CorrectAddress

Version 10.0.0

User Guide

1 Beacon ST FL 33
Boston MA 02108
Tel: (888) 712-3332 (Support) or (888) 727-8822 (Sales)
Fax: (888) 882-7082

Email: dataquality.info@experian.com
Web: www.edq.com

Copyright

Trademarks

Table of Contents

Chapter1- Introduction

CorrectAddress® is a United States Postal Service® CASS Certified™ and Canada Post SERP-certified address correction, address verification and standardization software solution that enables users to cleanse, verify, and standardize their addresses, both in real-time and batch mode. Designed to overcome a significant degree of variation, fix misspellings and erroneous information, fill in omitted components and normalize incorrect formatting, *CorrectAddress* will easily manage vast quantities of data while exhibiting outstanding speed and accuracy.

- Address verification (U.S.A. and Canada)
- Advanced name and address parsing
- Address standardization
- Recognition of addresses of major corporations
- ZIP® correction
- ZIP+4® Code appending
- DPV™ (Delivery Point Validation) for USA and PoC (Point of Call) verification for Canada
- LOT (Line of Travel) coding
- LACS$^{Link_{™}}$ (Locatable Address Conversion System)
- Suite$^{Link_{™}}$ coding
- Delivery point bar coding with Postnet barcode font included
- Carrier route codes
- Support for FIPS codes
- Geocoding Add-on
- Merge-Purge and Deduplication Add-on

CorrectAddress is a powerful and accurate, multi-platform, multi-user solution that fits a variety of development frameworks and production environments. Intuitive APIs allow programmers to plug address verification and address standardization functionality into existing enterprise applications, such as point-of-sale, screening and payroll services, customer management and order tracking just to name a few. The product can be automated for daily or weekly, address correction, batch jobs or it can handle interactive address correction.

Powered by Experian's advanced fuzzy searching and matching engine, *CorrectAddress* delivers unsurpassed address correction speed and accuracy, as compared to standard address handling methodologies. The accuracy is further enhanced as Experian regularly sends out updates when the postal data is changed.

CorrectAddress will save you money by correcting, parsing, and standardizing every address in your database. It also adds the Carrier Route, LOT, ZIP+4 and Delivery Point Barcodes (DPBC) to every deliverable address.

CorrectAddress easily integrates into many platforms including Microsoft® Windows®, Linux®, Sun Solaris®, AIX. *CorrectAddress* seamlessly integrates with SQL Server, Oracle®, DB2®, TeraData®, Sybase®, MySQL, PostgreSQL, Progress and other commercially available database systems. In addition it comes with a set of APIs that enable applications to call *CorrectAddress*® from programming environments like Java, VB.NET, ASP, ASP.NET, C#, C/C++, PHP, Perl, COBOL, SQL, PowerBuilder, etc.

NEW AND IMPROVED IN VERSION 10.0

The latest and most comprehensive release of *CorrectAddress* has met all requirements and passed all tests administered by the United States Postal Service to maintain CASS Certification for Windows, Linux and UNIX-based platforms. The product is CASS Certified for Cycle N (2023-05).

The new release contains CASS Cycle O updates.

- A comprehensive and advanced installation wizard
- PO Box™ Only Delivery Zones
- 5-Digit ZIP Validation
- DPV® Flags updates
- R777/R779 Phantom Route Matching
- PBSA – PO Box Street Address Identifier
- CMRA – PMB Identifier & DPV® Confirmation
- Single Trailing Alpha on a Primary Number
- Deliverable Street / Highrise Default improved matching
- SuiteLink® matching improvements
- Military Addressing
- Cross State Addresses
- TotalDPS
- Informed Addressing

Software developers can expect:

- Simple and elegant Application Programming Interfaces (API)
- API - Support for all versions of .NET framework
- GUI and Batch - Support for .NET 4.8
- Support for of Microsoft SQL Server
- The availability of SQL Server Integration Services (SSIS) components.
- Pre-built binary modules for 32- and 64-bit UNIX platforms. Main supported platforms - IBM AIX 64 bit 7.2, Ubuntu 20.04, Sun Solaris 64bit (Sparc) 11.4, CentOS 7 - latest version.
- Support for Windows 32-bit and 64-bit operating systems. Windows server: Windows Server 2016, Windows Server 2019 (Main Supported Platform), Windows Server 2022 . Windows Desktop: Windows 10 – latest version (Main Supported Platform), Windows 11 – latest version.
- Software Development Kits (SDKs) available for C/C++, C#, VB.NET, Visual Basic, Java, SQL Server (T-SQL), Oracle (PL/SQL), MySQL, DB2, Perl, PHP, COBOL and a variety of other languages and frameworks.
- Functional library is entirely independent of the postal data and does not require to be updated monthly

Also available:

CorrectAddress® Plus – Perform data de-duplication and merge/purge operations in conjunction with address validation processes.

Geocoder Add-on-– Enhance your addresses with latitude, longitude, Census tract, block, FIPS state, county, congressional district codes, and statistical area information.

Residental Delivery Indicator (RDI™) – The Residential Delivery Indicator (RDI™) add-on enables users to determine whether a given address is classified as a residential or a business address. The RDI process may be run directly or as part of a standard address lookup. In order to obtain RDI data, users need to sign a license agreement with the USPS. Once the agreement is signed and the fee is paid, the USPS will start shipping monthly data updates. See Appendix L for more

details.

Chapter2- Installation

SYSTEM REQUIREMENTS

OS:

Windows Server
- **Windows Server 2016**
- **Windows Server 2019 (Main Supported Platform)**
- **Windows Server 2022**

Windows Desktop
- **Windows 10 – latest version (Main Supported Platform)**
- **Windows 11 – latest version**

Unix/Linux
- **Sun Solaris 64bit (Sparc) 11.4**
- **IBM AIX 64 bit 7.2**
- **CentOS 7 - latest version**
- **Ubuntu 20.04**

Disk Space 4 GB+

Memory 2 GB+

Data files arrive monthly and need to be copied over the previous month's files to stay up to date.

Unlike earlier versions of the product, v10.0 does not require users to replace the library file with every data update.

WINDOWS INSTALLATION

Prior to updating a previously installed version of CorrectAddress, make sure that all instances of the program are closed, and that no applications are using the library.

