます。

内容

ファイル識別子	８バイト文字列　″multzip¥0″	
バージョン	４バイト整数	
予備	１２バイト文字列	
オフセットテーブルのレコード数	４バイト整数	
登録済みのオフセットレコード数	４バイト整数	
オフセットテーブル		
{		
オフセットレコード[]		
{		
町大字コード	４バイト整数	
郵便番号データへのポインタ	４バイト整数	ファイル先頭からのバイト位置
}		
}		
郵便番号データ[]	テキストデータ	

・暫定のファイル仕様であり、次のバージョンで変更が予定されています。
・郵便番号データのフォーマット
　郵便番号の次に区切り文字'>'１文字が入り、その後にその郵便番号に属する丁目や字名が来て、その後に一つの郵便番号のレコードの終わりを示す文字';'が来る。そして郵便番号の数だけそのパターンを繰り返す。
　　例）札幌市中央区大通西の場合
　　600042>１～１９丁目;640820>２０～２８丁目;

名称　**郵便番号インデックスファイル**

ファイル名　gcj_post.idx

説明　郵便番号から町大字コードを求めるためのインデックスファイルです。

内容

ファイル識別子	８バイト文字列　″postidx¥0″
バージョン	４バイト整数
予備	１６バイト文字列
レコード数	４バイト整数
郵便番号レコード[]	
{	
郵便番号	４バイト整数
市区町村コード	４バイト整数
町大字コード	４バイト整数
}	

・郵便番号レコードは郵便番号で昇順ソートされています。

8. 住所基本データのファイルフォーマット

住所データベースの元となる住所基本データは、階層毎に、いくつかのファイルから構成されています。

	郵便番号＋街区レベル位置参照情報	IPC社製住所基本データ
都道府県、支庁、郡政令市、市区町村名	行政区域名リストファイル（gcj_gyoseimei.csv）	
町大字名	郵便番号データ	町丁目データ
丁目、字名	街区レベル位置参照情報	町丁目データ
地番、街区符号	街区レベル位置参照情報	番地データ
枝番、住居番号	なし	号データ

これらの基本データ以外に、色々な住所表記に対応するために、表記バリエーションファイルがあります。
ここでは、行政区域名リストファイルと表記バリエーションファイルのフォーマットについて解説します。その他のファイルのフォーマットについては、各ソースのホームページないしは製品に添付のフォーマットをご参照下さい。

名称　**行政区域名リストファイル**

ファイル名　gcj_gyoseimei.csv

説明　都道府県名、支庁名、郡政令市名、市区町村名のリストです。

内容　以下の８つのフィールドのCSVファイルです。

１）通し番号
２）名称種別
　″都道府県名″、″支庁名″、″郡政令市名″、″市区町村名″のいずれか。
３）JISコード
４）郡コード（北海道のみ）
５）名称
６）読み仮名
７）緯度（ddmmss.s）
８）経度（dddmmss.s）
７）新JISコード１
８）新名称１
９）新JISコード２
１０）新名称２
１１）新JISコード３
１２）新名称３

・旧市区町村名も網羅する。旧の場合は新しく変わったJISコードと名称を記入。新しいJISコード名称が複数ある場合は新JISコード２，３、新名称２、３に記入。
　郡の名称のように無くなるものは新JISコードを-1とします。
・北海道および長崎県対馬支庁の郡名のコードは当該支庁のコード範囲内で使用さ　れていないものを割り当てます。

名称　**表記バリエーションファイル**

ファイル名　gcj_variation.csv

説明　住所の色々な表記方法に対応するための、文字列変換テーブルです。

内容　変換前文字列と変換後文字列をコンマで区切ったファイルです。
このファイルに登録されている変換前文字列が市区町村名や町大字名に現れた場合、変換前文字列を変換後の文字列で置き換えた名称も同一の市区町村名ないしは町大字名と見なします。
１項目１行で記録します。

(例)

