Critchlow Associates
BUSINESS MAPPING SOLUTIONS

GeoStan™ New Zealand Reference Manual

Version 1.2

GeoStan New Zealand Reference Manual

Document Number: 0800

Contents

Chapter 1

About GeoStan New Zealand

This manual details the use and functionality of the GeoStan New Zealand library. The GeoStan New Zealand library allows the developer to incorporate address standardization and address geocoding into virtually any application.

The process of standardizing and geocoding an address is very straightforward using these functions and gives the developer a large amount of freedom in implementing the user interface.

A standardized urban address will contain the street address, suburb, city / town, and Post Code. A standardized rural address will contain the street address, Rural Delivery number, locality, and Post Code. A geocoded address will contain the address as found in the geocode database, as well as the COORDX, COORDY, and census meshblock values. A detailed match code is also returned for each process.

Installing GeoStan New Zealand

For information on installing and using GeoStan New Zealand, please refer to the installation notes that came with the CD.

About This Manual

This manual contains chapters on the methodologies employed for address standardization and address geocoding, collectively referred to as address matching. This manual also explains the general use of the functions and includes a complete function reference.

Within this manual, all language references that appear embedded in descriptive text, including function, macro and variable references, will be shown in a monospaced Courier font:

```
x = 1; // Just a sample
```

Function names embedded in text are shown in **`bold`**.

Both decimal and hexadecimal numbers may be used in this document. Hexadecimal numbers are always preceded by 0x:

```
0x1F  // hex 1F
0x04  // hex 04
12    // decimal 12
```

Within GeoStan New Zealand, you will find that all of the public functions begin with **`GsNZ`**, such as **`GsNZClear`**. Where possible, you should avoid using variables, constants or macros beginning with **`GsNZ`**. Check the include file in your development environment's directory for a complete list of visible symbols.

Contacting Technical Support

If you have any technical questions regarding the use of GeoStan New Zealand, contact Critchlow Associates' Help Desk.

Critchlow Associates
Help Desk hours: 0830 - 1700 hrs Monday-Friday (New Zealand Time)

(64) 4 472 8244 or 0800 627 746 (voice)
(64) 4 472 6695 (fax)
`gs_support@critchlow.co.nz`

Chapter 2

Basic Concepts

The functions described here work on files called GeoStan New Zealand Directory files, which are encoded in a proprietary format. These files are referred to as “address directory” or GSD files throughout this document and have as part of their name “.GSD”. The primary GSD file is called NZ.GSD.

Valid Addresses

GeoStan New Zealand uses sophisticated algorithms to help improve match rates for poorly formed input addresses, but addresses should be as close to New Zealand Post standards as possible for the highest match rate. An address is required to have at least a street name, and either a suburb / ward / city / town / locality name or Post Code in order for it to be processed. If any address elements are missing and the address was matched, you can retrieve the rest of the information about the street address using functions within GeoStan New Zealand. See the next section for a discussion of address elements.

Street intersections may be geocoded but may not return all information, such as Post Code, since it may not be possible to accurately return a single value.

GeoStan New Zealand performs a number of “enhanced” matches as well. These are performed automatically and do not require any special setup or usage.

Hyphen and Slash Address Support

GeoStan New Zealand handles hyphens and slashes ('/') in addresses containing unit numbers. Specifically, a hyphen or slash is used between a unit number and a house number. For example:

7-128 Cooke Street, or

7/128 Cooke Street

Either of these addresses resolves to Unit 7 in the building at 128 Cooke Street.

Address Elements

A street address consists of the following elements:

- Flat Number
- House Number (optional)
- Prefix Direction (i.e. North, South, East, West)
- Street Name
- Street Type/Suffix (e.g. Road, Drive, Street)
- Postfix Direction

Some elements may not be present for all addresses. Unit type and range (e.g. Flat 3C) may also be provided. These street data are passed to GeoStan New Zealand as a single element. The library parses the individual street elements to separate fields, and library functions are used to retrieve each parsed street element.

Note: GeoStan New Zealand will reassemble a pre-parsed address before processing it. This is done because the GeoStan New Zealand parser was also used in the loading of the data, and it will ensure that the address is parsed in the same manner as it was loaded.

Do not try to set discrete address fields using address elements. GeoStan New Zealand's parsing technology is designed to optimally match whole address lines. Using discrete fields bypasses the parsing mechanism and defeats the optimizations built into GeoStan New Zealand.

An internal element, parity, is also determined by calculating if the House Number is even or odd. Since GeoStan New Zealand street files include information on which side of the street contains the odd addresses and which side the even, parity is used to resolve COORDX/COORDY assignments. Parity is also used in conjunction with the file meshblk.gsb to assign meshblocks.

Not all street addresses contain all elements. The address "123 Elm St." does not have any directionals or unit information, yet is a valid address. Street suffix (also called type), Pre-and Post-directional elements, and house numbers may not be critical elements of some addresses. By comparing an address as input against all known addresses, GeoStan New Zealand can usually determine if any of these elements were missing or incorrect. If there is more than one feasible address in the GeoStan New Zealand database, then a multiple match occurs, which can be resolved by the developer, or the actual user.

A house number is optional when matching to an address. New Zealand contains streets that have no associated house numbers. See “Internal Data Structure and Concepts” on page 47 for more detailed information.

Intersection Addresses

The address line (`GSNZ_ADDRLINE`) for an intersection matche may be formatted in either of two ways:

1. <street 1> <separator> <street 2>

   where:

   <street 1> and <street 2> = <pre dir> <street name> <street type> <post dir>

   <separator> = "AND", "AT", "@", or "&"

2. <corner specifier> <street 1> <separator> <street 2>

   <corner specifier> = "CORNER OF", "CORNER", "CORN OF", "CORN", "CRNR OF", "CRNR", "CNR OF", "CNR", "COR OF", or "COR"

For example, an intersection match could be formatted as "Weymouth and Gainsborough" (method one) or as "corner of Weymouth and Gainsborough" (method two).

Note: The supplemental file NZT.GSD is required in order to do intersection matching.

Street Name Matching

GeoStan New Zealand incorporates a sophisticated scoring mechanism that compares two street names to determine the likelihood of a match. This mechanism will allow a match, even though minor misspellings or transpositions may occur. Dropped characters, transpositions, etc., are all parts of a street name’s match score.

Address Match Methodology

GeoStan New Zealand searches for and resolves addresses using the following process:

1. Search all of New Zealand for matching street names. Potential (possible) matches are assigned to a list.
2. Score each potentional match.

3. For each possible match, every address element is scored (assigned a confidence level from 0-100) and the scores are weighted. The sum of these scores is the final match score. A potential match with a score of 0 exactly matches all scored address elements.
4. The scoring system takes into account all anomalies that should differentiate addresses. This includes “identical” address ranges, where one range is for a street and the other range is for a high-rise.
5. If the lowest-scoring match has a unique score, it is returned as the matching address.
6. If two or more matches share the lowest score, you will need to supply routines that resolve the multiple match.

Multiple Match Resolution

GeoStan New Zealand may find more than one address that has the best rating. In this instance, GeoStan New Zealand cannot determine on its own which record is correct and therefore allows the developer to resolve the multiple matches.

When a multiple match collision occurs, **`GsNZFind`** returns the value `GSNZ_ADDRESS_NOT_RESOLVED`, as well as match code EO23. Using the functions **`GsNZNumMultiple`** and **`GsNZMultipleGet`**, the developer may get the list of possible matches, present them to the user, and allow the user to select one of the choices. Alternatively, the developer can resolve the matches internally.

Match Location

An address-match location is returned in COORDX and COORDY values interpolated from the street segments available in the database, including shape points. Street coordinate values are returned in whole meters on the New Zealand Map Grid, guaranteeing that the returned point will display very well on top of base maps derived from the same street network. The location is derived by calculating the percentage of the range in which the house number falls and placing the location of that same percentage along the segment in which the address is found. The location assignment correctly handles shape points that define the segment. If an offset distance is used, that distance is calculated perpendicular to the portion of the street segment in which it lands. This will yield the best visual representation for any mapping package, as well as be the most accurate location able to be derived from the geographic data. Meshblock data is returned for the side of the segment on which a match was found.

Optimization Issues

The GeoStan New Zealand data format and functions were designed and developed to allow data to be used directly from CD. Records will be processed most quickly if they are first sorted in street name order.

Chapter 3

Using GeoStan New Zealand

This chapter explains the general use of the GeoStan New Zealand function set and how to incorporate these functions into your application.

Basic Use of GeoStan New Zealand

The GeoStan New Zealand functions all use a common internal data structure that is initialized with a call to **`GsNZInit`**. This function returns a `GsNZId` that is then passed to every other GeoStan New Zealand function. Upon completion of the geocoding session, a call to **`GsNZTerm`** frees the memory allocated by **`GsNZInit`** and closes all files used by GeoStan New Zealand.

Each GeoStan New Zealand function returns either a status code or a pointer. There is a symbolic constant for every status code returned by a function. If the function returns a pointer, there is usually only one possible error that would be indicated by the return of `NULL`. "Function Reference", starting on page 15, details all possible status codes for each function.

Get and Set Functions

GeoStan New Zealand provides a variety of functions to load and retrieve data from its internal data structures. The Set functions require a "Switch" and a "Value" argument. The Switch argument is a symbolic constant that identifies the data element to be stored. The Value argument is the actual data string to be stored. Refer to "Common Enums for Storing and Retrieving Data" on page 55 for Switch constant definitions, and to "Function Reference" on page 15 for an explanation of each Set function's Value argument. The Get functions require the same Switch arguments used by the Set function and a pointer to a buffer to be filled with the returned data, and an argument specifying the maximum length of data that GeoStan New Zealand should return.

"Spell-checking" Input Locations

When matching an address, it is sometimes desirable to retrieve the standardized input location name, rather than the official name that was matched (this is also known as "spell-checking" a name). You may do this by calling **`GsNZDataGet`** with `GSNZ_OUTPUT` as the *`fInOut`* parameter retrieving `GSNZ_LOCATION1` or `GSNZ_LOCATION2` (used as the *`fSwitch`* parameter).

Location 1 and Location 2 are the suburb and city names used in the input address. The library chooses which input fields are assigned to Locations 1 and 2 based on the following table:

Input Suburb Field	Input City Field	Location 1	Location 2
Blank	Blank	Blank	Blank
Not blank	Blank	Suburb	Blank
Blank	Not blank	City	Blank
Not blank	Not blank	Suburb	City

For example, imagine that you wanted to match an address with the following suburb and city:

	Input	Matched Location (Output)
City	Akld	Auckland
Suburb	Hendersen	Henderson

Calling **`GsNZDataGet with`** `GSNZ_LOCATION1` and `GSNZ_LOCATION2` would result in the values `GSNZ_LOCATION1` = Henderson and `GSNZ_LOCATION2` = Auckland.

Note: If the match fails, GSNZ_LOCATION* may return the input value.

Elements Available via GeoStan New Zealand

A complete list of data available through GeoStan New Zealand can be found in “Common Enums for Storing and Retrieving Data” starting on page 55.

Basic Matching

Once GeoStan New Zealand has been initialized, there are many options available through the library to allow diverse implementations of matching strategies. A basic approach is outlined below:

1. Initialize GeoStan New Zealand using **`GsNZInit`**.
2. Load all components of an address using **`GsNZDataSet`**.
3. Ask GeoStan New Zealand to find the loaded address by calling **`GsNZFind`**.
4. Retrieve the match information by calling **`GsNZDataGet`**.
5. Clear the GsNZ data buffer by calling **`GsNZClear`**.
6. Repeat steps 2 through 5 for each address to be geocoded.
7. Close GeoStan New Zealand using **`GsNZTerm`**.