1. You will be prompted to choose a directory for installation; the default directory name is **Program Files\Intelligent Search Technology\CorrectAddress.**
2. The install will then place the DLL used with the GUI in your system32 directory and copy the program itself into the directory you specified. A shortcut to the GUI will be placed in your **Start Menu** under **Program Files\Intelligent Search Technology**.
3. Data files will be copied into the **\Data** subdirectory. The installation will create an initialization file (**CorrectA.ini**) in your **\system32** directory containing the location of the data files; this is vital for the program to function properly. If you ever change the location of the data files, you may go to **Tools→Options** and modify the **Data Path Settings configuration**, or simply modify the .ini manually
4. To uninstall *CorrectAddress*, use the Windows Control Panel to **Add/Remove Programs** and select **CorrectAddress** for removal.

UNIX/LINUX INSTALLATION

The files involved in *CorrectAddress v10.0* installation on a UNIX machine are:

MONTHLY DATA ARCHIVES

CorrectAddressData.zip

canada.zip (Canada Post installation disk only)

SOURCE ARCHIVE

Available upon request

PRE-BUILT SHARED LIBRARY OBJECTS (BY PLATFORM)

libCorrectA.so

INSTALL CORRECTADDRESS FROM THE UNIX DVD

Unlike earlier versions of CorrectAddress, v9.0 and later do not require monthly shared object updates. Once the shared object or library is in place, it may be used in combination with any subsequent postal data set.

Using pre-built shared objects (PREFERRED):

1. Copy contents of all disks into a temporary data directory on the UNIX machine.
2. Extract contents of all data archives into the data directory (e.g., **/IstCorrectAddress/Data**)
3. Extract the shared object (**libCorrectA.so**) from **<platform>/libCorrectA.zip** (e.g., **linux32/libCorrectA.zip**) to your local machine.

Using source archive:

The following procedure is optional and should only be followed by users who:

- Run *CorrectAddress* on platforms for which no pre-built shared objects are shipped
- Wish to custom compile their libraries to include platform-specific optimization flags
- Wish to compile custom wrappers for PERL or PHP

1. Copy contents of all disks into a temporary data directory on the UNIX machine.
2. Extract contents of **source_v90.zip** into the source directory (e.g., **/IstCorrectAddress)**
3. Extract contents of all data archives into the data directory (e.g., **/IstCorrectAddress/Data**)
4. Build the shared object as described in the *Shared Objects* on *Page 4-2*.

By default, the shared object will look for data files in **/IstCorrectAddress/Data** directory. To override default data path settings, set **CA_DATA** environment variable accordingly: export **CA_DATA=/some_dir/Data.**

If Canada Post data is installed, by default, the shared object will look for data files in **/IstCorrectAddress/DataCanada** directory. To override default data path settings, set **CA_DATA_CANADA** environment variable accordingly: export **CA_DATA_CANADA=/some_dir/DataCanada**.

Chapter3- Registering the Software

For Windows installations, a license file **istca.lic** must reside in the same directory as the *CorrectAddress* library (**CorrectA.dll**). This file is specific to the machine that the library is running on. Generating a license file entails registering the software with Experian.

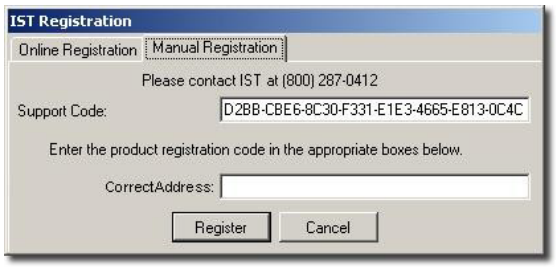
MANUAL WINDOWS REGISTRATION (WINDOWS)

On Windows, registration and generation of the license file is performed through the Windows GUI.

To activate Windows registration, run **C:\Program Files\Intelligent Search Technology\CorrectAddress\CorrectGUI.exe** program, or **Start→All Programs→ CorrectAddress.**

In the case where online registration is not possible because you do not have an internet connection or have firewall limitations, your registration form will indicate that you must register manually after clicking on the **Register** button.

A 9-digit **Support Code** number will be provided to you as shown below. It is highly recommended that you write this number down.



Contact Experian and relay the **Support Code** number that was returned to you. A program-unlocking key will be supplied to you to enter that will complete your registration.

UNIX REGISTRATION

On UNIX/Linux, product registration is performed at program load time. No special license files are required.

Chapter4-BeforeStarting

ABOUT SHARED OBJECTS

The following section describes how to create port and test *CorrectAddress* shared objects (dynamic libraries) on UNIX-based platforms. Windows users and UNIX users who utilize pre-built shared objects should skip this section.

You must have a Java Runtime Environment or JDK installed as well as a C compiler to create a shared object. This is available upon request.

AVAILABLE UNIX OS AND COMPILERS

To build CorrectAddress we need your choice about Operating System and Compiler in order to generate the default compiler and linker commands.
Generally, the up-to-date choice about compiler is gcc or clang which are available on almost all UNIX systems.
We offer default compiler and linker commands for these Operating Systems (the version of the compilers is the minimum required; later versions should work as well):

AIX
gcc/g++
(GCC) 8.3.0
xlcclang/xlcclang++
IBM XL C/C++ for AIX, V16.1.0

Solaris SPARC
gcc/g++
(GCC) 9.5.0

Linux
gcc/g++
(GCC) 4.8.5
clang/clang++
version 10.0.0

The buildLib tool will automatically generate default commands for C compilation, C++ compilation and linking which are needed to build the CorrectAddress library.
You may also provide your own compiler and linker commands based on your needs. The only requirement is to use compilers which support at least the C++11 standard
like gcc/g++, clang/clang++, xlclang/xlclang++(AIX).

RUNNING THE BUILDER

Run the Library Builder by executing the following command:

```
java –jar BuildLib.jar
```

This will start a configuration script. If you want to use a previously saved configuration file (**conf.txt**) in an automated environment, execute **java –jar BuildLibAuto.jar** instead. Make sure that the configuration file is located in the same directory as the .jar file.