ケ,ヶ
ケ,が
ケ,ガ
が,ケ
が,ヶ
が,ガ
の,ノ
ノ,の
峰,峯
岡,丘
丘,岡

関数リファレンス

ライブラリの関数で、通常のアプリケーションで使用する関数には、次のようなものがあります。

1	gcjDbLoad()	住所データベースをメモリにロードする
2	gcjDbEnd()	住所データベースに関する処理を終了する。
3	gcjSetArgEncoding()	受け渡しする住所文字列の文字コードを設定する。
4	gcjAdrStr2Code()	住所文字列を住所コードに変換する
5	gcjPost2AdrCode()	郵便番号を住所コードに変換する
6	gcjAdrCode2Point()	住所コードが示す場所の代表点の緯度経度を取得する
7	gcjAdrCode2Str()	住所コードを住所文字列に変換する
8	gcjSetAdrStringFormat	住所文字列の出力形式を設定する
9	gcjGetNumAdrRecords()	指定した住所レベルのレコード数を取得する
10	gcjGetAdrRecordList()	指定した住所レベルのレコードのリストを取得する。
11	gcjGetAdrRecord()	指定した住所コードのレコードを取得する。

これら以外にも、ライブラリには住所データベースにアクセスする下位レベルの関数があります。
それらの関数については、ソースコードをご参照下さい。

関数名　gcjDbLoad()

機能　住所データベースをメモリにロードする

宣言

```
int gcjDbLoad(
        char *dbDirPath,
        int loadLevel )
```

引数

(in)

dbDirPath : 住所データベースが格納されているディレクトリ

loadLevel : ロードする住所DBのレベル

DB_AUTO　= 0 : 最も詳細なレベル
DB_GYOSEI = 1 : 行政区域（県、郡、市区町村レベル）
DB_CHO　= 2 : 町大字
DB_AZA　= 3 : 丁目字
DB_GAIKU　= 4 : 地番、街区符号
DB_JUKYO　= 5 : 枝番、住居番号

戻り値

1以上 : ロードできた住所DBのレベル

0 : ロードできるデータが無かった

負数: エラー

-1 : ディレクトリのパスが長すぎる（255文字以内）
-2 : 指定したレベルは現在サポートされていない。
-10.. : 行政区域レコード　に関するエラー
-20.. : 行政区域名ハッシュテーブル　〃
-30.. : 町大字レコード　〃
-40.. : 町大字名ハッシュテーブル　〃
-50.. : 丁目字テーブル　〃
-60.. : 地番街区符号テーブル　〃
-70.. : 枝番住居番号テーブル　〃

・-10以降の場合、一桁目のエラーコード

-1 : メモリが確保できない
-2 : ファイルをオープンできない
-3 : ファイルIDの文字列が違う
-4以降 : ファイルのリードエラー

説明

住所データベースをメモリにロードします。
loadLevelがDB_CHO～DB_JUKYOの場合、指定したレベル以上（値としては小さい）の全てのデータをロードします。
DB_AUTOを指定すると利用できる最も詳細なレベルのデータをロードします。
郵便番号インデックスは町大字レベル以下のレベルで自動的にロードされます。
この関数でライブラリで必要な初期化も行いますので、他の関数を使用する前にこの関数を呼び出して下さい。

注意事項

ライブラリの使用が終了した時は、gcjDbEnd()を実行して、この関数で確保したメモリを解放して下さい。

関数名 gcjDbEnd()

機能 住所データベースに関する処理を終了する。

宣言 int gcjDbEnd()

引数 なし

戻り値 0 : 正常終了

説明 住所データベース用に確保したメモリを解放します。

注意事項

関数名　gcjSetArgEncoding()

機能　受け渡しする住所文字列の文字コードを設定する。

宣言

```
int gcjSetArgEncoding(
                char *encoding )
```

引数

(in)

encodeing : 符号化方式を示す文字列

"EUC-JP"

"SHIFT_JIS"（既定値）

"UTF-8"