Below is a sample code fragment, in C, that illustrates the use of GeoStan New Zealand to match an address and communicate with the user.

```
GsNZFunStat retcode;                     /* return code from library call */
GsNZId gs;                                   /* GeoStan New Zealand ID */
char    firm[GSNZ_FIRM_NAME_LENGTH],         /* firm name */
        address[GSNZ_ADDRLINE_LENGTH],       /* street address */
        suburb[GSNZ_SUBURB_LENGTH],          /* suburb name */
        lastline[GSNZ_LASTLINE_LENGTH],      /* city and/or Post Code */
        coordx[GSNZ_COORDX_LENGTH],          /* NZ Map Grid coord X */
        coordy[GSNZ_COORDY_LENGTH];          /* NZ Map Grid coord Y */
char Message[256], Detail[256];              /* error message output */
GsNZInitStruct gsInit;                       /* initialization structure */

/* Initialize the library */

memset( &gsInit, 0, sizeof(GsNZInitStruct) );

/* GeoStan New Zealand version number */
gsInit.version = GSNZ_GEOSTAN_VERSION;

/* enums that specify the components to initialize */
/* load files for address geocoding */
gsInit.options = GSNZ_FILE_ADDR_CODE;

/* delimited list of paths to search for initialization files */
strncpy(gsInit.paths, "C:\\NZData", GSNZ_MAX_STR_LEN);

/* license password */
gsInit.licFilePassword = 11111111L;

/* path and name of license file */
strncpy(gsInit.licFileName, "C:\\NZData\\gsnz.lic", GSNZ_MAX_STR_LEN);

/* relative cache to create and use */
gsInit.cacheSize = GSNZ_CACHESIZE_LARGE;

/* file status - bit flags describing which files were loaded */
gsInit.status = 0;

/* string of the latest date to use in initialization */
/* format is YYYYMMDD */
memset(gsInit.relDate, 0, GSNZ_MAX_STR_LEN);

gs = GsNZInit( &gsInit );
```

```
/* initialization failed */
if ( gs == 0 ) {
    fprintf( stderr, "GsNZInit failed, status: 0x%04x.\n", gsInit.status );
    return 1;
}

GsNZDataSet( gs, GSNZ_DIST, "15" ); /* offset 15 meters from center line */

/* load an address record into GeoStan New Zealand */
GsNZClear( gs );
GsNZDataSet( gs, GSNZ_FIRM_NAME,   "Critchlow Associates Ltd" );
GsNZDataSet( gs, GSNZ_ADDRLINE,    "Brandon Street" );
GsNZDataSet( gs, GSNZ_SUBURB_NAME, "Wellington" );
GsNZDataSet( gs, GSNZ_LASTLINE,    "Wellington" );

/* standardize and geocode the address */
retcode = GsNZFind( gs, GSNZ_ADDR_CODE );

/* analyze the result */
switch( retcode )
{
    case GSNZ_SUCCESS:
        /* retrieve standardized address  */
        GsNZDataGet( gs, GSNZ_OUTPUT, GSNZ_FIRM_NAME, firm,
    GSNZ_FIRM_NAME_LENGTH );
        GsNZDataGet( gs, GSNZ_OUTPUT, GSNZ_ADDRLINE, address,
    GSNZ_ADDRLINE_LENGTH );
        GsNZDataGet( gs, GSNZ_OUTPUT, GSNZ_SUBURB_NAME, suburb,
    GSNZ_SUBURB_LENGTH );
        GsNZDataGet( gs, GSNZ_OUTPUT, GSNZ_LASTLINE, lastline,
    GSNZ_LASTLINE_LENGTH );

        /* retrieve geocode */
        GsNZDataGet( gs, GSNZ_OUTPUT, GSNZ_COORDY, coordy, GSNZ_COORDY_LENGTH
    );
        GsNZDataGet( gs, GSNZ_OUTPUT, GSNZ_COORDX, coordx, GSNZ_COORDX_LENGTH
    );

        /* output the results */
        printf( "%s\n"
                "%s\n"
                "%s\n"
                "%s\n"
                "coordx: %lf\n"
                "coordy: %lf\n",
                firm, address, suburb, lastline, coordx, coordy );
        break;

    case GSNZ_ERROR:/* Each error can be handled */
        GsNZErrorGet(gs, Message, Detail);
        fprintf( stderr, "%s\n%s.\n", Message, Detail );
    case GSNZ_ADDRESS_NOT_FOUND:
    case GSNZ_ADDRESS_NOT_RESOLVED:
    case GSNZ_LASTLINE_NOT_FOUND:
        default: /* or just the default */
        fprintf( stderr, "Error geocoding address: %d.\n", retcode );
    }
```

This approach does not include any “advanced” features such as query or multiple match resolution. In reality, however, the burden of development is mostly in creating the user interface; query or multiple match logic can be implemented quickly and easily.

Analyzing a Match

`GsNZDataGet` can return a status code that explains the match results. The match ratings are described in "Status Codes" on page 59. This code will define exactly what elements of an input address were modified, or the reason a match was not made.

Using GeoStan New Zealand in Windows

In order to successfully use the 32-bit version of GeoStan New Zealand, your project must define the symbol:
`_WIN32`

Please note that you will not be warned with compile or link errors if this symbol is undefined. However, your application will not function properly. The most common error caused by not defining the symbol is to run out of stack space.

Microsoft compilers typically define this symbol for you, but we strongly recommend that you define it in your project for completeness.

GSNZ.H

GSNZ.H defines the library API, and contains all symbolic constants used and/or returned by the GeoStan New Zealand functions. This file is extensively commented; we recommended that you review this file before using the GeoStan New Zealand library.

Using GeoStan New Zealand with C and C++

GeoStan New Zealand's native development platform is Microsoft Visual C and C++. While the library is developed internally in C++, the API is a "C-wrapper" that allows the developer to work in C or C++.

The test programs provided on the CD have batch files to build programs with Microsoft Visual C. They are robust test programs that show most of the functions available within GeoStan New Zealand.

Also included is gnzcoder.c, which demonstrates batch processing functionality.

The samples can be freely copied and included in your own code.

GSNZ.H is the proper include file for most C environments.

Using GeoStan New Zealand with Visual Basic

GeoStan New Zealand applications may be created using the Visual Basic development platform. When creating a Visual Basic application, the GeoStnNZ.bas file must be included in the project, but may require modification if used with other BASIC environments. As with using any DLL in Visual Basic, the GeoStan New Zealand DLL file (gsnz.dll) must be in the Windows directory, and not the program directory.

GeoStan New Zealand also includes a Visual Basic test program that may be helpful in application development.

Chapter 4

Function Reference

This chapter describes in detail each of the public functions available through the GeoStan New Zealand API. The format for the syntax of each function is:

```
ReturnType FunctionName ( ArgumentType identifier,
   ArgumentType identifier )
```

where:

- `ReturnType` is the data type of return value associated with the function.
- **`FunctionName`** is the name of the function.
- `ArgumentType` is the type of the argument that follows.
- `identifer` is the descriptive variable name of the argument.

Data types

`intlu`	32-bit unsigned long integer
`intl`	32-bit long integer
`intsu`	16-bitt unsigned short integer
`ints`	16-bit short integer
`char *`	pointer to null-terminated string
`gsNZ_const_str`	pointer to constant string
`pstr`	pointer to null terminated string
`intlu *`	pointer to unsigned long integer
`intl *`	pointer to long integer
`pintl`	pointer to long integer
`ints *`	pointer to short integer
`GsNZId`	32-bit unsigned long integer
`GsNZEnum`	16-bit unsigned short integer
`GsNZFunStat`	integer
`GsNZInitStruct`	Structure's members pass initialization information to the library; structure used when calling GsNZInit
`GsNZStreetHandle`	Structure contains a string street handle
`GsNZSegmentHandle`	Structure contains a street handle member
`GsNZRangeHandle`	Structure contains a segment handle member

All data types designated as “short” are 16-bit; “long” integers are 32-bit.

Quick Reference

Note: All examples in the function reference are written in C.

Initialization and Termination Functions

GsNZInit

Checks the license file and key, sets the cache size to be used by GeoStan New Zealand, sets the oldest acceptable date for GeoStan New Zealand data files, and initializes GeoStan New Zealand in one call.

GsNZTerm

Terminates GeoStan New Zealand. No functions may be called following **GsNZTerm**.

GsNZGetLibVersion

Returns the version of the GeoStan New Zealand library in use.

GsNZSetMixedCase

Sets whether information is returned in mixed or upper case.

Data Passing Functions

GsNZClear

Clears the data buffer in GeoStan New Zealand's internal data structures.

GsNZDataSet

Passes data for all address elements to GeoStan New Zealand.

GsNZDataGet

Returns data for all address and matched elements from GeoStan New Zealand.

GsNZErrorGet

Retrieves current error information.

GsNZGetCoords

Retrieves shape point coordinates for the street segment found via **GsNZFind**.

GsNZHandleGet

Retrieves data for objects found via **GsNZFindFirst** and **GsNZFindNext**.

GsNZHandleGetCoords

Retrieves the shape point coordinates for the current range handle.

Find/Query Functions

GsNZFind

Tells GeoStan New Zealand to match the current address loaded via **GsNZDataSet**.

GsNZFindFirst__

Finds first street, segment or range object to meet search criteria.

GsNZFindNext__

Finds next street, segment or range object to meet search criteria.

GsNZSoundex

Generates a soundex key for use in **GsNZFindFirst**.

`GsNZSetSelectionRange`

Allows a record found outside of **`GsNZFind`** to be used as a match.

`GsNZTestRange`

Determines whether a house number falls within a range.

Multiple Match Resolution Functions

`GsNZSetSelection`

Allows you to choose which multiple match is the correct match.

`GsNZNumMultiple`

Returns the number of multiple matches found.

`GsNZMultipleGet`

Returns the address elements for the multiple match item specified.

`GsNZMultipleGetHandle`

Returns the range handle for the multiple match item specified.

GsNZClear

CLEARS THE DATA BUFFER IN GEOSTAN NEW ZEALAND'S INTERNAL DATA STRUCTURES.

Syntax

```
GsNZFunStat GsNZClear( GsNZId gs );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

Return value

```
GSNZ_SUCCESS, GSNZ_ERROR.
```

Notes

GsNZClear sets the internal data geocode buffers to NULL values. Call **GsNZClear** before loading new data into the buffer to ensure that the buffer is reset. This call clears only address elements and locational information about the previously processed address and does not affect distance or centroid match type.

Example

See the code example following "Basic Matching" on page 11 for a complete example.

GsNZDataGet

RETURNS DATA FOR ALL ADDRESS AND MATCHED ELEMENTS FROM GEOSTAN NEW ZEALAND.

Syntax

```
GsNZFunStat GsNZDataGet( GsNZId gs, GsNZEnum fInOut, GsNZEnum
   fSwitch, pstr pbuffer, intsu bufLen );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

fInOut

Flag indicating whether to retrieve original (input) or matched (output) data. Valid enums are are described below. *Input.*

GSNZ_INPUT	Retrieve original data
GSNZ_OUTPUT	Retrieve matched data
GSNZ_INPUT_OUTPUT	Retrieve matched data if available, else get original data

fSwitch

The symbolic constant for the data item to retrieve. Valid GsNZEnums are listed in "Common Enums for Storing and Retrieving Data" on page 55. *Input.*

pbuffer

The location to store the returned data. *Output.*

bufLen

The maximum size of data that GeoStan New Zealand should return. The recommended buffer sizes for each item are defined as constants in GSNZ.H and are listed below in Table 1.

These sizes are the lengths required to get the full output string and include the null terminator. You may prefer to allocate a buffer that is smaller or larger than these numbers. However, if bufLen is shorter than the returned data, the data is truncated and no error is generated. *Input.*

Return value

GSNZ_SUCCESS, GSNZ_ERROR.