SETTING UP SYSTEM CONFIGURATION

The first time you run this program it will ask you for your environment specifications, including:

- Operating system
- C\C++ compiler name
- Type of shared object (standard or with Java support)

If you include Java support, you will need to supply the paths on your system to your JDK include directories. JDK (if not present already) must be installed prior to this step. Within the default Java directory created by JDK there is a directory called include. This is the path you specify when prompted to.

Within that include directory should be another directory that is platform specific; the name will refer to your operating system. For example, on a Windows based machine the directory would be named win32. This is the path you specify if prompted to enter the location of your Java include/platform directory.

```
enter the full path to your Java include directory:
==>
```

After a configuration is created you may use it in future installations.

BUILDING SHARED OBJECTS

After your initial configuration you will be asked if you want to change this configuration. Select NO if there are no changes or select YES and change the previous configuration.

Next BuildLib will automatically generate C compile command, C++ compile command and a linker command based on your settings for Operating system and Compiler.
After that you will be asked if you need to change any of the commands generated. If you don't need to do this choose 'n'. If you need to provide your own compiler and linker commands choose 'y'. After that you will be asked which exactly command you need to change.
You have to choose "1" to change the C compile command, "2" to change the C++ compile command and "3" to change the linker command.
You can also choose "4" in case you need to reset the commands to the default ones.
The process can be repeated as many times as you need until you answer 'n' to the question
"Do you want to make changes to the commands or reset to the default ones?".

After the program completes you will be asked if you wish to clean up the object files. Select YES unless you are including PHP, PERL, MySQL or PostgreSQL support as you will need them to create these special modules.

The shared object is now built and ready to be registered and used with your applications.

RUNNING DEMO PROGRAMS ON UNIX-BASED SYSTEMS

CorrectAddress comes with a number of utilities that demonstrate its functionality. They are described in more detail in *Demo Files* on *Page 4-2*. To create one or more of the demo programs, type:

```
$CC -pthread -m64 –o $PROGRAM_NAME $DEMO_FILENAME $SHARED_OBJECT_NAME
```

For example, to create a program named **CADemo** using a gcc compiler,

```
gcc -pthread -m64 –o CADemo CADemo.c ./libCorrectA.so
```

type: You can now execute **CADemo** from the shell prompt.

The -pthread flag is needed to include the POSIX Thread library.
The -m64 flag switches on the 64 bit mode, it can also be -maix64 for gcc/AIX or -q64 for xlclang/AIX.

DEMO FILES

CADemoCorrectA.c – This file contains a sample execution of the product; read through it to check the syntax used when making calls to the **CorrectA** function. This example will take in a misspelled version of 445 Hamilton Ave Ste 608, White Plains NY 10601 and correct it to the appropriate spelling and assign a four digit add-on to the ZIP code. This will be displayed upon running. Other info is returned but will not be displayed by print statements. Explanations of parameters are given in the section titled: Calling **CorrectA** from C.

CallCorrectA.c – This is a simple interactive demo. The user is asked to enter delivery line 1, line 2 and last line (city/state/ZIP). The program returns corrected result onto the screen.

CADemoFindCityCounty.c – This is a demo example that returns city/county information for a particular ZIP code. Explanations of parameters are given in the section titled: Calling **CorrectA** from C.

CADemoFindZipCity.c – This demo returns all ZIP files that belong to particular city in a state. Explanations of parameters are given in the section titled: Calling **CorrectA** from C.

CADemoLOTUtility.c – This example shows how Line of Travel information is returned based on ZIP and carrier route information. Explanations of parameters are given in the section titled: Calling **CorrectA** from C.

CADemoParseAddress.c – This demo calls the **ParseAddress** function that splits address information into fields, such as street number, street name, suffix, city, state, ZIP etc.

CallFileImport.c – A demo example that can take a text file (delimited or fixed width) and return a text file with corrected values in it. The program takes the path of an .ini file as input. The .ini file information is structured as follows and a sample **.ini** file is included (called **CAImport.ini**), as well as a sample piece of data in the file **testCA.txt**. To better understand the syntax, open the example **CAImport.ini** file, and compare as you read through the instructions below.

INI FILE FORMAT

The schema below is for UNIX. Refer to Appendix G for description of a Windows batch configuration file, G :

```
[CorrectAddress Configuration File]

[THREAD]

<threadname>

<startindex>

<endindex>

UpdateProcessPath: <filepath>

CancelProcessPath: <filepath>

ErrorLogPath: <filepath>


[INPUT]
Type: Text
Input: <filepath>
Format: Delimited or FixedWidth
[Delimited Case:
TEXTDELIM: "
DELIM: ,

]

ColHeader: True
Cols: num Columns

[Options]

[Output]
same as input

[AddressValues]

[<field type(firm,urbanization,dline1,dline2,and lastline)>]
```

The thread section sets up how the data is processed. For each thread, the following information needs to be specified inside this section: name of the thread, starting index (1-counted), ending index, where the update ini file should be, where the cancel ini file should be, and where the error ini file is. If you want a thread to process all of the data in the input file, specify **0** as the start and end indexes.

The Options section can the following options specified:

Geo: True | False	<Turn On/Off Geocoding>	By default, False
NoMatch: True | False	<Output/Not No Match Records>	By default, True
MixedCase: True | False	<Ouput Mixed/Upper Case>	By default, True
PS3553 TEXT: True | False	<Generate a text PS3553Form>	By default, False
PS3553 XML: True | False	<Generate an XML PS3553Form>	By default, False
Codes:	<Add custom option flags (See Appendix I)>	By default, blank

In the AddressValues section, you can place more than one field in a field type (each on a separate line). This means that, for instance, the field type **Dline1** contains the street address (without city/state/zip). If you had the field's streetnumber, predirectional, streetname, postdirectional, and streetsuffix and they contained the pieces of the street address that their names imply, then this part of the ini file would look like this.

In the code, this takes these fields in order and concatenates them into one string that it then uses as the input street address. This is useful when your file's information is split into separate fields and you have no pure field with street address information in it.

```
[DLINE1]
streetnumber
predirectional
streetname
postdirectional
streetsuffix
```

This process continues down until you reach...

```
[END ADDRESSVALUES]
[PS3553]
True | False <generate form PS3553>

/*****************************************************\
```

In the [Output] section, you can return only certain fields.