のいずれか

戻り値

0 : 正常終了

-1: 指定された符号化方式はサポートしていない

説明

当ライブラリの関数を呼び出す際に、引数として受け渡しする文字列の符号化方式を設定します。

注意事項

住所データベースの文字コードはシフトJISを採用しています。そのため、Unicodeで管理されている住所の場合、文字が住所データベースの文字とマッチングしない事があります。文字はJIS漢字の第１，２水準に限定するようにして下さい。

関数名　gcjAdrStr2Code()

機能　住所文字列を住所コードに変換する

宣言

```
int gcjAdrStr2Code(
                char *address,
                ADR_CODE *adrCode )
```

引数

(in)
　address : 住所文字列

(out)
　adrCode : 住所コード

戻り値

1以上 ： 住所文字列のうち、コードに変換できた文字列のバイト数
0　： 指定された住所文字列は住所DBには登録されていない
負数 ： エラー
-1　： 住所の頭部分に該当する行政区域名が複数ある。
　　より上位の行政区域名（郡名や県名）が必要。
-2　： 引数が異常（住所文字列が空か、adrCodeがNULL）
-3以下 ： 検索上のエラー

説明

住所文字列を住所コード（ADR_CODE構造体）に変換します。変換は住所文字列の先頭から順に住所データベースの県名、支庁名、郡名、市区町村名、町大字名、丁目字名と照らし合わせることで行います。
住所データベースとのマッチングがとれなくなった時点で関数から戻ります。どの段階までマッチングできたかは住所コード（構造体）のcodeLevelのフィールドで知る事ができます。

関数名　gcjPost2AdrCode()

機能　郵便番号を住所コードに変換する

宣言

```
int gcjPost2AdrCode(
                long postCode,
                ADR_CODE *adrCode )
```

引数

(in)
　postCode : 郵便番号
　　7桁の郵便番号を数値に変換したもの

(out)
　adrCode : 住所コード

戻り値

0 : 正常終了
負数 : エラー
-1 : 郵便番号インデックスレコードがメモリにロードされていない
-2 : 指定された郵便番号が見つからない
-3以下 : その他のエラー

説明

郵便番号を住所コードに変換します。
７桁の郵便番号からは町大字レベルの住所コードが得られます。

関数名　gcjAdrCode2Point()

機能　住所コードが示す場所の代表点の緯度経度を取得する

宣言

```
int gcjAdrCode2Point(
                ADR_CODE *adrCode,
                double *latitude,
                double *longitude)
```

引数

(in)
　adrCode　：住所コード
(out)
　latidute　：緯度(度単位)
　longitude：経度(　〃　)

戻り値

　0 ：正常終了
負数 ： エラー
　-1：行政区域 取得段階でのエラー
　-2：町大字　　〃
　-3：丁目字　　〃
　-4：地番街区符号　〃
　-5：枝番住居番号　〃
　-6：指定された住所コードには緯度経度データが無い

説明

住所コードが示す場所は、住所コードに格納されているコードレベルによって、都道府県の広域から街区符号、住居番号といった狭い範囲まで色々です。この関数は、その指定された住所コードの表す区域の、代表点の緯度経度を返します。
緯度経度は度単位の実数で返されます。
測地系は住所DBを作成した元データの測地系と同じです。元データが街区レベル位置参照情報の場合は世界測地系で、IPC社のデータの場合は東京測地系です。

関数名　gcjAdrCode2Str()

機能　住所コードを住所文字列に変換する

宣言

```
int gcjAdrCode2Str(
        ADR_CODE *adrCode,
        char *kanji,
        int kanjiBytes,
        char *kana,
        int kanaBytes )
```

引数

(in)
- adrCode　: 住所コード(構造体)
- kanji　: 漢字住所を格納するバッファ(漢字住所が不要な場合はNULL)
- kanjiBytes : 漢字住所を格納するバッファのバイト数
- kana　: 仮名住所を格納するバッファ(仮名住所が不要な場合はNULL)
- kanaBytes : 仮名住所を格納するバッファのバイト数