Notes

Table 1 lists the enums for the recommended buffer sizes of data items that GsNZDataGet can retrieve. These enums correspond to the data item enums passed with the fSwitch parameter.

Table 1: Symbolic constants for GsNZDataGet buffer lengths.

Constant	Size	Constant	Size
GSNZ_ADDRLINE_LENGTH	61	GSNZ_NAME2_LENGTH	41
GSNZ_CITY_LENGTH	29	GSNZ_POST_CODE_LENGTH	6
GSNZ_COORDX_LENGTH	9	GSNZ_POSTDIR_LENGTH	15
GSNZ_COORDY_LENGTH	9	GSNZ_PREDIR_LENGTH	15
GSNZ_DIST_LENGTH	3	GSNZ_PREDIR2_LENGTH	15
GSNZ_FIRM_NAME_LENGTH	41	GSNZ_ROAD_ID_LENGTH	30
GSNZ_HIRANGE_LENGTH	12	GSNZ_RANGE_PARITY_LENGTH	12
GSNZ_HOUSE_NUMBER_LENGTH	12	GSNZ_ROAD_ID_LENGTH	30
GSNZ_INTERSECTION_LENGTH	2	GSNZ_SEGMENT_DIRECTION_LENGTH	2
GSNZ_LASTLINE_LENGTH	61	GSNZ_SEGMENT_PARITY_LENGTH	12
GSNZ_LOC_CODE_LENGTH	5	GSNZ_STREET_SIDE_LENGTH	2
GSNZ_LOCATION1_LENGTH	41	GSNZ_SUBURB_NAME_LENGTH	31
GSNZ_LOCATION2_LENGTH	41	GSNZ_TYPE_LENGTH	15
GSNZ_LORANGE_LENGTH	12	GSNZ_TYPE2_LENGTH	15
GSNZ_MAIL_STOP_LENGTH	61	GSNZ_UNIQUE_ID_LENGTH	30
GSNZ_MATCH_CODE_LENGTH	5	GSNZ_UNIT_NUMBER_LENGTH	12
GSNZ_MESH_CODE_LENGTH	16	GSNZ_UNIT_TYPE_LENGTH	5
GSNZ_NAME_LENGTH	41		

When the *fInOut* parameter is set to GSNZ_INPUT, data is returned capitalized. When the parameter is set to GSNZ_OUTPUT, the following address elements:

- GSNZ_FIRM_NAME
- GSNZ_ADDRLINE
- GSNZ_SUBURB_NAME
- GSNZ_LASTLINE
- GSNZ_CITY
- GSNZ_LOCATION1
- GSNZ_LOCATION2

are returned in either mixed or upper case, as set by **GsNZSetMixedCase** (mixed case is the default setting). For GSNZ_OUTPUT, all other address elements are always returned in upper case.

When calling **GsNZDataGet** with one of the following symbolic constants as the *fSwitch* parameter:

- GSNZ_TYPE

- `GSNZ_TYPE2`
- `GSNZ_PREDIR`
- `GSNZ_PREDIR2`
- `GSNZ_POSTDIR`
- `GSNZ_POSTDIR2`

`GsNZDataGet` will return the whole word instead of an abbreviation (e.g. "Street" instead of "St").

See Also

`GsNZFind`

Example

See the code sample following "Basic Matching" on page 11 for a complete example.

GsNZDataSet

PASSES DATA FOR ALL ADDRESS ELEMENTS TO GEOSTAN NEW ZEALAND.

Syntax

```
GsNZFunStat GsNZDataSet( GsNZId gs, GsNZEnum fSwitch,
   gsNZ_const_str pBuffer );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

f*Switch*

The symbolic constant for the data item to load. Valid `GsNZEnums` are listed in "Common Enums for Storing and Retrieving Data" on page 55. *Input.*

pBuffer

String pointer containing the data to be loaded. *Input.*

Return value

`GSNZ_SUCCESS, GSNZ_ERROR.`

Notes

GsNZDataSet loads data elements into internal buffers. When address information is loaded as a complete address or last line, GeoStan New Zealand parses the data into its respective fields. For example, if you entered a last line of "Auckland 1730", GeoStan New Zealand parses the data and "sets" the city and Post Code fields. You can retrieve parsed data using **GsNZDataGet** and requesting the INPUT to these fields.

Note: Do *not* try to set discrete address fields using address line elements such as street name and number. GeoStan New Zealand's parser optimally matches whole address lines. Using discrete fields bypasses the parsing mechanism and defeats the optimizations.

See also

GsNZDataGet.

Example

See the code sample following "Basic Matching" on page 11 for a complete example.

GsNZErrorGet

RETRIEVES CURRENT ERROR INFORMATION.

Syntax

```
GsNZFunStat GsNZErrorGet( GsNZId gs, pstr pMessage, pstr
   pDetail );
```

Arguments

gs
: The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

p*Message*
: The basic explanation for the error; up to 256 bytes in length. *Output*.

p*Detail*
: The details of an error, such as filename; up to 256 bytes in length. *Output*.

Note: Both pMessage and pDetail can return up to 256 bytes of data. Always set buffers for these arguments to 256 bytes or larger.

Return value

Returns the error number of the most recent GeoStan New Zealand error.

`-1`	No error.
`0 through 99`	Indicates the actual DOS error values.
`100`	Unclassified error.
`101`	Unknown error.
`102`	Invalid file signature.
`103`	Table overflow.
`104`	Insufficient memory.
`105`	File not found.
`106`	Invalid argument to a function.
`107`	File is out of date.
`108`	Invalid license, bad license file name, or incorrect password

Notes

GsNZErrorGet returns the error number, the error message, and detailed information about the most recent GeoStan New Zealand error.

Example

See the code sample following "Basic Matching" on page 11 for a complete example.

GsNZFind

MATCHES THE CURRENT ADDRESS LOADED VIA **GSNZDATASET**.

Syntax

```
GsNZFunStat GsNZFind( GsNZId gs, GsNZEnum options );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

options

GsNZFind options. Valid enums are as follows. *Input*.

GSNZ_ADDR_CODE	Match the address and interpolate the location. **Required**.
GSNZ_WIDE_SEARCH	Use only the first letter when searching for streets. **Optional**.

Note: These option settings are additive. At minimum, you must add the GSNZ_ADDR_CODE enum.

Return value

- GSNZ_SUCCESS is returned if a match was found.
- GSNZ_ERROR is returned for low-level errors; use **GsNZErrorGet** to retrieve the error information.
- GSNZ_ADDRESS_NOT_RESOLVED is returned if GeoStan New Zealand cannot resolve which possible is a match.
- GSNZ_ADDRESS_NOT_FOUND is returned if an address match was not found.
- GSNZ_LASTLINE_NOT_FOUND is returned if a match was not found for city, suburb, or Post Code.

Notes

Even if an address cannot be standardized, you can retrieve normalized address information, match codes, or other elements with **GsNZDataGet**. To do this, you must call **GsNZMultipleGet** with an index of 0.

To successfully call **GsNZFind**, you must load the proper data elements with **GsNZDataSet**. In general, GeoStan New Zealand requires an address line element and a location specification for that address (suburb, city, Post Code, etc.). The location specification may be built from one or more different data elements. Which elements you pass to **GsNZDataSet** depend upon what elements are available for an address. The table below shows the various combinations of data elements that may be passed to **GsNZDataSet** to create a valid address.

Table 2: Combinations of Data Elements that May Be Passed to GsNZDataSet

Address Element	A	B	C	D	E	F	G	H	I	J
GSNZ_ADDRLINE	●	●	●	●	●	●	●	●	●	●
GSNZ_LASTLINE	●		●							
GSNZ_SUBURB_NAME		●	●		●		●		●	
GSNZ_CITY				●	●			●	●	
GSNZ_POST_CODE						●	●	●	●	

For example, the table above shows that a complete address could consist of an address line and last line (combination A); an address line, suburb name, and Post Code (combination G); or just an address line (combination J—if you are trying to find a street address or intersection that is unique throughout all of New Zealand).

See the section, "Address Range Values" on page 48 for more information on how GeoStan New Zealand matches a record in the data.

See also

`GsNZDataGet`, **`GsNZDataSet`**, **`GsNZMultipleGet`**.

Example

See the code sample following "Basic Matching" on page 11 for a complete example.

GsNZFindFirst___

FINDS FIRST STREET, SEGMENT OR RANGE OBJECT TO MEET SEARCH CRITERIA.

Syntax

```
GsNZFunStat GsNZFindFirstRange( GsNZId gs, GsNZRangeHandle
   *pRange );
GsNZFunStat GsNZFindFirstSegment( GsNZId gs,
   GsNZSegmentHandle *pSegment );
GsNZFunStat GsNZFindFirstStreet( GsNZId gs, GsNZStreetHandle
   *pStreet, GsNZEnum option, gsNZ_const_str reserved,
   gsNZ_const_str pName, gsNZ_const_str pNumber );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

**pRange*

A pointer to a segment/range handle. A valid handle is returned if a range was found. *Input, Output.*

**pSegment*

A pointer to a street/segment handle. A valid handle is returned if a segment was found. *Input, Output.*

**pStreet*

A pointer to a street handle. A valid handle is returned if a street was found. *Output.*

option

Valid enums are: *Input.*

GSNZ_NZ_SEARCH	Searches all of New Zealand for the street name specified in *pName*.
GSNZ_SDX_SEARCH	Searches all of New Zealand using the soundex value of the street name specified in *pName*.

reserved

Currently unused.

pName

This is the street name, or partial street name, to search for. If searching for a soundex key then this is a pointer to the numeric soundex key returned by **GsNZSoundex**. *Input.*

pNumber

The house number that must be contained within a range. *Input.*

Return value

- GSNZ_SUCCESS is returned if a match was found. If a match is found, the data for that match can be retrieved by using **GsNZHandleGet**.
- GSNZ_NOT_FOUND is returned if no match was found.

- `GSNZ_LASTLINE_NOT_FOUND` is returned if no match was found because an address last line was missing.
- `GSNZ_ERROR` is returned if an error occurs; use **`GsNZErrorGet`**.

Notes

`GsNZFindFirstStreet` finds the first street that meets the name criteria. **`GsNZFindFirstStreet`** also sets criteria for segment and range searches.

The *`pName`* argument limits the search to street names that begin with the name string. For example, if *`pName`* is set to "APPLE", then only streets beginning with Apple will be returned, such as Apple or Appleton. If the *`pName`* string is left blank, all street segments are returned. If the `GSNZ_SDX_SEARCH` switch is used in *`option`*, then the name pointer should actually be a pointer to a numeric soundex key as returned by **`GsNZSoundex`**.

The *`pNumber`* argument will return only those ranges that contain that house number. This can be an alphanumeric house number, such as 12c.

`GsNZFindFirst___` and **`GsNZFindNext___`** functions work together to allow you to access to the entire address directory database.

Note: Refer to "Extracting Street Data" on page 47 for details on using all of the **`GsNZFindFirst___`** and **`GsNZFindNext___`** functions.

See Also

`GsNZHandleGet`

Example

See the code sample associated with "Using the FindFirst, FindNext Functions" on page 49 for a complete example.

GsNZFindNext___

FINDS NEXT STREET, SEGMENT OR RANGE OBJECT TO MEET SEARCH CRITERIA.

Syntax