The names of the *CorrectAddress* fields you can return, and their values are as follows:

Firm	Firm name
Urbanization	Urbanization name (Puerto Rico Only)
Dline1	Primary delivery address (i.e. 445 Hamilton Ave Suite 608)
Dline2	Secondary delivery address (only used in dual addressing, usually blank)
LastLine	Contains city, state, and zip (i.e. White Plains, NY 10601-2306)
Streetnum	Street number
Addon	Four digit add-on to zip code
Checkdigit	Checkdigit sum used with DPC
DPC	Delivery Point Code, used for creating bar codes
City	City name
County	County name
Countynum	County number
Croute	Carrier route
LACS	Locatable Address Conversion System Indicator
LOT	Line of Travel number

PMB	Any private mail box designation
Aptname	4-character apartment abbreviation (i.e. STE, APT, etc.)
Aptnum	Apartment number
State	State abbreviation
ZIP	Five digit zip code
ZIP-Addon	5-digit zip code + four-digit add-on
Predir	Pre-directional (i.e. 445 N Hamilton Ave)
Postdir	Post-directional (i.e. 445 Hamilton Ave S)
Streetname	Street name (445 Hamilton Ave)
Suffix	Street suffix (445 Hamilton Ave)
Errcode	Error code string
Retcode	Return code

Additionally DPV, LACS, and Geocoding output fields are specified in their respective sections.

Chapter5- ApplicationProgrammingInterface(API)

MAIN FUNCTIONS

The following is a list of currently supported *CorrectAddress* API functions.

Function	Description
CorrectA	Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing. All near matches are returned when applicable.
CorrectAWorld	Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing. Verifies and corrects Canadian addresses. All near matches are returned when applicable.
CorrectACASS	Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing. All near matches are returned when applicable. Includes path to a text file to store address statistics for generating CASS report (Postal Form PS3553, see Appendix A). This function is used in conjunction with **PrintPSForm3553**.
capconv	Converts a string to mixed ("Proper") case format.
DPCutility	Accepts street number, unit number and 9-digit ZIP code as input. Creates Delivery Point Code (DPC) and Checkdigit values.
FindCityCounty	Accepts ZIP code as input. Returns preferred city name, state, county information.
FindCityState	Accepts ZIP code as input. Returns all valid mailing city names, state, county information.
FindStateCounties	Accepts state abbreviation as input. Returns all county names and numbers within the state.
FindZipCity	Accepts city name and state abbreviation as input. Returns list of all ZIP codes for the given city/state area.
GetBuildDate	Retrieves *CorrectAddress* USPS data build time.
GetBuildDateCanada	Retrieves *CorrectAddress* Canada Post data build time.
GetCAVersion	Retrieves *CorrectAddress* version number.
getMuniProv	Accepts Canadian postal code as input. Returns municipality and province information.

Function	Description
isBusinessZip	Accepts 9-digit ZIP code (with or without 2-digit Delivery Point Code (DPC)) and the path to the RDI lookup tables. (See Appendix L) Returns residential delivery indicator.
LOTutility	Accepts 9-digit ZIP code, Delivery Point Code (DPC) and carrier route number. Returns Line of Travel (LOT) information.
ParseAddress	Breaks up input address into individual parts. Performs basic standardization.
PrintPSForm3553	Accepts path to file containing statistics generated by **CorrectACASS**. Creates CASS report (Postal Form PS3553) in text format.
RunCABatch	Accepts configuration file as input (see Appendix G) Runs a batch of addresses through **CorrectA** function according to specifications in the configuration file.
strerrorCA	Accepts two-character error code as input. Returns plain text description of the error code.
GetErrcodeString	Same as strerrorCA. Supports .NET managed code.
unloadData	Explicitly unloads postal data from memory.
freeHashTables	Free global hash tables

GEOCODING FUNCTIONS

The following is a list of functions used by the Geocoder add-on.

Function	Description
TigerCA	Validates, CASS-standardizes and geo-codes input address with Delivery Point Validation (DPV) and LACSLink™ processing. All near matches are returned when applicable.
GeoCode	Accepts 9-digit ZIP code as input. Returns end-points of latitude and longitude ranges and other geo-coding information based on ZIP+4 area.
getCentroid	Accepts ZIP code as input. Returns latitude and longitude coordinates of ZIP area centroids.

AUXILIARY/WRAPPER FUNCTIONS

The following is a list of auxiliary *CorrectAddress* functions.

Function	Description
db2CorrectA	**CorrectA** wrapper function for integration with DB2 database systems.
db2TigerCA	**TigerCA** wrapper function for integration with DB2 database systems.
CorrectAcat	**CorrectA** wrapper function for integration with environments that prohibit long function argument lists. Excludes Stringaddress parameter from the output.
CorrectAcat2	**CorrectA** wrapper function for integration with environments that prohibit long function argument lists. Includes Stringaddress parameter in the output.
CorrectACASSOracle	**CorrectACASS** wrapper function for integration with Oracle database systems.
CorrectACobol	**CorrectA** wrapper function for integration with environments that do not allow integer return codes. Includes optional flag to return secondary information on delivery line 2.
CorrectAN	**CorrectA** variant function that allows setting the maximum number of results per address.
CorrectAOracle	**CorrectAWorld** wrapper function for integration with Oracle database systems. Includes optional flag to return secondary information on delivery line 2.
FindCityCountyCobol	**FindCityCounty** wrapper function for integration with environments that do not allow integer return codes.
TigerCAcat	**TigerCA** wrapper function for integration with environments that prohibit long function argument lists. Excludes Stringaddress parameter from the output.
TigerCAcat2	**TigerCA** wrapper function for integration with environments that prohibit long function argument lists. Includes Stringaddress parameter in the output.
TigerCAN	**TigerCA** variant function that allows setting the maximum number of results per address.
TigerCAOracle	**TigerCA** wrapper function for integration with Oracle database systems.

CORRECTA

Description

Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACS$^{Link™}$ processing.

All near matches are returned when applicable.