(out)
- kanji　: 住所文字列(漢字)
- kana　: 住所文字列(仮名)

戻り値

0以上 ： 変換できたレベル
- 5 ： 枝番住居番号まで
- 4 ： 地番街区符号まで
- 3 ： 丁目字名まで
- 2 ： 町大字名まで
- 1 ： 行政区域名まで
- 0 ： 変換できなかった

負数 ： エラー
- -1 ： 行政区域名 取得段階でのエラー
- -2 ： 町大字名　〃
- -3 ： 丁目字名　〃
- -4 ： 地番街区符号　〃
- -5 ： 枝番住居番号　〃
- -6 ：文字コード変換におけるエラー

説明

住所コードを住所文字列に変換します。
変換の際のフォーマットはgcjSetAdrStringFormat () で指定された内容となります。指定されていないと、既定のフォーマットで出力されます。

関数名　gcjSetAdrStringFormat()

機能　住所文字列の出力形式を設定する

宣言

```
int gcjSetAdrStringFormat(
                int formatNumber,
                int mode )
```

引数

(in)

formatNumber　表示項目

0 : 全ての表示
1 : 都道府県名の表示
2 : 政令指定都市名の表示
3 : 支庁名の表示
4 : 郡名の表示
5 : 市区町村名の表示
6 : 町・大字名の表示
7 : 丁目・字名の表示
8 : 地番、街区符号の表示
9 : 地番の枝番、住居番号の表示

20 : "丁目"の文字を表示
　表示しない場合は"-"で区切られる
21 : "番地"、住居表示の"番"、"号"の文字を表示
　表示しない場合は"-"で区切られる
22 : 丁目の数字を漢字で表示
　表示しない場合は算用数字で表示

mode　表示のON、OFF

0 : 表示しない
1 : 表示する

・既定の表示は全てmode=1とした表示方法。
・formatNumber=0でmode=0とすると全ての表示がmode=0とした表示となる。

戻り値

0 ： 正常終了
-1 ： 指定された項目番号はサポートしていない

説明

住所コードを住所文字列に変換する際のフォーマットを指定します。
表示項目の番号のうち、１～9はその項目を表示するかしないかの設定です。20以上は表示する場合の形式を設定します。

関数名　gcjGetNumAdrRecords()

機能　指定した住所レベルのレコード数を取得する

宣言

```
int gcjGetNumAdrRecord(
                ADR_CODE *adrCode )
```

引数

(in)
adrCode : レコード数を取得する住所コード
どの住所レベルのレコード数を取得するかをcodeLevelフィールドで指定する。
・指定できるレベルは
５:町大字のレコード数
６:丁目字のレコード数
７:地番街区符号のレコード数
８:枝番住居番号のレコード数
のみである。

戻り値

0 以上 ： レコード数
負数 ： エラー
-1 ： 指定されたcodeLevelより上位のコード番号が無い
-2 ： 指定されたcodeLevelはサポートされていない。
-3 ： 必要な住所DBがロードされていない
-4 ： コード番号が正常な範囲でない
-5 ： その他のエラー

説明

ある県の市区町村数がいくつあるかなど、住所データベースに登録されているデータについて知りたい時に、データのレコード数を取得します。
取得したい住所区域の種類をadrCode構造体のcodeLevelフィールドに設定し、そのレベルより上位のコード番号を各フィールドに設定します。

各レコードの内容を取得する場合は、この関数でレコード数を取得し、そのレコード数分のInt配列を確保します。次に、gcjGetAdrRecordList()を呼び出し、コード番号のリストを取得します。そして、そのリストにあるコードについて、実際のレコードをgcjGetRecord()で取得します。

使用例

東京都新宿区の町大字の数を知りたい場合:

```
ADR_CODE adrCode;

adrCode.codeLevel = 5;          // 町大字区域を表すコードは5
adrCode.choCode = 13104;    // 新宿区の市区町村コードは13104


numRecords =  gcjGetNumAdrRecords( &adrCode );  // numReordsに町の数
```