```
GsNZFunStat GsNZFindNextRange( GsNZId gs, GsNZRangeHandle
   *pRange );
GsNZFunStat GsNZFindNextSegment( GsNZId gs, GsNZSegmentHandle
   *pSegment );
GsNZFunStat GsNZFindNextStreet( GsNZId gs, GsNZStreetHandle
   *pStreet );
```

Arguments

gs
The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

**pRange*
A pointer to a segment/range handle. A valid handle is returned to the next range object, if there is one. *Input, Output*.

**pSegment*
A pointer to a street/segment handle. A valid handle is returned to the next segment object, if there is one. *Input, Output*.

**pStreet*
A pointer to a street handle. A valid handle is returned to the next street object, if there is one. *Input, Output*.

Return value

- GSNZ_SUCCESS is returned if a match was found.
- GSNZ_NOT_FOUND is returned if no match was found.
- GSNZ_ERROR is returned if an error occurs; use **GsNZErrorGet**.

Notes

This function continues to find objects that match the criteria specified in **GsNZFindFirst___**.

If a matching segment is found, the data for that segment can be retrieved by using **GsNZHandleGet**.

Note: Refer to "Extracting Street Data" on page 47 for details on using all of the **GsNZFindFirst___** and **GsNZFindNext___** functions.

See also

GsNZFindFirst.

Example

See the code sample associated with "Using the FindFirst, FindNext Functions" on page 49 for a complete example.

GsNZGetCoords

RETRIEVES SHAPE POINT COORDINATES FOR THE STREET SEGMENT FOUND VIA **GSNZFIND**.

Syntax

```
ints GsNZGetCoords( GsNZId gsnz, pintl pCoords, intsu max-
   Points );
```

Arguments

gsnz

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

pCoords

The array of coordinates, in COORDX, COORDY order. *Output.*

maxPoints

The maximum number of points that **GsNZGetCoords** should return; used to prevent writing past the end of the *pCoords* buffer. *Input.*

Return value

Number of points assigned to buffer.

Notes

This routine returns an array of coordinates for the current feature found via **GsNZFind**. The maximum number that GeoStan New Zealand can return is 64 coordinate pairs, each pair consisting of two long integers.

Note that a coordinate is stored as a 7-digit whole number, and is returned interpolated to the 100th of a unit. Thus, a coordinate of (1111111) would be returned as (1111111.11). This is a different scale from that expected by similar GIS applications, which typically express coordinates in millionths of a unit. You may need to scale coordinates obtained with this function before they are used as input to other software libraries or applications.

Example

```
/* Retrieves coordinates for a street segment on
   which there was an address geocode found via GsNZFind and
   prints the coordinate pairs */

#define        MAX_POINTS   50

ints          NumCoords;
intl          Coords[MAX_POINTS * 2];  /* Two numbers per
                                          coordinate */

NumCoords = GsNZGetCoords( gs, Coords, MAX_POINTS );

for ( int i = 0; i < NumCoords * 2; i += 2)
   printf( "CoordX = %ld, CoordY = %ld\n", Coords[i],
   Coords[i+1] );
```

GsNZGetLibVersion

RETURNS THE VERSION OF THE GEOSTAN NEW ZEALAND LIBRARY IN USE.

Syntax

```
GsNZFunStat GsNZGetLibVersion( );
```

Arguments

None.

Return value

- Low Byte = Major Version number.
- High Byte = Minor Version number.

Notes

This call returns the library version and can be made at any time.

In general, the major version number changes whenever new API features are added, or when the data structures in the GeoStan New Zealand data files change.

The minor version number changes for each release of GeoStan New Zealand.

GsNZHandleGet

RETRIEVES DATA FOR OBJECTS FOUND VIA **GSNZFINDFIRST** AND **GSNZFINDNEXT**.

Syntax

```
GsNZFunStat GsNZHandleGet( GsNZId gs, GsNZEnum fSwitch,
   GsNZRangeHandle *pRange, pstr pBuffer, intsu bufLen );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

fSwitch

The enum for the argument you wish to retrieve. Valid GsNZEnums are in "Common Enums for Storing and Retrieving Data" on page 55. *Input*.

**pRange*

A pointer to the current range handle. *Input*.

pBuffer

The location to store the returned data. *Output*.

bufLen

The maximum size of data that GeoStan New Zealand should return. If *bufLen* is shorter than the data returned by GeoStan New Zealand, the library truncates the data. No error is generated. Maximum sizes of returned elements can be found in "Common Enums for Storing and Retrieving Data" on page 55. *Input*.

Return value

GSNZ_SUCCESS, GSNZ_ERROR.

Notes

This function retrieves data from the geocode buffer for a given range handle. If you have a street or segment handle, you must convert the handle to a range handle before you can use this function.

Note: Refer to "Extracting Street Data" on page 47 for details on using all of the **GsNZFindFirst___** and **GsNZFindNext___** functions.

GsNZHandleGet returns all data in upper case.

See also

GsNZMultipleGet.

Example

See the code sample associated with "Using the FindFirst, FindNext Functions" on page 49 for a complete example.

GsNZHandleGetCoords

RETRIEVES THE COORDINATES FOR THE CURRENT RANGE HANDLE.

Syntax

```
ints GsNZHandleGetCoords( GsNZId gsnz, GsNZSegmentHandle
   *pHandle, pintl pCoords, intsu maxPoints );
```

Arguments

gsnz

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

pHandle

The range handle. *Input*.

pCoords

The array of coordinates in COORDX, COORDY order. *Output*.

maxPoints

The maximum number of points that **GsNZHandleGetCoords** should return; used to prevent writing past the end of the *pCoords* buffer. *Input*.

Return value

Number of points assigned to buffer.

Notes

This function returns an array of coordinates for the segment that *pHandle* points to.

Example

```
GsNZFunStat retcode;         /* return code */
intsu       maxPoints = 64; /* max # points GsNZHandleGetCoords returns */
intl        coords[64 * 2]; /* array of points */
ints        points;         /* # points returned by GsNZHandleGetCoords */

retcode = GsNZFindFirstStreet( gsnz, &hStreet, srchopt, pc, street, HouseNumber
    );

if ( retcode == GSNZ_SUCCESS )
{
    hSegment.hStreet = hStreet;
    hRange.hSegment = hSegment;
    retcode = GsNZFindFirstSegment( gsnz, &hSegment );
    if ( retcode == GSNZ_SUCCESS )
    {
        hRange.hSegment = hSegment;
        points = GsNZHandleGetCoords( gsnz, &hSegment, coords, maxPoints );
        printf("segment coordinates: \n");
        for (i = 0; i < points*2; i+=2 )
        {
            printf( "x=%-07d y=%-07d\n", coords[i],coords[i + 1]);
        }
    }
}
```

GsNZInit

CHECKS THE LICENSE FILE AND KEY, SETS THE CACHE SIZE TO BE USED, SETS THE OLDEST ACCEPTABLE DATE FOR DATA FILES, AND INITIALIZES GEOSTAN NEW ZEALAND IN ONE CALL.

Syntax

```
GsNZId GsNZInit ( GsNZInitStruct *gis );
```

Arguments

gis

Pointer to the `GsNZInitStruct` structure. This structure (defined in gsnz.h) contains the following members:

Member	Input/ Output	Explanation
version (optional)	Input	Set equal to GeoStan New Zealand software version.
options	Input	Specifies which components of GeoStan New Zealand to initialize. Valid bit settings include: GSNZ_FILE_ADDR_CODE: Loads all files necessary for address geocoding and standardization.
paths	Input	A list of paths to search for necessary files; directories are delimited by “;”.
reserved	N/A	Currently unused.
licFilePassword	Input	Password for the license file.
licFileName	Input	Path and name of the license file, for example, “c:\license\geostan_nz.lic”.
cacheSize	Input	Relative cache size to be used by GeoStan New Zealand. Valid enums are GSNZ_CACHESIZE_SMALL, GSNZ_CACHESIZE_MEDIUM, and GSNZ_CACHESIZE_LARGE.
relDate (optional)	Input	String [GSNZ_MAX_STR_LEN] that specifies the latest data to use in initialization. String format is YYYYMMDD.
NZAddressDate (optional)	Input	Build date of the current New Zealand address data.
status	Output	A pointer to a long (32-bit) integer to receive specifics of which components were successfully initialized. The following constants are used to test each significant bit: GSNZ_FILE_CITY_DIR: CTYNZ.DIR, city file is loaded GSNZ_FILE_GEO_DIR: NZ.GSD and/or NZT.GSD is loaded GSNZ_FILE_LICENSE: license file is loaded GSNZ_FILE_LOCALE: location lookup file is loaded (LOCALE.DBF) GSNZ_FILE_MESHBLOCK_DIR: meshblock lookup file is loaded (MESHBLK.GSB) GSNZ_FILE_PARSE_TABLES: the parsing tables are loaded (PARSENZ.DIR) GSNZ_FILE_RESULT: result lookup file is loaded (RESULT.DBF)

Return value

A valid `GsNZId` is returned if the system was initialized correctly. If required files are missing, GeoStan New Zealand does not initialize and **`GsNZInit`** returns `NULL`. The `status` member of the `GsNZInitStruct` structure indicates which files were successfully found (and by omission, which files were not found).

`GsNZInit` can fail for one of the following reasons:

- The necessary files were not found. Check the `status` structure member to verify this.
- The license file was not found, an invalid license file was found, or an incorrect password was given.
- There is not enough memory for GeoStan New Zealand to initialize.

Notes

The first call to GsNZInit will take longer to execute because it is building index files for the .dbf files.

See Also

`GsNZTerm`

Example

See the code sample following "Basic Matching" on page 11 for a complete example.

GsNZMultipleGet

RETURNS THE ADDRESS ELEMENTS FOR THE MULTIPLE MATCH ITEM SPECIFIED.

Syntax

```
GsNZFunStat GsNZMultipleGet( GsNZId gs, GsNZEnum fSwitch,
   ints index, pstr pBuffer, intsu bufLen );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

fSwitch

The enum for the argument you want to retrieve. Valid GsNZEnums are listed in "Common Enums for Storing and Retrieving Data" on page 55. *Input*.

index

The entry number (0-based) of the possible match whose data is to be retrieved. *Input*.

pBuffer

The location to store the returned data. *Output*.

bufLen

The maximum size of data that GeoStan New Zealand should return. If *bufLen* is shorter than the data returned by GeoStan New Zealand, the library truncates the data and no error is generated. For the maximum sizes of returned elements, see "Common Enums for Storing and Retrieving Data" on page 55. *Input*.

Return value

`GSNZ_SUCCESS, GSNZ_ERROR.`

Notes

This function retrieves data from the GeoStan New Zealand buffer for multiple matches. A multiple match is indicated as the **GsNZFind** `GSNZ_ADDRESS_NOT_RESOLVED` return code. It is important to first test for an intersection match, since the enums are different for retrieving intersection and non-intersection matches. See "Common Enums for Storing and Retrieving Data" on page 55 for valid enums for intersection and non-intersection matches, and the code in the example that follows.

GsNZMultipleGet returns all data in upper case.

See also

GsNZFind, GsNZSetSelection, GsNZNumMultiple.

Example

See the code sample for "GsNZNumMultiple" on page 38.

GsNZMultipleGetHandle

RETURNS THE RANGE HANDLE FOR THE MULTIPLE MATCH ITEM SPECIFIED.

Syntax

```
GsNZFunStat GsNZMultipleGetHandle( GsNZId gs, ints index,
   GsNZRangeHandle *pHandle );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

index

The entry number (0-based) of the possible match whose data is to be retrieved. *Input*.

pHandle

The range handle for the possible match entry that is returned by this function. *Output*.

Return value

GSNZ_SUCCESS, GSNZ_ERROR.

Notes

This function is used to extract more information about a possible match (by using the possible match's range handle) than is returned by **GsNZMultipleGet**. It retrieves the range handle for the possible match indicated by the entry argument. Once you have the correct range handle, you can retrieve information by using **GsNZHandleGet**. For a list of elements returned by **GsNZMultipleGet** or **GsNZHandleGet**, see "Common Enums for Storing and Retrieving Data" on page 55.