C prototype

```
int CorrectA(       char inputAddress[194],
                    char sentLen[4],
                    char errcode[30],
                    char firmname[40],
                    char urbanization[28],
                    char Dline1[64],
                    char Dline2[64],
                    char LastLine[64],
                    char Stringaddress[260],
                    char DPC[2],
                    char Checkdigit[1],
                    char cityname[28],
                    char stcode[2],
                    char zip[5],
                    char  addon[4],
                    char  croute[4],
                    char LACS[1],
                    char LOTsequence[4],
                    char LOTcode[1],
                    char PMB[12],
                    char results[200][194],
                    char strnum[10],
                    char secname[4],
                    char secnum[8],
                    char countyname[25],
                    char countynum[3]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery line

2 information

(OUTPUT) city, state, ZIP information
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code
Checkdigit
(OUTPUT) Delivery point check digit
cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see *Appendix F* for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, MySQL, PERL, PHP, PostgreSQL, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

CORRECTAWORLD

Description

Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing.

Verifies and corrects Canadian addresses.

All near matches are returned when applicable.

CorrectAWorld is a modified version of the **CorrectA** API that facilitates address matching against both U.S. and foreign-address databases. By default **CorrectAWorld** will attempt to determine the country database that is appropriate for the given input based on address keywords and formatting. Users may also specify their desired search database by passing the appropriate country flag in the errcode field (see *Remarks* below).

C prototype

```
int CorrectAWorld( char inputAddress[194],
                   char sentLen[4],
                   char errcode[30],
                   char firmname[40],
                   char urbanization[28],
                   char Dline1[64],
                   char Dline2[64],
                   char LastLine[64],
                   char Stringaddress[260],
                   char DPC[2],
                   char Checkdigit[1],
                   char cityname[28],
                   char stcode[2],
                   char zip[5],
                   char  addon[4],
                   char  croute[4],
                   char LACS[1],
                   char LOTsequence[4],
                   char LOTcode[1],
                   char PMB[12],
                   char results[200][194],
                   char strnum[10],
                   char secname[4],
                   char secnum[8],
                   char countyname[25],
                   char countynum[3]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below*)* (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery

line 2 information

(OUTPUT) city, state, ZIP information (municipality, province, postal code for Canada)

Stringaddress

(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-K)

DPC

(OUTPUT) Delivery point code

Checkdigit

(OUTPUT) Delivery point check digit

cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code

LOTsequence

(OUTPUT) Line of Travel sequence number

LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number

countyname

(OUTPUT) county name

countynum

(OUTPUT) countynum

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Users may specify a country-specific search database by passing the appropriate flag in the *errcode* field:

"Us" – Search only the USPS database (United States and territories)
"Cd" – Search only the Canadian address database

The results string may contain multiple records, but users must check the *errcode* string to determine the record size and layout, which is different for each country. See Appendix C for information about country-specific error codes. See Appendix F for information about country-specific record formats.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, PHP, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

CORRECTACASS

Description

Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing.

All near matches are returned when applicable.

Includes path to a text file to store address statistics for generating CASS report (Postal Form PS3553, see Appendix A). This function is used in conjunction with **PrintPSForm3553**.

C prototype

```
int CorrectACASS( char inputAddress[194],
                  char sentLen[4],
                  char errcode[30],
                  char firmname[40],
                  char urbanization[28],
                  char Dline1[64],
                  char Dline2[64],
                  char LastLine[64],
                  char Stringaddress[260],
                  char DPC[2],
                  char Checkdigit[1],
                  char cityname[28],
                  char stcode[2],
                  char zip[5],
                  char  addon[4],
                  char  croute[4],
                  char LACS[1],
                  char LOTsequence[4],
                  char LOTcode[1],
                  char PMB[12],
                  char results[200][194],
                  char strnum[10],
                  char secname[4],
                  char secnum[8],
                  char countyname[25],
                  char countynum[3],
                  char path[256]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery line

2 information

(OUTPUT) city, state, ZIP information
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code
Checkdigit
(OUTPUT) Delivery point check digit
cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum
path
(INPUT) location of a text file to store address statistics for generating CASS report

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

CorrectAddress Graphical User Interface (see Chapter 6) is normally used for batch database jobs and can be used to create a facsimile copy of PS Form 3553 on your default printer. PS Form 3553 can be used as proof of valid addresses when approaching the United States Postal Service for a rate discount on bulk mail. In the case that you are integrating *CorrectAddress* into an application and not using the GUI for your batch jobs, the function **CorrectACASS** allows you to keep track of your records and create a text file with a facsimile PS Form 3553 that you can later print out.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, PERL, PostgreSQL, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

CAPCONV

Description

Converts a string to mixed (“Proper”) case format.

C prototype

```
int capconv(        char funct[9],
                    char namein[80],
                    char nameout[80]);
```

Parameters

funct namein nameout

Return codes

input service name (see *Remarks* below) input string

output string

Remarks

Examples

Return code: 0

All input parameters must be initialized prior to calling the function. Input service

name must be set to **"CADLL "** (note the spaces)

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, Oracle, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

DPCUTILITY

Description

Accepts street number, unit number and 9-digit ZIP code as input.

Creates Delivery Point Code (DPC) and Checkdigit values

C prototype

```
int DPCutility(     char DPC[2],
                    char Checkdigit[1],
                    char strnum[10],
                    char secnum[8],
                    int secflag,
                    char zipcode[11]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

DPC
(OUTPUT) delivery point code
Checkdigit
(OUTPUT) delivery point check digit
strnum secnum secflag zipcode

Return codes

(INPUT) primary (street) number (INPUT) secondary (unit)

number

(INPUT) address type flag (see *Remarks* below) (INPUT) ZIP code with

+4 addon

Remarks

Examples

-66 – out of memory
All other codes: DPC generation completed successfully.

All input parameters must be initialized prior to calling the function. Address type

flag can be set to one of the following:

1. general delivery address
2. highrise address
3. firm address

Any other value will assume a regular street address.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, VB.NET

Examples for other languages and environments are available upon request.

FINDCITYCOUNTY

Description

Accepts ZIP code as input.

Returns preferred city name, state, county information.