関数名　gcjGetAdrRecordList()

機能　指定した住所レベルのレコードのリストを取得する。

宣言

```
int gcjGetAdrRecordList(
                ADR_CODE *adrCode,
                int *codeArray,
                int recordMax )
```

引数

(in)

adrCode : レコードリストを取得する住所コード
どの住所レベルのレコードリストを取得するかをcodeLevel
フィールドで指定する。gcjGetNumAdrRecords()参照。

recordMax : コード番号を格納する配列の要素数

(out)

codeArray : コード番号を格納する配列

戻り値

0 以上 ： 格納したレコード数
負数 ： エラー
-1 ： 指定されたcodeLevelより上位のコード番号が無い
-2 ： 指定されたcodeLevelはサポートされていない。
-3 ： 必要な住所DBがロードされていない
-4 ： コード番号が正常な範囲でない
-5 ： その他のエラー

説明

指定した住所レベルのレコードのリストを取得します。リストはコード番号の配列の形で得られます。このリストに格納されているコード番号についてgcjGetRecord()を呼び出す事で実際のレコードを取得できます。

使用例

東京都新宿区の町大字レコードのリストを取得する場合

```
ADR_CODE adrCode;
int *codeArray;

adrCode.codeLevel = 5;          // 町大字区域を表すコードは5
adrCode.choCode = 13104;    // 新宿区の市区町村コードは13104

numRecords =  gcjGetNumAdrRecords( &adrCode );  // numReordsに町の数
                                                が返されます。
codeArray = (int*) malloc( numRecords, sizeof(int) );
numCodes = gcjGetAdrRecordList( &adrCode, codeArray, numRecords );
                // codeArrayの配列にコード番号が格納されます。
```

関数名　gcjGetAdrRecord()

機能　指定した住所コードのレコードを取得する。

宣言

```
int gcjGetAdrRecord(
                ADR_Code *adrCode,
                void* record
                 int recordBytes )
```

引数

(in)
　adrCode : レコードを取得する住所コード
　　どの住所レベルのレコードを取得するかをcodeLevel
　　フィールドで指定する。
　recordBytes : レコードバッファのバイト数
(out)
　record ：取得したレコードを格納するバッファ

戻り値

0　：正常終了
負数 ：エラー
-1　：指定されたcodeLevelより上位のコード番号が無い
-2　：指定されたcodeLevelはサポートされていない。
-3　：必要な住所DBがロードされていない
-4　：コード番号が正常な範囲でない
-5　：バッファのバイト数が足りない
-6　：その他のエラー

説明

指定した住所レベルのレコードのリストを取得します。リストはコード番号の配列の形で得られます。このリストに格納されているコード番号についてgcjGetRecord()を呼び出す事で実際のレコードを取得できます。

使用例

東京都新宿区の町大字レコードの最初のレコードを取得する場合

```
 ADR_CODE adrCode;
int *codeNums;
CHO_RECORD choRecord;

adrCode.codeLevel = 5;          // 町大字区域を表すコードは5
 adrCode.cityCode = 13104;    // 新宿区の市区町村コードは13104

 // numReordsに町の数が返されます。
 numRecords =  gcjGetNumAdrRecords( &adrCode );

 // 町の数分のintの配列を確保します。
```

```
 codeNums = (int*) malloc( numRecords * sizeof(int) );

// codeNumsの配列にコード番号が格納されます。
numCodes = gcjGetAdrRecordList( &adrCode, codeNums, numRecords );

// 最初のレコードがrecordに格納されます。
adrCode.choCode = codeNums[0];
result = gcjGetAdrRecord( &adrCode, (void*) &choRecord, sizeof(record) );
```