See also

GsNZFind, GsNZHandleGet, GsNZMultipleGet.

GsNZNumMultiple

RETURNS THE NUMBER OF MULTIPLE MATCHES FOUND.

Syntax

`GsNZFunStat` **`GsNZNumMultiple`**`( GsNZId` *`gs`* `);`

Arguments

gs

The ID returned by **`GsNZInit`** for the current instance of GeoStan New Zealand. *Input.*

Return value

A positive value indicates the number of multiple matches. `GSNZ_ERROR` indicates an error.

Notes

This function returns the number of multiple matches found. Use **`GsNZMultipleGet`** to retrieve the actual address elements for each possible match. A multiple match is indicated as a return code from **`GsNZFind`**.

See also

`GsNZFind`, **`GsNZMultipleGet`**.

Example

```
/*
  This example prints information about possible matches.
  It assumes that GsNZFind returned GSNZ_ADDRESS_NOT_RESOLVED.
*/

char buffer[GSNZ_MAX_STR_LEN];
int  bInter;
ints selection;

printf( “Possibles:\n” );
/*
   test for multiple match and set bInter to non-zero if
   true...
*/

GsNZMultipleGet( gs, GSNZ_INTERSECTION, 0, buffer, sizeof(buffer) );
bInter = *buffer == 'T';

/*
   get the number of multiples and
   print them as we loop through each one.
*/

int n = GsNZNumMultiple( gs );
for ( int i = 0; i < n; ++i )
{
    printf( “%d: “, i );
    /* if we don’t have an intersection, print the ranges */
    if ( !bInter )
    {
        GsNZMultipleGet( gs, GSNZ_LORANGE, i, buffer, sizeof(buffer) );
```

```
            printf( “%s “, buffer );
            GsNZMultipleGet( gs, GSNZ_HIRANGE, i, buffer, sizeof(buffer) );
            printf( “%s “, buffer );
        }
        /*
           print the street name for both intersections and non
           intersections.
        */
        GsNZMultipleGet( gs, GSNZ_PREDIR, i, buffer, sizeof(buffer) );
        printf( “%s “, buffer );
        GsNZMultipleGet( gs, GSNZ_NAME, i, buffer, sizeof(buffer) );
        printf( “%s “, buffer );
        GsNZMultipleGet( gs, GSNZ_TYPE, i, buffer, sizeof(buffer) );
        printf( “%s “, buffer );
        GsNZMultipleGet( gs, GSNZ_POSTDIR, i, buffer, sizeof(buffer) );
        printf( “%s “, buffer );
        /*
           if an intersection match, print the second street name...
        */
        if ( bInter )
        {
            printf( “ & “ );
            GsNZMultipleGet( gs, GSNZ_PREDIR2, i, buffer, sizeof(buffer) );
            printf( “%s “, buffer );
            GsNZMultipleGet( gs, GSNZ_NAME2, i, buffer, sizeof(buffer) );
            printf( “%s “, buffer );
            GsNZMultipleGet( gs, GSNZ_TYPE2, i, buffer, sizeof(buffer) );
            printf( “%s “, buffer );
            GsNZMultipleGet( gs, GSNZ_POSTDIR2, i, buffer, sizeof(buffer) );
            printf( “%s “, buffer );
        }
        /*
           print each possible on a new line...
        */
        printf( “\n” );
    }

    do
    {
        printf( "Select one: " );
        gets( buffer );
        if ( !*buffer )
            break;
        selection = atoi( buffer );
    } while ( selection < 0 || selection >= n );

    GsNZSetSelection( gsnz, selection );

    /*
       Call a series of GsNZDataGet(GSNZ_OUTPUT)
       to get more specific data for the selected address.
    */
```

GsNZSetMixedCase

SETS WHETHER INFORMATION IS RETURNED IN MIXED OR UPPER CASE.

Syntax

`ints` **`GsNZSetMixedCase`**`( GsNZId` *`gs`*`, ints` *`bMixedCase`* `);`

Arguments

`gs`

The ID returned by **`GsNZInit`** for the current instance of GeoStan New Zealand. *Input.*

`bMixedCase`

0 = Upper case, 1 = mixed case (default). *Input*.

Return value

Returns the current setting for this option (that is, the setting before it was changed).

Notes

This function determines whether standardized data is returned in upper or mixed case. Specifically, it sets the case of the following matched address elements returned by **`GsNZDataGet`**(`GSNZ_OUTPUT`):

- `GSNZ_FIRM_NAME`
- `GSNZ_ADDRLINE`
- `GSNZ_SUBURB_NAME`
- `GSNZ_LASTLINE`
- `GSNZ_CITY`
- `GSNZ_LOCATION1`
- `GSNZ_LOCATION2`

All other output values for all other data retrieval functions and enum conbinations will be output in upper case.

If you specify mixed case, generally only the first letter is set to upper case. However, you can set the letter after “Mc” to upper case (for words beginning with “Mc”). You can also set to upper case all words that contain digits, except for ordinals (1st, 2nd, and so on.).

GsNZSetSelection

SELECTS A PARTICULAR MATCH FROM A SET OF MULTIPLE MATCHES.

Syntax

`GsNZFunStat` **`GsNZSetSelection`**`( GsNZId` *`gs`*`, ints` *`index`* `);`

Arguments

`gs`

The ID returned by **`GsNZInit`** for the current instance of GeoStan New Zealand. *Input.*

`index`

The index number (0-based) of the possible to return as match. *Input.*

Return value

`GSNZ_SUCCESS`, `GSNZ_ERROR` (if the selection is out of range).

Notes

After **`GsNZFind`** returns `GSNZ_ADDRESS_NOT_RESOLVED`, use **`GsNumMultiple`** and **`GsNZMultipleGet`** to determine which possible match is the correct match. Then use **`GsNZSetSelection`** to load that possible match into the data retrieval buffers and retrieve it using **`GsNZDataGet`**.

See also

`GsNZFind`, **`GsNZNumMultiple`**, **`GsNZMultipleGet`**.

Example

See the code example for "GsNZNumMultiple" on page 38.

GsNZSetSelectionRange

ALLOWS A RECORD FOUND OUTSIDE OF **GSNZFIND** TO BE USED AS A MATCH.

Syntax

`GsNZFunStat` **`GsNZSetSelectionRange`**`( GsNZId` *`gs`*`, GsNZRangeHandle` *`*pRange`*`, GsNZEnum` *`options`* `);`

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

pRange

The range pointer of object to return as a match. A range record indicates a range of addresses such as “280 - 296 Greenpark Drive”. *Input.*

options

GsNZSetSelectionRange options. Valid enums are as follows. *Input*.

GSNZ_ADDR_CODE	Match the address and interpolate the location.

Return value

GSNZ_SUCCESS, GSNZ_ERROR.

Notes

This function allows you to get a range handle which points to the address range you want to match to. Use this command to indicate to GeoStan New Zealand that **GsNZFind** found the match, when in fact the match is set to whichever range handle is passed through the **pRange* argument. You must specify the *options* enum so that subsequent calls to **GsNZDataGet** have the type of location information to return.

This function returns standardized results from a list generated by **GsNZFindFirst** / **GsNZFindNext**. This function is used to perform a “Query” function that lets the user choose from a list of possible matches.

Example

```
char              houseNumber[GSNZ_HOUSE_NUMBER_LENGTH];
char              street[GSNZ_NAME_LENGTH];
char              buffer[GSNZ_MAX_STR_LEN];
char              lo[GSNZ_LORANGE_LENGTH];
char              hi[GSNZ_HIRANGE_LENGTH];
GsNZFunStat       retcode;
GsNZStreetHandle  hStreet;
GsNZSegmentHandle hSegment;
GsNZRangeHandle   hRange;

printf( "Enter the house number: " );
gets( houseNumber );
printf( "Enter the street name: " );
gets( street );

do
{
    /* walk through all of the streets */
    retcode = GsNZFindFirstStreet( gsnz, &hStreet, GSNZ_NZ_SEARCH, 0, street,
        "" );
    while ( retcode == GSNZ_SUCCESS )
    {
        hSegment.hStreet = hStreet;

        /* walk through all of the segments */
        retcode = GsNZFindFirstSegment( gsnz, &hSegment );
        while ( retcode == GSNZ_SUCCESS )
        {

            /* walk through all of the ranges */
            hRange.hSegment  = hSegment;
            retcode = GsNZFindFirstRange( gsnz, &hRange );
            while ( retcode == GSNZ_SUCCESS )
            {

                retcode = GsNZSetSelectionRange( gsnz, &hRange, GSNZ_ADDR_CODE
                    );
                if ( retcode == GSNZ_SUCCESS )
                {
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_LORANGE, lo,
                                    GSNZ_LORANGE_LENGTH );
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_HIRANGE, hi,
                                    GSNZ_HIRANGE_LENGTH );
                    retcode = GsNZTestRange( gsnz, houseNumber, lo, hi );
                    if ( !retcode )
                        printf( "House number '%s' is NOT between %s and
                                    %s.\n\n",houseNumber, lo, hi );
                    else
                        printf( "House number '%s' is between %s and %s.\n\n",
                                houseNumber, lo, hi );

                    /* information about the selected range record */

                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_PREDIR,     buffer,
                                    sizeof(buffer) );
                    printf("Predir: %s  ", buffer );
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_NAME,       buffer,
                                    sizeof(buffer) );
                    printf("Street: %s  ", buffer );
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_TYPE,       buffer,
                                    sizeof(buffer) );
                    printf("Type: %s ", buffer );
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_POSTDIR,    buffer,
                                    sizeof(buffer) );
                    printf("Postdir: %s \n", buffer );
                     GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_SUBURB_NAME, buffer,
                                     sizeof(buffer) );
                    printf("Suburb: %s ", buffer );
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_CITY,       buffer,
                                    sizeof(buffer) );
                    printf("City: %s ", buffer );
                    GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_POST_CODE,  buffer,
                                    sizeof(buffer) );
                   printf("Post Code: %s \n\n\n", buffer );
                }

                retcode = GsNZFindNextRange( gsnz, &hRange );
            } /* end of ranges */

            retcode = GsNZFindNextSegment( gsnz, &hSegment );
        } /* end of segments */

        retcode = GsNZFindNextStreet( gsnz, &hStreet );

    }/* end of streets */

} while ( retcode == GSNZ_SUCCESS );
```

GsNZSoundex

Generates a soundex key for use in **GsNZFindFirst**

Syntax

```
intlu GsNZSoundex( pstr pName );
```

Arguments

pName

String to be converted to a soundex key. *Input*.

Return value

Returns the soundex key.

Notes

This function generates a soundex key for a street name. Use it in conjunction with **GsNZFindFirstStreet** when you want to perform a soundex search (that is, a match based on fields' pronunciation).

GeoStan New Zealand uses a modified version of the standard soundex algorithm first published by Donald Knuth. GeoStan New Zealand's modifications include special treatment of certain prefixes such as "MAC", "KN", and "WR", special treatment for numeric street names; and an encoding scheme to pack the key into the smallest number of bits.

See also

GsNZFindFirstStreet.

Example

See "Using the FindFirst, FindNext Functions" on page 49.

GsNZTerm

TERMINATES GEOSTAN NEW ZEALAND.

Syntax

```
GsNZFunStat GsNZTerm( GsNZId gs );
```

Arguments

gs

The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input*.

Return value

`GSNZ_SUCCESS, GSNZ_ERROR`.

Notes

You must call **GsNZTerm** at the conclusion of a session to close files and release other resources. It closes GeoStan New Zealand and invalidates `GsNZId`.

See also

GsNZInit.

GsNZTestRange

DETERMINES WHETHER A HOUSE NUMBER FALLS WITHIN A RANGE.