C prototype

```
int FindCityCounty( char ZIP[5],
                    char cityname[28],
                    char state[2],
                    char countyname[25],
                    char countynum[3]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

ZIP

cityname state

input ZIP code (OUTPUT) city name

(OUTPUT) state abbreviation

countyname
(OUTPUT) county name

countynum
(OUTPUT) county number

Return codes

0 – completed successfully
-2 – no valid license key
-3 – trial expired
-99 – no matches found

Remarks

Examples

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, MySQL, Oracle, PERL, PHP, Ruby, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

FINDCITYSTATE

Description

Accepts ZIP code as input.

Returns all valid mailing city names, state, county information.

C prototype

```
int FindCityState(    char ZIP[5],
                      char results[50][55]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

ZIP

input ZIP code

results

(OUTPUT) array of up to 50 valid mailing city names, state abbreviations and county names.

Return codes

Return code value is the total number of valid mailing city names found.

Remarks

All input parameters must be initialized prior to calling the function.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, Oracle, VB.NET

Examples for other languages and environments are available upon request.

FINDSTATECOUNTIES

Description

Accepts state abbreviation as input.

Returns all county names and numbers within the state.

C prototype

```
int FindStateCounties(      char state[2],
                            char countynames[260][25],
                            char countynums[260][3]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

state
input state abbreviation
countynames
(OUTPUT) array of up to 260 county names within the given state
countynames
(OUTPUT) county numbers corresponding to county names above

Return codes

Return code value is the total number of county names found.

Remarks

Examples

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, PHP, Ruby, VB.NET

Examples for other languages and environments are available upon request.

FINDZIPCITY

Description

Accepts city name and state abbreviation as input.

Returns list of all ZIP codes for the given city/state area.

C prototype

```
int FindZipCity(    char cityname[28],
                    char newcityname[28],
                    char state[2],
                    char zips[200][5]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

cityname
input city name
newcityname
(OUTPUT) validated city name
state zips

Return codes

input state abbreviation

(OUTPUT) list of all ZIP codes within the given city/state area

Remarks

Return code value is the total number of county names found. Negative return code indicates invalid license conditions.

All input parameters must be initialized prior to calling the function.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, Oracle, PERL, PostgreSQL, PHP, Ruby, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

GETBUILDDATE

Description

Retrieves *CorrectAddress* USPS data build time.

C prototype

```
int GetBuildDate(    char day[2],
                     char month[2],
                     char year[4]);
```

Parameters

day month year

Return codes

(OUTPUT) build day of the month (OUTPUT) build month

(OUTPUT) build year

Remarks

Return code: 0.

All parameters must be initialized prior to calling the function.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, VB.NET

Examples for other languages and environments are available upon request.

GETBUILDDATECANADA

Description

Retrieves *CorrectAddress* Canada Post data build time.

C prototype

```
int GetBuildDateCanada(     char day[2],
                            char month[2],
                            char year[4]);
```

Parameters

day month year

Return codes

(OUTPUT) build day of the month (OUTPUT) build month

(OUTPUT) build year

Remarks

Return codes:

> 0 – success
< 0 - failure

All parameters must be initialized prior to calling the function.

Examples

See examples for GetBuildDate() . Specific examples for this function are available upon request.

GETCAVERSION

Description

Retrieves *CorrectAddress* version number.

C prototype

```
int GetCAVersion(char version[30]);
```

Parameters

version
(OUTPUT) full version number

Return codes

Remarks

Examples

Return code is a numeric representation of the version number.

None

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, VB.NET

Examples for other languages and environments are available upon request.

GETMUNIPROV

Description

Accepts Canadian postal code as input.

Returns municipality and province information.

C prototype

```
int getMuniProv(    char postalcode[6],
                    char province[2],
                    char municipality[30]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

postalcode
Canada Post postal code

province
(OUTPUT) province code

municipality
(OUTPUT) municipality name

Return codes

1 – municipality/province combination found
0 – no match found

Remarks

Examples

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, VB.NET

Examples for other languages and environments are available upon request.

ISBUSINESSZIP

Description

Accepts 9-digit ZIP code (with or without 2-digit Delivery Point Code (DPC)) and the path to the RDI lookup tables. (See Appendix L)

Returns residential delivery indicator.

C prototype

```
int isBusinessZip(char zip[11],
                  int length,
                  char file9_path[256],
                  char file11_path[256]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

zip

length file9_path

input 9-digit ZIP+4 code or 11-digit ZIP+4+DPC

input ZIP length (9 or 11) path to 9-digit RDI lookup

file

file11_path

path to 11-digit RDI lookup file

Return codes

See Appendix L.

Remarks

Examples

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, Java

Examples for other languages and environments are available upon request.

LOTUTILITY

Description

Accepts 9-digit ZIP code, Delivery Point Code (DPC) and carrier route number.

Returns Line of Travel (LOT) information.

C prototype

```
int LOTutility(     char LOTsequence[4],
                    char LOTcode[1],
                    char croute[4],
                    char zipcode[11],
                    char DPC[2]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode croute zipcode DPC

Return codes

(OUTPUT) Line of Travel code (INPUT) carrier route

number (INPUT) 9-digit ZIP code (INPUT) delivery

point code

Remarks

Examples

-66 – out of memory
All other codes: LOT generation completed successfully.

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, VB.NET

Examples for other languages and environments are available upon request.

PARSEADDRESS

Description

Breaks up input address into individual parts. Performs basic standardization.

C prototype

```
int ParseAddress(    char inputAddress[194],
                     char sentLen[4],
                     char strnum[10],
                     char predir[2],
                     char strname[28],
                     char postdir[2],
                     char strsuffix[4],
                     char secname[4],
                     char secnum[8],
                     char cityname[28],
                     char stcode[2],
                     char zip[5],
                     char addon[4],
                     char urbanization[28],
                     char PMB[12]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen strnum predir strname postdir strsuffix secname secnum cityname stcode zip addon

input address length (see *Remarks* below) (OUTPUT) primary

(street) number (OUTPUT) pre-directional abbreviation

(OUTPUT) street name

(OUTPUT) post-directional abbreviation (OUTPUT) street suffix

(OUTPUT) secondary (unit) designator (OUTPUT) secondary (unit)

number (OUTPUT) City name

(OUTPUT) State abbreviation (OUTPUT) 5-digit ZIP

code

(OUTPUT) +4 extension for the ZIP code

urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)

PMB
(OUTPUT) Private mail box number

Return codes

-66 – out of memory
All other codes: address parsing completed successfully.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, Oracle, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

PRINTPSFORM3553

Description

Accepts path to file containing statistics generated by **CorrectACASS**.