構造体リファレンス

名称　ADR_CODE

用途　**住所コードを格納する。**

宣言

```
typedef struct {
        int codeLevel;    /* コードが有効なレベル  */
        int cityCode;     /* 市区町村コード(JIS) 5桁        :レベル4  */
        int choCode;      /* 町・大字コード(独自)           :レベル5  */
        int azaCode;      /* 小字コード(独自)               :レベル6  */
        int gaikuCode;    /* 地番、街区符号コード(独自)     :レベル7  */
        int jukyoCode;    /* 地番の枝番、住居番号           :レベル8  */
} ADR_CODE
```

説明　住所は都道府県から始まって、地番、住居番号まで、場所を表す文字列が階層化されて並んでいます。この構造体はその階層化された住所をコードで表します。

名称 GYOSEI_HEADER

用途 行政区域レコードファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[8];      /* 未使用 */
    int numRecords;       /* 行政区域レコード数 */
} GYOSEI_HEADER;
```

説明

行政区域レコードファイルのヘッダーを読み書きする際に使用します。

名称 GYOSEI_RECORD

用途 行政区域レコードを格納

宣言

```
typedef struct {
    int shubetsu;                       /* 種別 */
    int kenCode;                        /* 県コード2桁 */
    int shichoCode;                     /* 支庁コード5桁 */
    int gunCode;                        /* 郡政令市コード5桁 */
    int cityCode;                       /* 市区町村コード5桁 */
    int removeFlag;                     /* 削除フラグ */
    int newGyoseiCode[NEW_CODE_MAX]; /*変更後の市区町村コード
*/
    char name[GYOSEI_NAME_LEN+1];       /* 漢字名称 */
    char kana[GYOSEI_KANA_LEN+1];       /* 仮名名称 */
    int choHashTableNum;                /* 町大字ハッシュテーブル
番号 */
    int latitude;                       /* 緯度（ミリ秒）*/
    int longitude;                      /* 経度（ミリ秒）*/
} GYOSEI_RECORD;
```

説明

都道府県から市区町村までの区域を表すための構造体です。

行政区域レコードファイルのアクセスと主要な処理で使用します。

名称 GYOSEI_HASH_HEADER

用途 行政区域名ハッシュテーブルファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[4];      /* 未使用 */
    int numRecords;       /* ハッシュレコード数 */
    int hashSize;         /* ハッシュサイズ */
} GYOSEI_HASH_HEADER;
```

説明

行政区域レコードファイルのヘッダーを読み書きする際に使用します。

名称 GYOSEI_HASH_BUCKET

用途 行政区域検索用名称のレコードを格納

宣言

```
typedef struct {
    char name[GYOSEI_NAME_LEN+1]; /* 検索用名称 */
    int gyoseiCode;               /* 行政区域コード */
    int shubetsu;                 /* 種別 */
    int nextBucketNum;            /* 次のバケット番号 */
} GYOSEI_HASH_BUCKET;
```

説明

行政区域名の検索をハッシュ値で行いますが、その検索の際に使用します。
通常のライブラリ使用ではこの構造体は使いません。

名称 CHO_HEADER

用途 町大字レコードファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];         /* ファイルID */
    int version;            /* バージョン */
    char notUsed[8];        /* 未使用 */
    int numRecords;         /* 町大字レコード数 */
    int numBytes;           /* 1レコードのバイト数 */
    int offsetRecordTop;    /* レコード先頭のバイト位置 */
} CHO_HEADER;
```

説明

町大字レコードファイルのヘッダーを読み書きする際に使用します。

名称 CHO_OFFSET

用途 町大字レコードの格納位置

宣言

```
typedef struct {
    int numRecords;         /* 町大字レコード数 */
    int startRecNum;        /* 先頭のレコード番号 */
} CHO_OFFSET;
```

説明

町大字レコードは市区町村毎に格納されていますが、各市区町村のレコードがどこから始まるか示します。

名称 CHO_RECORD

用途 町大字レコードを格納

宣言