Syntax

```
GsNZFunStat GsNZTestRange( GsNZId gsnz, gsNZ_const_str num-
   ber, gsNZ_const_str rangeLo, gsNZ_const_str
   rangeHi );
```

Arguments

gsnz
The ID returned by **GsNZInit** for the current instance of GeoStan New Zealand. *Input.*

number
The house number to test. *Input.*

rangeLo
The low house number in range. *Input.*

rangeHi
The high house number in range. *Input.*

Return value

- Non-zero — number is within the range.
- 0 — number is not within the range or the comparison cannot be made.

Notes

GsNZTestRange tests to see if the house number is within the range defined by *rangeHi* and *rangeLo*. It handles all valid house number patterns, such as

123	123A	A123	1A23	A1B23
1A23B	1-23	AB		

Note: Some patterns are not comparable with other patterns; for example, 123 is not comparable to a range of 1A01 to 1A99.

Example

See the code sample for "GsNZSetSelectionRange" on page 42.

Chapter 5

Advanced Concepts

Extracting Street Data

All of the information contained within the GSD files is accessible via the functions in GeoStan New Zealand. While sophisticated matching algorithms are used to facilitate address standardization and geocoding, there may be other purposes that require the data to be accessed in a more direct method. The **`GsNZFindFirst__`**, **`GsNZFindNext__`**, and **`GsNZHandleGet`** functions allow just that. This section will detail their use. Please note that these are advanced features that are not normally used for most standardization and geocoding applications.

Internal Data Structure and Concepts

The data within the GSD files is organized in the following manner. At the highest level there is a Street Object. Street Objects contain information about the street as a whole, specifically its PreDir, Name, Type and PostDir. Each Street Object can contain one or more Segment Objects. A Segment Object relates to only one Street Object and contains the COORDX and COORDY values of the corners of the street segment, as well as the shape points, the Census Meshblock codes, and the maximum and minimum ranges for that segment.

Each Segment object can contain zero or more Range Objects. A Range Object relates to only one Segment Object, and therefore only one Street Object as well. A Range Object contains low and high house ranges (alphanumeric ranges are allowed).

Note: The low and high range values are 0 for segment records (GSNZ_SEG_LORANGE and GSNZ_SEG_HIRANGE) and range records (GSNZ_LORANGE and GSNZ_HIRANGE) that do not have house number data.

Address Range Values

In the New Zealand data, not all range records have address range information (a high and low value for either side of a street; see the enums GSNZ_LORANGE and GSNZ_HIRANGE in "Common Enums for Storing and Retrieving Data" on page 55). Even though there is addressable space for those records, house number data either does not exist or the house numbers are irregular. These range records have been assigned high and low values of 0.

A segment containing a 0-range record will not include 0 in the segment high and low values (enums GSNZ_SEG_LORANGE and GSNZ_SEG_HIRANGE) unless both sides of the street have 0-range records. For example, the left side of the street has house numbers 10 – 14, creating an even range record with GSNZ_LORANGE = 10 and GSNZ_HIRANGE = 14. The right side of the street has GSNZ_LORANGE = 0 and GSNZ_HIRANGE = 0. The segment high and low values will be GSNZ_SEG_LORANGE = 10 and GSNZ_SEG_HIRANGE = 14.

For each street in New Zealand that has one or more 0-range records, there is a set of -1 records (one segment and one range record with low and high values of -1).

Address matching with **`GsNZFind()`** is based upon the user's input address falling between a range record's low and high values. If the user does not enter a house number in the address (GSNZ_HOUSE_NUMBER or GSNZ_ADDRLINE) or the entered house number is not found in any of the range records for that street, **`GsNZFind`** will match to a -1-range record first and a 0-range record if a -1 record does not exist for that street. See "Address Location Codes" on page 62.

Range records that do not contain an addressable location (a bridge, a roundabout, or an underpass, for example) have been assigned segment and range low and high values of "9999". **`GsNZFind`** will not match to one these records.

Filtering

The **`GsNZFindFirstStreet`** function allows you to set a filter that determines the objects that are "found". Range Objects are filtered using all the criteria given in **`GsNZFindFirstStreet`**.

Filtering on a house number requires a Range or Segment Object to contain that house number before it is returned as a match. Street and Segment Objects may be returned that have no Segment or Range Objects that contain the house number. For this reason, it is important to check for Range Objects that meet the filter criteria. Any valid house numbers can be given.

Street, Segment, and Range Handles

Because of the hierarchical nature of the GSD data, we can create handles that are attached to any of the Street, Segment and Range Object types, and that contain the reference to higher objects in the order. For example, a range handle also contains a segment handle, which in turn contains a street handle. A street handle, however, does not contain a segment or range handle. A handle can be promoted to either predecessor. For example,

```
hStreet = hSegment.hStreet;
hStreet = hRange.hSegment.hStreet;
```

are acceptable, but

```
hSegment = hStreet;
```

is undefined. However, the statement

```
hSegment.hStreet = hMyStreet;
```

is valid and is used to promote a street handle to a segment handle for entry into the Find routines, as shown in the next section. Please note that the segment handle of `hSegment` is still undefined, but that `hSegment.hStreet` can be used as a valid street handle (assuming `hMyStreet` was a valid handle to begin with).

Using the FindFirst, FindNext Functions

In order to search the GSD files, you must begin with a call to **`GsNZFindFirstStreet`**. This function sets the street name and filtering options, as well as returning a street handle for the first street. The next call is normally to **`GsNZFindFirstSegment`**, which takes as an argument the street handle received from **`GsNZFindFirstStreet`** promoted to a segment handle, as shown in the previous section. Often, a series of three nested "while" statements are used to traverse all possible entries.

```
GsNZFunStat     iStat;
intl     coords[1000];
int      i;
GsNZEnum option;
intsu    points = 0,
         maxpoints = 500;
char     buffer[GSNZ_MAX_STR_LEN];
char     street[GSNZ_MAX_STR_LEN];
char     houseNumber[GSNZ_MAX_STR_LEN] = {0};
pintlu   sdx = street;
```

```
GsNZStreetHandle  hStreet;
GsNZSegmentHandle hSegment;
GsNZRangeHandle   hRange;

while( 1 )
{

    /* Get what type of search to execute */
    printf( "\t1. Search for the street name \n");
    printf( "\t2. Search using the street soundex \n" );
    printf( "\tENTER: exit\n" );
    printf( "(1 or 2): " );
    fflush( stdin );
    gets( street );
    option = 0;
    switch ( *street )
    {
        case '1':
            option = GSNZ_NZ_SEARCH;
            break;

        case '2':
            option = GSNZ_SDX_SEARCH;
            break;

        default:
            return;
    }

    /* get the street name */
    printf( "\nEnter a street name:             " );
    gets( street );
    if ( *street == 0 )
    {
        printf( "\nStreet name was blank.\nExiting.\n" );
        return;
    }

    /* if it's a soundex search, convert the street name to a soundex */
    if ( option == GSNZ_SDX_SEARCH )
    {
        *sdx = GsNZSoundex( street );
    }

    /* get the optional house number for a narrower search */
    printf( "Enter a house number (optional): " );
    gets( houseNumber );

    printf ("\n");

    iStat = GsNZFindFirstStreet( gsnz, &hStreet, option, 0, street,
   houseNumber );

    switch ( iStat )
    {
        case GSNZ_SUCCESS:
            break;

        case GSNZ_ERROR:
        {
            GsNZErrorGet( gsnz, buffer, street );
            printf( "Error: %s\n%s\n", buffer, street );
            break;
        }
        case GSNZ_NOT_FOUND:
        {
            printf( "Address Not Found.\n") ;
```

```
                GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_MATCH_CODE,  buffer,
sizeof(buffer) );
                printf( "Match code = %s\n", buffer );
                break;
            }
            case GSNZ_LASTLINE_NOT_FOUND:
            {
                printf( "Lastline Not Found.\n" );
                GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_MATCH_CODE,  buffer,
               sizeof(buffer) );
                printf( "Match code = %s\n", buffer );
                break;
            }

            default:
            {
                printf("Unknown return code: %ld\n", iStat );
                GsNZDataGet( gsnz, GSNZ_OUTPUT, GSNZ_MATCH_CODE,  buffer,
               sizeof(buffer) );
                printf( "Match code = %s\n", buffer );
                break;
            }
        }

        /* successfully found the street */

        while( iStat == GSNZ_SUCCESS )
        {
            /* print the street data */

            printf("Street handle:\n");
            hSegment.hStreet = hStreet;
            hRange.hSegment = hSegment;

            /* Enums typically associated with a street handle */

            GsNZHandleGet( gsnz, GSNZ_PREDIR, &hRange, buffer, sizeof(buffer));
            printf(" Predir: %s ", buffer );
            GsNZHandleGet( gsnz, GSNZ_NAME,    &hRange, buffer,
           sizeof(buffer));
            printf(" Street: %s ", buffer );
            GsNZHandleGet( gsnz, GSNZ_TYPE,    &hRange, buffer,
           sizeof(buffer));
            printf(" Type: %s ", buffer );
            GsNZHandleGet( gsnz, GSNZ_POSTDIR, &hRange, buffer,
           sizeof(buffer));
            printf(" Postdir: %s\n", buffer );

            /* print all of the segments for the street */

            iStat = GsNZFindFirstSegment( gsnz, &hSegment );
            while( iStat == GSNZ_SUCCESS )
            {
                hRange.hSegment = hSegment;
                printf("Segment handle: \n" );

                GsNZHandleGet( gsnz, GSNZ_SEG_LORANGE, &hRange, buffer,
               sizeof(buffer) );
                printf("  LoRange: %s  ", buffer );
                GsNZHandleGet( gsnz, GSNZ_SEG_HIRANGE, &hRange, buffer,
               sizeof(buffer) );
                printf("HiRange: %s  ", buffer );
                GsNZHandleGet( gsnz, GSNZ_SEGMENT_PARITY, &hRange, buffer,
               sizeof(buffer) );
                printf("SegParity: %s  ", buffer );
                GsNZHandleGet( gsnz, GSNZ_SEGMENT_DIRECTION, &hRange, buffer,
               sizeof(buffer) );
                printf("SegDir: %s \n", buffer );
```

```
 GsNZHandleGet( gsnz, GSNZ_ROAD_ID, &hRange, buffer,
sizeof(buffer) );
 printf("  Road ID: %s  ", buffer );
 GsNZHandleGet( gsnz, GSNZ_UNIQUE_ID, &hRange, buffer,
sizeof(buffer) );
 printf("Unique ID: %s \n", buffer );

 points = GsNZHandleGetCoords( gsnz, &hSegment, coords,
maxpoints );
 points *= 2;
 printf("  Segment Coordinates:  ");
 for (i = 0; i < points; i += 2 )
 {
     printf( "%d, %d  ", coords[i], coords[i+1] );
 }
 printf("\n");

 /* print all of the ranges for the segment */

 iStat = GsNZFindFirstRange( gsnz, &hRange );
 while( iStat == GSNZ_SUCCESS )
 {
     printf("Range handle: \n" );

     GsNZHandleGet( gsnz, GSNZ_LORANGE, &hRange, buffer,
   sizeof(buffer) );
     printf("  LoRange: %s  ", buffer );
     GsNZHandleGet( gsnz, GSNZ_HIRANGE, &hRange, buffer,
   sizeof(buffer) );
     printf("HiRange: %s  ", buffer );
     GsNZHandleGet( gsnz, GSNZ_RANGE_PARITY, &hRange, buffer,
   sizeof(buffer) );
     printf("RangePar: %s  ", buffer );
     GsNZHandleGet( gsnz, GSNZ_PREDIR, &hRange, buffer,
   sizeof(buffer) );
     printf("Predir: %s  ", buffer );
     GsNZHandleGet( gsnz, GSNZ_NAME, &hRange, buffer,
   sizeof(buffer) );
     printf("Street: %s  ", buffer );
     GsNZHandleGet( gsnz, GSNZ_TYPE, &hRange, buffer,
   sizeof(buffer) );
     printf("Type: %s ", buffer );
     GsNZHandleGet( gsnz, GSNZ_POSTDIR, &hRange, buffer,
   sizeof(buffer) );
     printf("Postdir: %s \n", buffer );
     GsNZHandleGet( gsnz, GSNZ_SUBURB_NAME,  &hRange, buffer,
   sizeof(buffer) );
     printf("  Suburb: %s ", buffer );
     GsNZHandleGet( gsnz, GSNZ_CITY, &hRange, buffer,
   sizeof(buffer) );
     printf("City: %s ", buffer );
     GsNZHandleGet( gsnz, GSNZ_POST_CODE, &hRange, buffer,
   sizeof(buffer) );
     printf("Post Code: %s \n", buffer );
     GsNZHandleGet( gsnz, GSNZ_ROAD_ID, &hRange, buffer,
   sizeof(buffer) );
     printf("  Road ID: %s ", buffer );
     GsNZHandleGet( gsnz, GSNZ_UNIQUE_ID, &hRange, buffer,
   sizeof(buffer) );
     printf(" Unique ID: %s \n", buffer );

     iStat = GsNZFindNextRange( gsnz, &hRange );

 }  /* printing range data */

 if ( iStat == GSNZ_ERROR )
 {
     GsNZErrorGet(gsnz, buffer, street);
```

```
                    printf("Error: %s\n%s\n", buffer, street);
                    break;
                }

                iStat = GsNZFindNextSegment( gsnz, &hSegment );

            }  /* printing segment data */

            if ( iStat == GSNZ_ERROR )
            {
                GsNZErrorGet(gsnz, buffer, street);
                printf("Error: %s\n%s\n", buffer, street);
                break;
            }

            iStat = GsNZFindNextStreet( gsnz, &hStreet );
            printf("\n");

        } /* printing street data */

    }

}
```