Creates CASS report (Postal Form PS3553) in text format.

C prototype

```
int PrintPSForm3553(char path[256]);
```

Parameters

path
input path to the job statistics file generated by **CorrectACASS** function (null-terminated)

Return codes

Remarks

Examples

1 – form generated successfully 0 – unable to
generate the form

Generates text file named "{statistics_file_name}PS3553.txt" in the directory where the statistics file is located. All input

parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, Oracle, PERL, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

RUNCABATCH

Description

Accepts configuration file as input (see Appendix G).

Runs a batch of addresses through **CorrectA** function according to specifications in the configuration file.

C prototype

```
int RunCABatch(char path[256]);
```

Parameters

path
(INPUT) configuration file containing job settings

Return codes

Remarks

Examples

>=0 – completed successfully
-2 – no valid license key
-3 – trial expired

All input parameters must be initialized prior to calling the function. Configuration file format is provided in Appendix G.

This function takes place of the older **ImportFile()** routine, which has been deprecated.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C

Examples for other languages and environments are available upon request.

STRERRORCA

Description

Accepts two-character error code as input.

Returns plain text description of the error code.

C prototype

```
char *strerrorCA(char errcode[2]);
```

Parameters

errcode
(INPUT) U.S. error code

Return codes and error codes

See Appendix C.

Remarks

Examples

All input parameters must be initialized prior to calling the function.
This function cannot be used from managed code. For integrations into .NET applications, use GetErrcodeString() instead.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, Java, Oracle

Examples for other languages and environments are available upon request.

GETERRCODESTRING

Description

Accepts two-character error code as input.

Returns plain text description of the error code.

C prototype

```
int GetErrcodeString(const char errcode[2], char errcodedesc[128]);
```

Parameters

errcode
(INPUT) U.S. error code
errcodedesc
(OUTPUT) U.S. error code description

Return codes and error codes

See Appendix C.

Remarks

Examples

All input parameters must be initialized prior to calling the function.
Unlike strerrorCA(), this function supports applications written in managed and unmanaged code.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C#, VB.NET

Examples for other languages and environments are available upon request.

UNLOADDATA

Description

Explicitly unloads postal data from memory.

C prototype

```
int unloadData();
```

Parameters

None

Return codes and error codes

Remarks

Examples

Return code: 1

This function can be used in environments that do not explicitly unload the library after the process completes (e.g., Java). Postal data is loaded into memory on the first call to the library (unless "Zd" flag is set in the code – See Appendix I) and remains memory-resident for the duration of the process.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Java

Examples for other languages and environments are available upon request.

FREEHASHTABLE

Description

Free global hash tables

C prototype

```
void freeHashTables();
```

Parameters

None

Return codes and error codes

None

Remarks

Free global hash tables, currently used for RDI data. In case CorrectAddress library has to be unloaded and loaded again this function needs to be called before unloading the library for avoiding memory leaks. If the application doesn't unload the library the function call is not needed.

Examples

Code samples can be found in the \Development kits folder in the CorrectAddress installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Java

Examples for other languages and environments are available upon request.

TIGERCA

Description

Validates, CASS-standardizes and geo-codes input address with Delivery Point Validation (DPV) and LACSLink™ processing.

All near matches are returned when applicable. Verifies and corrects Canadian addresses.

TigerCA is an extension of the **CorrectA/CorrectAWorld** API with geocoding support. The function will first perform a search against the USPS database, and if the search fails, will attempt validation against Canada Post data. Geocoding information is currently only available for addresses in the U.S. No country-specific switches are available.

C prototype

```
int TigerCA(    char inputAddress[194],
                char sentLen[4],
                char errcode[30],
                char firmname[40],
                char urbanization[28],
                char Dline1[64],
                char Dline2[64],
                char LastLine[64],
                char Stringaddress[260],
                char DPC[2],
                char Checkdigit[1],
                char cityname[28],
                char stcode[2],
                char zip[5],
                char  addon[4],
                char  croute[4],
                char LACS[1],
                char LOTsequence[4],
                char LOTcode[1],
                char PMB[12],
                char results[200][194],
                char strnum[10],
                char secname[4],
                char secnum[8],
                char countyname[25],
                char countynum[3],
                char TLID[20][10],
                char tigererrcode[30],
                char tigerstcode[20][2],
                char tigercroute[20][4],
                char tigercounty[20][3],
                char MiscData[20][1],
                char tract[20][6],
                char block[20][4],
                char fLat[20][11],
                char tLat[20][11],
                char fLong[20][12],
                char tLong[20][12],
                char AddonStart[20][4],
                char AddonEnd[20][4],
                char tigerRet[10]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below*)* (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery

line 2 information

(OUTPUT) city, state, ZIP information (municipality, province, postal code for Canada)
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code
Checkdigit
(OUTPUT) Delivery point check digit
cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum
TLID
(OUTPUT) TIGER/Line identification number for use with Census Bureau files
tigererrcode
(OUTPUT) TIGER/Line specific error code (see Appendix H)
tigerstcode
(OUTPUT) FIPS state code
tigercroute

(OUTPUT) TIGER/Line carrier route number
tigercounty
(OUTPUT) FIPS county number
MiscData

tract block fLat tLat fLong tLong

(OUTPUT) Contains CBSA indicator (5 bytes), followed by FIPS congressional district code (7 bytes) ("exact match indicator" in the earlier, ZIP+4 level version of the geocoder)

(OUTPUT) Census tract number (OUTPUT) Census block number

(OUTPUT) Latitude ("from latitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version) (OUTPUT) Longitude

("from longitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version)
AddonStart
(OUTPUT) {not used in the rooftop version} ("ZIP+4 start range" in the earlier, ZIP+4 level version)
AddonEnd tigerRet

(OUTPUT) {not used in the rooftop version}("ZIP+4 end range" in the earlier, ZIP+4 level version) (OUTPUT) Geocoder return code,

equal to the number of Census records returned

Return codes and error codes

See Appendices C and H.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

The results string may contain multiple records, but users must check the *errcode* string to determine the record size and layout, which is different for each country. See Appendix C for information about country-specific error codes. See Appendix F for information about country-specific record formats.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, MySQL, PERL, PHP, PostgreSQL, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

GEOCODE

Description

Accepts 9-digit ZIP code as input.

Returns end-points of latitude and longitude ranges and other geo-coding information based on ZIP+4 area.

C prototype