```
typedef struct {
    int choCode;                /* 町大字コード */
    int reserved;               /* 予備 */
    char name[CHO_NAME_LEN+1]; /* 町大字名 */
    char kana[CHO_KANA_LEN+1]; /* 読み仮名 */
    int postCode;               /* 郵便番号 */
    int azaTableNum;            /* 丁目字テーブル番号 */
    int chibanTableNum;         /* 地番街区符号テーブル番号*/
    int latitude;               /* 緯度（ミリ秒）*/
    int longitude;              /* 経度（ミリ秒） */
} CHO_RECORD;
```

説明 町大字のレコードを格納します。
町大字レコードファイルのアクセスと主要な処理で使用します。

名称 CHO_HASH_HEADER

用途 **町大字名ハッシュテーブルファイルのヘッダー**

宣言

```
typedef struct{
    char fileId[8];         /* ファイルID */
    int version;            /* バージョン */
    char notUsed[16];       /* 未使用 */
    int hashTableBytes;     /* ハッシュテーブルのバイト数 */
    int numRecords;         /* 名称レコード総数 */
} CHO_HASH_HEADER;
```

説明

町大字ハッシュテーブルファイルのヘッダーを読み書きする際に使用します。

名称 CHO_HASH_BUCKET

用途 **町大字検索用名称のレコードを格納**

宣言

```
typedef struct {
    char name[CHO_NAME_LEN+1];      /* 検索用名称 */
    int choCode;                    /* 町大字コード */
    int nextBucketNum;              /* 次のバケット番号 */
} CHO_HASH_BUCKET;
```

説明

町大字名の検索をハッシュ値で行いますが、その検索の際に使用します。
通常のライブラリ使用ではこの構造体は使いません。

名称 AZA_HEADER

用途 丁目字レコードファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[8];      /* 未使用 */
    int maxNumTables;     /* 未使用 */
    int numTables;        /* 丁目字テーブル数 */
    int numTotalRecords; /* 丁目字レコード総数 */
} AZA_HEADER;
```

説明 丁目字レコードファイルのヘッダーを読み書きする際に使用します。

名称 AZA_TABLE_LIST

用途 丁目字テーブルのリスト

宣言

```
typedef struct {
    int cityCode;           /* ５桁市区町村コード */
    int choCode;            /* 町大字コード */
    int numRecords;         /* 丁目字レコード数 */
    int numSerialChomoku;   /* 連続丁目の数 */
    int startRecNum;        /* テーブル先頭のレコード番号 */
} AZA_TABLE_LIST;
```

説明 丁目字レコードは町大字毎にまとまった形(テーブル)で格納されていますが、各町大字のテーブルがどこから始まるかなどの情報を格納します。

名称　AZA_RECORD

用途　丁目字レコードを格納

宣言

```
typedef struct {
    int choazaCode;             /* 町大字コード＋丁目字コード */
    int reserved;               /* 予備 */
    char name[AZA_NAME_LEN+1]; /* 丁目字漢字名称 */
    char kana[AZA_KANA_LEN+1]; /* 丁目字仮名名称 */
    int chibanTableNum;         /* 地番街区符号テーブル番号 */
    int latitude;               /* 緯度（ミリ秒） */
    int longitude;              /* 経度（ミリ秒） */
} AZA_RECORD
```

説明　丁目字のレコードを格納します。
丁目字レコードファイルのアクセスと主要な処理で使用します。

名称 GAIKU_HEADER

用途 地番街区符号レコードファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[8];      /* 未使用 */
    int maxNumTables;     /* 未使用 */
    int numTables;        /* 地番街区符号テーブル数 */
    int numTotalRecords; /* 地番街区符号レコード総数 */
} GAIKU_HEADER;
```

説明

地番街区符号レコードファイルのヘッダーを読み書きする際に使用します。

名称 GAIKU_TABLE_LIST

用途 地番街区符号テーブルのリスト

宣言

