

## 関数リファレンス

**関数名****gcjDbSetPath()****機能**

住所データベースのディレクトリを設定する。

**宣言**

```
int gcjDbSetPath(  
    char *path )
```

**引数**

(in)  
path : 住所データベースが格納されているディレクトリ

**戻り値**

0 : 正常終了  
-1: 指定されたディレクトリが存在しない

**説明**

住所データベースが格納されているディレクトリを設定します。  
以降、他の関数はここで指定されたディレクトリの各ファイルを参照します。その時に、必要なファイルが存在しない場合はエラーとなります。必要なファイルは住所データベースの一覧をご覧ください。

**注意事項**

## 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 : 使用する住所のレベル

1 : 行政区域(県、郡、市区町村レベル)

2 : 町大字

3 : 丁目字

4 : 地番、街区符号

5 : 枝番、住居番号

## 戻り値

0 : 正常終了

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        "

--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。  
指定したレベルより上位のデータも同時にロードされます。

## 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。

<b>関数名</b>	<b>gcjDbEnd()</b>
<b>機能</b>	住所データベースに関する処理を終了する。
<b>宣言</b>	<code>int gcjDbEnd()</code>
<b>引数</b>	なし
<b>戻り値</b>	0 : 正常終了
<b>説明</b>	住所データベース用に確保したメモリを解放します。
<b>注意事項</b>	

**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)  
encoding : 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ことがあります。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

0 : 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  
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（通常起きないエラー）

## 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 : 指定された郵便番号の数値が正常な範囲でない  
-2 : 指定された郵便番号に対応する住所が見つからなかった  
-3以下 : 検索上のエラー(通常起きないエラー)

**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)  
latitude : 緯度(度単位)  
longitude : 経度( " ")

**戻り値**

0 : 正常終了  
負数 : エラー

- 1 : 指定された郵便番号の数値が正常な範囲でない
- 2 : 指定された郵便番号に対応する住所が見つからなかった
- 3以下 : 検索上のエラー(通常起きないエラー)

**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、世界測地系での緯度経度数値です。

## 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       "

## 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

- 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 : 小字名の表示
- 8 : 地番、街区符号の表示
- 9 : 地番の枝番、住居番号の表示

20 : “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

21 : 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

22 : “丁目”の文字を表示  
表示しない場合は“-”で区切られる

`mode` 表示のON、OFF

- 0 : 表示しない
- 1 : 表示する

**戻り値** 0 : 正常終了

負数 : エラー

-1 : `formatNumber`の数値がサポート範囲外である。

## 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

## 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

## 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_RECORD *adrRecord,  
    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。  
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 : 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

## 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_RECORD *adrRecord,  
    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 以上 : 正常終了  
負数 : エラー  
      -1 : 指定された住所コードに対応するレコードが無い  
      -2 : バッファの容量が足りない

## 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  
  
// numRecordsに町の数返されます。  
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) );
```