Appendix A

Common Enums for Storing and Retrieving Data

Enumerations

Table 3 on page 57 lists the valid enums used with **GsNZHandleGet**, **GsNZDataSet**, **GsNZDataGet** and **GsNZMultipleGet**. These tables should be complete, but check the Readme file for last minute additions. The contents of each table column are described below:

Enum

This column lists the defined constant used to access this element.

GsNZHandleGet

Street

A "●" in this column indicates that this enum is valid when using a Street handle with **GsNZHandleGet**. This will include only Street level information.

Segment

A "●" in this column indicates that this enum is valid when using a Segment handle with **GsNZHandleGet**. This will include Street and Segment information.

Range

A "●" in this column indicates that this enum is valid when using a Range handle with **GsNZHandleGet**. This will include Street, Segment and Range information.

GsNZDataSet

A "●" in this column indicates that this enum is valid when using **GsNZDataSet**.

GsNZDataGet

Input

A “●” in this column indicates that this enum is valid when using the `GSNZ_INPUT` flag with **`GsNZDataGet`**.

Output

A “●” in this column indicates that this enum is valid when using the `GSNZ_OUTPUT` flag with **`GsNZDataGet`**.

GsNZMultipleGet

For multiple matches, it is important to first test if the match was to a street intersection. Enums listed with an “A” are valid for non-intersection matches only. Enums listed with an “I” are valid for intersection matches only. Enums listed with a “●” are valid for either type of match.

Length

The number in this column indicates the largest string GeoStan New Zealand will return for this enum. Please note that this length includes a `NULL`-termination character.

Description

A brief explanation of the information returned. Additional information for several enums is given below:

`GSNZ_MESH_CODE` refers to the Census Meshblock ID.

If you use one of these functions with an invalid enum, the function will return a GSNZ_ERROR, and **`GsNZErrorGet`** will return an “Invalid Argument” error.

Note: The symbols shown in the table are taken from GSNZ.H for implementations using “C”. For languages that have a length restriction on symbols, different symbols may be used. See the notes regarding specific development systems for more The symbols shown in the table are taken from GSNZ.H for implementations using “C”. For languages that have a length restriction on symbols, different symbols may be used. See the notes regarding specific development systems for more information.

Table 3: Common enums for storing and retrieving data

Enum	GsNZHandleGet			GsNZDataSet	GsNZDataGet		GsNZMultipleGet	Length	Description
	Street	Segment	Range		Input	Output			
GSNZ_ADDRLINE				●	●	●		61	First address line
GSNZ_CITY			●	●	●	●	●	29	City name
GSNZ_COORDX						●		12	COORDX of located point. Value is in whole meters on the New Zealand Map Grid.
GSNZ_COORDY						●		11	COORDY of located point. Value is in whole meters on the New Zealand Map Grid.
GSNZ_DIST				●	●			3	Offset distance (default 15) in meters from centerline
GSNZ_FIRM_NAME			●	●	●	●	A	41	Name of firm
GSNZ_HIRANGE			●			●	A	12	House number at high end of range
GSNZ_HOUSE_NUMBER					●	●		12	House number of entered address
GSNZ_INTERSECTION					●	●	●	2	Flag: (T, F) - T indicates a cross street match was found
GSNZ_LASTLINE			●	●	●	●	●	61	Complete last line (Post Code not avail with GsNZMultipleGet)
GSNZ_LOC_CODE						●		5	Location code - quality of location
GSNZ_LOCATION1						●		41	The first input location, "spell-checked" against the data
GSNZ_LOCATION2						●		41	The second input location, "spell-checked" against the data
GSNZ_LORANGE			●			●	A	12	House number at low end of range
GSNZ_MATCH_CODE						●		5	Match code—How the address was matched
GSNZ_ADDRLINE				●	●	●		61	First address line

Table 3: Common enums for storing and retrieving data

Enum	GsNZHandleGet			GsNZDataSet	GsNZDataGet		GsNZMultipleGet	Length	Description
	Street	Segment	Range		Input	Output			
GSNZ_MESH_CODE						●		7	The meshblock is the smallest geographic area used by Statistics New Zealand in the collection and/or processing of data. It is a building block that is aggregated into larger areas such as area units, territorial authorities, regional councils, and electoral districts.
GSNZ_NAME	●	●	●		●	●	●	41	Street name
GSNZ_NAME2					●	●	I	41	Cross street name
GSNZ_POST_CODE			●	●	●	●	A	6	4-digit Post Code
GSNZ_POSTDIR	●	●	●		●	●	●	3	Postfix direction
GSNZ_POSTDIR2					●	●	I	3	Cross street Postfix direction
GSNZ_PREDIR	●	●	●		●	●	●	3	Prefix direction
GSNZ_PREDIR2					●	●	I	3	Cross street prefix direction
GSNZ_RANGE_PARITY			●				A	2	Flag: Range contains even, odd, or both (E, O, B)
GSNZ_ROAD_ID		●	●			●	A	42	ID for the road name
GSNZ_SCORE							●	12	GeoStan New Zealand matching score
GSNZ_SEG_HIRANGE		●	●				A	12	High range number in segment.
GSNZ_SEG_LORANGE		●	●				A	12	Low range number in segment
GSNZ_SEGMENT_ DIRECTION		●	●				A	2	Flag: (F, R) R indicates numbers on segment are reversed
GSNZ_SEGMENT_PARITY		●	●				A	2	Flag: Odd numbers on left or right side of street (L, R, B, U)
GSNZ_STREET_SIDE						●		1	The values are left (0), right (1), or either (2).
GSNZ_SUBURB_NAME			●		●	●	●	30	Name of suburb
GSNZ_TYPE	●	●	●		●	●	●	5	Street type (also called street suffix)
GSNZ_TYPE2					●	●	I	5	Cross street type
GSNZ_UNIQUE_ID		●	●			●	A	43	ID for the road segment
GSNZ_UNIT_NUMBER					●	●		12	Unit number
GSNZ_UNIT_TYPE			●		●	●	A	5	Unit type (APT, STE, etc.)

Appendix B

Status Codes

Address matching returns a variety of information through *GSNZ_MATCH_CODE* and *GSNZ_LOC_CODE* used with **GsNZDataGet**.

Match Codes

Match codes are returned via the *GSNZ_MATCH_CODE* enum. Match codes indicate the portions of the address that matched (or did not match) to the GeoStan New Zealand Directory file. The match code digits indicate what elements of an address were unmatched, in the case of a "no match," or which elements of an address were modified to find a match in the GeoStan New Zealand Directory.

Match Codes Returned by GSNZ_MATCH_CODE

Ennn	Indicates an error, or no match. This can occur when the address entered either simply did not exist in the GeoStan New Zealand Directory, or the address was badly malformed and could not be parsed correctly. The last two digits of an error code (below) indicate which parts of an address were unable to be matched to the GeoStan New Zealand Directory.
001	Low level error. Use **GsNZErrorGet** to query.
003	Incorrect GSD file signature or version ID.
010	No city, suburb, or Post Code found.

011	Input Post Code was not in the directory.
012	Input city was not in the directory.
013	Input city was not unique in the directory.
020	No matching streets found in directory.
021	No matching cross streets for an intersection match.
022	No matching ranges.
023	Match is unresolved.
025	Too many possibles.
030	Unknown error.
Thh	Indicates a match to a box, bag, or route. This is considered the best address match, since it was matched directly against the New Zealand data. See below for the interpretation of the hex digits.
Xhhh	Match found was for an intersection of two streets, e.g., “Clay St & Michigan Ave.” The first hex digit refers to the last line information, the second hex digit refers to the first street in the intersection, and the third hex digit refers to the second street in the intersection. To decode the hex digits, see the list below.
Z	No address was given, but the Post Code or city was verified as valid.

Return Codes for Match Codes–First Hex Digit

0	No change in last line.
1	Post Code was changed.
2	City was changed.
3	City and Post Code were changed.
4	Suburb was changed.
5	Suburb and Post Code were changed.
6	Suburb and City were changed.
7	Suburb, City, and Post Code were changed.

Return Codes for Match Codes–Second and Third Hex Digit

0	No change in address line.
1	Street type was changed.
2	Pre-directional was changed.
3	Street type and Pre-directional were changed.
4	Post-directional was changed.
5	Street type and Post-directional were changed.
6	Pre-directional and Post-directional were changed.
7	Street type, Pre-directional and Post-directional were changed.
8	Street name was changed.
9	Street name and Street type were changed.
A	Street name and Pre-directional were changed.
B	Street name, Street type and Pre-directional were changed.
C	Street name and Post-directional were changed.
D	Street name, Street type and Post-directional were changed.
E	Street name, Pre-directional and Post-directional were changed.
F	Street name, Street type, Pre-directional and Post-directional were changed.

Location Codes

Location codes are returned via the *`GSNZ_LOC_CODE`* enum. Location codes indicate the accuracy of the assigned geocode. There are two types of location codes and one error location code:

AS	Address level street match
AX	Intersection match
E	No location available

Address geocodes are simple to interpret because they indicate a geocode made directly to a street network segment (or two segments, in the case of an intersection).

Address Location Codes

Address location codes detail the known qualities about the geocode. An address location code has three characters. The first character is always an "A," indicating an address location. The second character is an "S," indicating a location on a street range. The third character is a digit, indicating other qualities about the location. Address codes are defined in detail as follows:

`ASn`	Indicates a house range address geocode. This is the most accurate geocode available. The digit at the end indicates the following:

`0`	Best location.
`8`	Matched to -1 or 0-range record. See "Address Range Values" on page 48 for more information.