```
typedef struct {
    int cityCode;          /* ５桁市区町村コード */
    int choCode;           /* 町大字コード */
    int azaCode;           /* 丁目字コード */
    int numRecords;        /* 地番街区符号レコード数 */
    int startRecNum;       /* テーブル先頭のレコード番号 */
    short jukyoHyoji;      /* 住居表示フラグ */
} GAIKU_TABLE_LIST;
```

説明

地番街区符号レコードは丁目字毎にまとまった形(テーブル)で格納されていますが、各丁目字のテーブルがどこから始まるかなどの情報を格納します。

名称　GAIKU_RECORD

用途　地番街区符号レコードを格納

宣言

```
typedef struct {
    int gaikuCode;          /* 地番街区符号コード */
    int jukyoTableNum;      /* 枝番住居番号テーブル番号 */
    int latitude;           /* 緯度（ミリ秒） */
    int longitude;          /* 経度（ミリ秒） */
} GAIKU_RECORD;
```

説明

地番街区符号のレコードを格納します。
地番街区符号レコードファイルのアクセスと主要な処理で使用します。

名称 JUKYO_HEADER

用途 枝番住居番号レコードファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[8];      /* 未使用 */
    int maxNumTables;     /* 未使用 */
    int numTables;        /* 枝番住居番号テーブル数 */
    int numTotalRecords; /* 枝番住居番号レコード総数 */
} JUKYO_HEADER;
```

説明

枝番住居番号レコードファイルのヘッダーを読み書きする際に使用します。

名称 JUKYO_TABLE_LIST

用途 枝番住居番号テーブルのリスト

宣言

```
typedef struct {
    int numRecords;     /* 上位16ビット : 上位コードのチェックサム */
                        /* 下位16ビット : 枝番住居番号レコード数 */
    int startRecNum;    /* テーブル先頭のレコード番号 */
```

説明

枝番住居番号レコードは地番街区符号毎にまとまった形(テーブル)で格納されていますが、各地番街区符号のテーブルがどこから始まるかなどの情報を格納します

名称　JUKYO_RECORD

用途　**枝番住居番号レコードを格納**

宣言

```
typedef struct {
    int jukyoCode;          /* 枝番住居番号コード */
    int latitude;           /* 緯度（ミリ秒） */
    int longitude;          /* 経度（ミリ秒） */
} JUKYO_RECORD;
```

説明　枝番住居番号のレコードを格納します。
枝番住居番号レコードファイルのアクセスと主要な処理で使用します。

名称 STRING_HEADER

用途 地番文字列レコードファイルのヘッダー

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[12];     /* 未使用 */
    int recordBytes;      /* １レコードのバイト数 */
    int numRecords;       /* 文字列レコード数 */
} STRING_HEADER;
```

説明

地番文字列レコードファイルのヘッダーを読み書きする際に使用します。

名称 STR_RECORD

用途 文字列レコードを格納

宣言

```
typedef struct {
    char str[CHIBANSTR_BYTES];  /* 文字列 */
} STR_RECORD;
```

説明

地番、枝番、街区符号など、通常は数字ですが、たまに文字列である場合があります。
そのような場合にその文字列を格納します

名称 POST_INDEX_HEADER

用途 **郵便番号インデックスファイルヘッダー**

宣言

```
typedef struct{
    char fileId[8];       /* ファイルID */
    int version;          /* バージョン */
    char notUsed[16];     /* 未使用 */
    int numRecords;       /* レコード数 */
} POST_INDEX_HEADER;
```

説明

郵便番号インデックスファイルのヘッダーを読み書きする際に使用します。

名称 POST_INDEX

用途 **郵便番号インデックスを格納**

宣言

```
typedef struct {
    int postCode;       /* 郵便番号 */
    int cityCode;       /* 市区町村コード */
    int choCode;        /* 町大字コード */
} POST_INDEX;
```

説明

郵便番号から市区町村コードと町大字コードを取得するのに使用します。

オープンソースジオコーダ
jeocoder.ja
マニュアル

株式会社オークニー
〒231-0002 横浜市中区海岸通1-2　JA共済横浜ビル6階
Tel 045-228-3320　Fax 045-228-3321