Appendix C

Sample Applications

This appendix discusses the sample applications Gnzcoder and Gnztest. Both applications are included with GeoStan New Zealand.

Gnzcoder

The Gnzcoder utility program for 32-bit Windows platforms (95 or NT) standardizes and geocodes addresses in fixed-length text files as a batch process. Gnzcoder is distributed with full source code for two reasons. First, you can use it as a learning aid to assist you in developing your own applications using GeoStan New Zealand. Second, you can modify Gnzcoder to work with a certain file type, return different information, or perform other functions as needed.

Installing Gnzcoder

Gnzcoder is distributed as a Windows executable. For information on installing Gnzcoder, please refer to the installation notes that came with the CD package.

Using Gnzcoder

Gnzcoder takes four command line parameters. The command line parameters are shown below:

`gnzcoder.exe` *`formatFile inputFile outputFile /L nn`*

where:

formatFile

Contains format initialization data used in processing the input file, similar to a Windows .INI file.

inputFile

The path and name of the input file containing addresses to process. The file must be a fixed-length ASCII text file.

outputFile

The path and name of a new text file that will be created to hold the processed records. The output file normally has the same format as the input file, with the addition of fields at the end of each record.

/L nn

Only process *nn* records from the input file and then stop.

A sample format file (gnzcoder.fmt) is included with the library. It is extensively commented and can serve as a useful template for creating your own configuration file.

In the gnzcoder.fmt file, you will find:

- Lines that begin with open brackets ([ ) or semicolons (;) are comment lines
- Blank lines are ignored
- Quotation marks around strings are unacceptable

In the keyword lists below, keywords may be preceded by a character. (Do not include these characters in the actual format file.) Each character's meaning is described in Table 4.

Table 4: Gnzcoder keyword control characters

Symbol	Definition
*	Indicates that a keyword is required.
&	Indicates that you must provide data element fields to create a valid address. See Table 2 on page 26 for a table of these data element combinations.
#	Indicates a definition is required in either the format file or the source file.
@	Indicates an optional keyword that requires a filename definition as an additional optional keyword. See the corresponding keyword with (!).
!	Indicates a keyword required if a @ keyword is set to ON.

Control keywords

The control keywords take the following syntax:
`keyword = value`

Keywords are not case sensitive. Table 5 contains all possible control keywords that can be used in the setup file.

Table 5: Gnzcoder keywords and values

Keyword	Value	Explanation
`@audit`	`0`	No records are copied to an Audit file during processing. ***Default.***
	`1 - 100000`	Every Nth record (as specified by this parameter) is written to an audit output file as named by the parameter *`audFile`*. (*`audFile`* must be initialized in the .CFG file).
`!audFile`		Fully qualified path and filename for audit record output. (Required if *`audit > 0`*.)
`headerLines`	`0`	Number of header lines to skip from the input file. ***Default.***
`*inReclen`		Record length of input file. (Maximum line length to read if `inRecType=1`.)
`inRecType`	`0`	Records are not delimited.
	`1`	Records are terminated with cr/lf (DOS/ Windows). ***Default.***
`keepNoMatch`	`0`	Do not output records that do not match.
	`1`	Output all records. ***Default.***
`*licensePath`		Full name and path of the license file (e.g. "d:\license\gsnz.lic").
`mixedCase`	`0`	Return results are upper case.
	`1`	Return results are in mixed case. ***Default***.
`@nonStand`	`0`	No redirection of non-standardized records. ***Default.***
	`1`	Redirect all non-standardized records to a separate output file (*`nStdFile`*). (*`nStdFile`* must be initialized.) See "Non-standardized File" on page 68 for more details.
`!nStdFile`		Fully qualified path and filename for non-standardized record output. (Required if *`nonStand=1`*.)
`outHeader`	`0`	The header is not copied to the output file.
	`1`	Copy the header to the output file. ***Default***.
`*outReclen`		Record length of output file. Specifying an `outReclen` shorter than `inReclen` will truncate the input record to this length.
`outRecType`	`0`	Records are not delimited.

Keyword	Value	Explanation
	`1`	Records are terminated with cr/lf (DOS/ Windows). ***Default.***
`#password`		GeoStan New Zealand password to enable operation (provided by Critchlow Associates).
`*path`		Search path for the GeoStan New Zealand data file (delimited with semicolons, e.g. “c:\;d:\geocode”).
`#reportFile`		Fully qualified path and file name for statistics report file. If this keyword is not present, Gnzcoder writes a report file “gnzcoder.rpt” to the current working directory.

Input & output keywords

These keywords specify the format of each address in the input and output files. Input fields are in the input and output files. The output fields appear only in the output file.

All input and output fields have the following syntax:

`keyword=`*`start_pos`*`,`*`length`*

`start_pos`
The offset for the first character of the field.

`length`
The actual length of the field.

For example, `inFirm=10,50` indicates that the Firm field starts at offset 10 and is 50 characters wide.

Table 6: Gnzcoder input fields

Field	Explanation
`*inAddress`	Address line
`inFirm`	Firm name
`&inLastline`	Combined city and Post Code
`&inSuburb`	Suburb name
`&inPostCode`	Post Code
`&inCity`	City name

Table 7: Gnzcoder output fields

Field	Explanation
`outAddress`	Address line
`outAddr2`	Second address line
`outFirm`	Firm name
`outHouseNumber`	House number of entered address
`outIntersect`	’T’ if there is an intersection match; ’F’ if not
`outLastLine`	Complete last line (city, Post Code)
`outLocation1`	The first input location, "spell-checked" against the data
`outLocation2`	The second input location, "spell-checked" against the data
`outLocCode`	Location code; see “Location Codes” on page 62.
`outMatchCode`	Match code; see “Match Codes” on page 59.
`outMeshBlock`	Meshblock
`outName`	Primary street name
`outName2`	Street name of 2nd street if there is an intersection match
`outPostCode`	Post Code
`outPostDir`	Street postfix directional
`outPostDir2`	Postfix directional of 2nd street if there is an intersection match
`outPreDir`	Street predirectional
`outPreDir2`	Predirectional of 2nd street if there is an intersection match
`outSuburb`	Suburb
`outType`	Street type (also called street suffix)
`outType2`	Street type of 2nd street if there is an intersection match
`outUnitNumber`	Unit number
`outUnitType`	Unit type (designator)
`outX`	CoordX in New Zealand Map Grid
`outY`	CoordY in New Zealand Map Grid

Note: Gnzcoder cannot perform any type of data checking. That is, the data positions and lengths you indicate in the file are used as-is. If these lengths are incorrect, the resulting output file will not contain the correct data and may be completely unusable. Before running large jobs, we recommend that you test the setup file on a small subset of records from the larger file.

If your input file does not contain the fields necessary to store the output data, you can add fields to the output record by simply specifying a larger `outReclen` size and adding the necessary output fields to the end of the record structure. For example, if your input record length was 200, you could set the output record length to 240 and have `outAddress = 201, 40`. You may add any number of fields in this manner.

Though we do not recommend it, you can overwrite existing fields. If, for example, you wanted to place the standardized address in the same location that the input address was in, you could have the following settings in your setup file:

```
inAddress = 20, 40
outAddress = 20, 40
```

The above lines would cause the input address to be overwritten with the standardized address.

Gnzcoder Report Files

Audit Report

The Audit output file is generated if the keyword `audit` is initialized to greater than 0 in the format file, using the actual value of `audit` to determine which records to output as audit information. For example, if *`audit = 1000`*, every 1000th record will also be written to the audit output file. The file is named using the path and filename initialized as the `audFile` keyword in the format file.

Non-standardized File

The Non-Standardized output file is generated if the keyword *`nonStand`* is initialized to greater than 0 in the format file. The file is named using the path and filename initialized as the *`nStdFile`* keyword in the format file. All records that receive an `Exxx` match code are sent to this file **instead** of being sent to the standard output file. Initializing the *`nonStand`* keyword will override a keyword *`keepNoMatch`* setting of 0 and will output the non-standardized records to *`nStdFile`* instead of dropping them completely.

Statistics Report

The file gnzcoder.rpt is created with every pass of Gnzcoder. It is created in the current working directory and contains a statistical compilation of the most recent pass of Gnzcoder. Each successive Gnzcoder pass will overwrite the previous gnzcoder.rpt file. If you want to maintain historical data you should rename this file prior to the next Gnzcoder pass. The report file can be redirected by assigning a path and filename in the report file keyword.

Example Data and Format Files

A sample test file (testgnz.dat) and format file (gnzcoder.fmt) for this application are included with the library. Before using the test file, you will need to modify the reportFile keyword value in gnzcoder.fmt to reflect your file system structure.

Technical Support

Should you have any questions about the installation or functionality of Gnzcoder, please refer to the source code. If the answer is still elusive, contact Critchlow Associates’ Help Desk.

Gnztest

Gnztest is a sample executable program included with the library that demonstrates the use of the GeoStan New Zealand libraries. Please refer to the source code for a complete overview of how Gnztest functions.

Installing Gnztest

The file test.ini (included with the library) contains setup information for Gnztest. You’ll need to modify test.ini to fit your system’s configuration.

The format of the test.ini file consists of the following lines:

- Line 1: Search path for the GeoStan New Zealand data files
- Line 2: Full license file name
- Line 3: Password for license

For example:

```
S:\data\gsnz\geocodedatafiles\
C:\licensefiles\gsnz.lic
76434423
```

You can run Gnztest with no arguments or three arguments (interactive mode). If no arguments are specified, then the test.ini file is read from the current directory. If test.ini can not be found, the program stops.

Using Gnztest

Gnztest can be used in interactive mode or in batch mode. In interactive mode, Gnztest displays an abbreviated menu of different types of tests to run.

To Run in Interactive Mode

1. At the command prompt, enter:
   `gnztest.exe` *`searchpath licensefile password`*

2. Press ENTER to exit any level of Gnztest.

If any of the setup criteria does not initialize correctly, a list of problems will be displayed. Check and correct all of the listed problems and start the executable again.

The following menu will be displayed when all configurations are correct:

```
Search, Test, Browse, Generate Soundex, Options, Quit,
   (S/T/B/G/O/Q):
```

`Search`
: Prompts for an address (e.g., firm name, address line, suburb, and last line). Attempts to standardize and geocode this address. Returns the closest matching address or a list of possible matches, if applicable.

`Test`
: Allows the user to interactively enter input and output files for the self-test. See Table 8 for the self-test file format. In addition, the test file testgnz.dat is provided as an example.

`Browse`
: Allows the user to browse by street name or street name soundex value.

`Generate Soundex`
: Allows the user to generate a soundex value.

`Options`
: Allows the user to select capitalization and find options for **`GsNZFind`**.

`Quit`
: Halts the application.

To run in batch mode:

If the user supplies five arguments, the program runs in batch mode using the fourth and fifth arguments as the input and output files for running a self-test.

Table 8: CPC self-test file format

Record Position	Length	Description
001-006	6	Record Identifier
007-071	GSNZ_FIRM_NAME_LENGTH = 65	Addressee
072-171	GSNZ_ADDRLINE_LENGTH = 100	Address Line
172-201	GSNZ_SUBURB_NAME_LENGTH = 30	Suburb
202-226	GSNZ_CITY_LENGTH = 25	City
257-266	GSNZ_POST_CODE_LENGTH = 10	Post Code (no space)

Note: We suggest using Gnzcoder for most batch processing.

1. At the command prompt, enter:
   `gnztest` *`searchpath licensefile password inputAddressfile outputAddressfile`*

2. Press ENTER to exit any level of Gnztest.

   If any of the setup criteria does not initialize correctly, a list of problems will be displayed. Check and correct all of the listed problems and start the executable again.

   In batch mode, Gnztest runs the Test option.

Note: The makefile for Gnztest has already been modified for use on its respective platform.

Index