```
int GeoCode(    char zip4[11],
                char tigererrcode[30],
                char TLID[20][10],
                char tigerstcode[20][2],
                char tigercroute[20][4],
                char tigercounty[20][3],
                char MiscData[20][1],
                char tract[20][6],
                char block[20][4],
                char fLat[20][11],
                char tLat[20][11],
                char fLong[20][12],
                char tLong[20][12],
                char AddonStart[20][4],
                char AddonEnd[20][4]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

zip4
input 9-digit ZIP code

tigererrcode
(OUTPUT) TIGER/Line specific error code (see Appendix H)

TLID
(OUTPUT) TIGER/Line identification number for use with Census Bureau files

tigerstcode
(OUTPUT) FIPS state code

tigercroute
(OUTPUT) TIGER/Line carrier route number

tigercounty
(OUTPUT) FIPS county number

MiscData

tract block fLat tLat fLong tLong

(OUTPUT) Contains CBSA indicator (5 bytes), followed by FIPS congressional district code (7 bytes) ("exact match indicator" in the earlier, ZIP+4 level version of the geocoder)

(OUTPUT) Census tract number (OUTPUT) Census block number

(OUTPUT) Latitude ("from latitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version) (OUTPUT) Longitude

("from longitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version)

AddonStart
(OUTPUT) {not used in the rooftop version} ("ZIP+4 start range" in the earlier, ZIP+4 level version)

AddonEnd
(OUTPUT) {not used in the rooftop version}("ZIP+4 end range" in the earlier, ZIP+4 level version)

Return codes and error codes

Return code is equal to the number of Census records returned.
For error code description, see Appendix H.

Remarks

Examples

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, Oracle, PERL, PostgreSQL, SQL Server (via wrapper), VB.NET

Examples for other languages and environments are available upon request.

GETCENTROID

Description

Accepts ZIP code as input.

Returns latitude and longitude coordinates of ZIP area centroids.

C prototype

```
int getCentroid(    char ZIP[5],
                    char lats[10][11],
                    char longs[10][12]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

ZIP

lats longs

input ZIP code

(OUTPUT) latitude coordinates of ZIP area centroids (OUTPUT) longitude

coordinates of ZIP area centroids

Return codes and error codes

Return code is equal to the number of centroids returned.
Return code -1 indicates corrupt or missing geocoder data.

Remarks

Examples

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C, C#, Java, VB.NET

Examples for other languages and environments are available upon request.

DB2CORRECTA

Description

CorrectA wrapper function for integration with DB2 database systems.

C prototype

```
void db2CorrectA( char inputAddress[194+1],
                  char sentLen[4+1],
                  char errcode[30+1],
                  char firmname[40+1],
                  char urbanization[28+1],
                  char Dline1[64+1],
                  char Dline2[64+1],
                  char LastLine[64+1],
                  char Stringaddress[260+1],
                  char DPC[2+1],
                  char Checkdigit[1+1],
                  char cityname[28+1],
                  char stcode[2+1],
                  char zip[5+1],
                  char addon[4+1],
                  char croute[4+1],
                  char LACS[1+1],
                  char LOTsequence[4+1],
                  char LOTcode[1+1],
                  char PMB[12+1],
                  char results[100+1][194],
                  char strnum[10+1],
                  char secname[4+1],
                  char secnum[8+1],
                  char countyname[25+1],
                  char countynum[3+1],
                  char retcode[10+1]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

See parameter list **CorrectA**.

Return codes and error codes

See Appendix C.

Remarks

Examples

DB2 requires all parameters to be null-terminated. Thus, each function argument is one byte longer than in the **CorrectA** definition.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

DB2

Examples for other languages and environments are available upon request.

DB2TIGERCA

Description

TigerCA wrapper function for integration with DB2 database systems.

C prototype

```
void db2TigerCA(    char inputAddress[194+1],
                    char sentLen[4+1],
                    char errcode[30+1],
                    char firmname[40+1],
                    char urbanization[28+1],
                    char Dline1[64+1],
                    char Dline2[64+1],
                    char LastLine[64+1],
                    char Stringaddress[260+1],
                    char DPC[2+1],
                    char Checkdigit[1+1],
                    char cityname[28+1],
                    char stcode[2+1],
                    char zip[5+1],
                    char addon[4+1],
                    char croute[4+1],
                    char LACS[1+1],
                    char LOTsequence[4+1],
                    char LOTcode[1+1],
                    char PMB[12+1],
                    char results[100+1][194],
                    char strnum[10+1],
                    char secname[4+1],
                    char secnum[8+1],
                    char countyname[25+1],
                    char countynum[3+1],
                    char retcode[10+1],
                    char TLID[20+1][10],
                    char tigererrcode [30+1],
                    char tigerstcode[20+1][2],
                    char tigercroute[20+1][4],
                    char tigercounty[20+1][3],
                    char MiscData[20+1][1],
                    char tract[20+1][6],
                    char block[20+1][4],
                    char fLat[20+1][11],
                    char tLat[20+1][11],
                    char fLong[20+1][12],
                    char tLong[20+1][12],
                    char addonStart[20+1][4],
                    char addonEnd[20+1][4],
                    char tigerret[10+1]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

See parameter list **TigerCA**.

Return codes and error codes

See Appendices C and H.

Remarks

Examples

DB2 requires all parameters to be null-terminated. Thus, each function argument is one byte longer than in the **TigerCA** definition.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

DB2

Examples for other languages and environments are available upon request.

CORRECTACAT

Description

CorrectA wrapper function for integration with environments that prohibit long function argument lists. Excludes Stringaddress parameter from the output.

C prototype

```
int CorrectAcat(    char inputAddress[194],
                    char sentLen[4],
                    char errcode[30],
                    char firmname[40],
                    char urbanization[28],
                    char stringout[39200]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
stringout
(OUTPUT) pipe-delimited output results

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Ruby

Examples for other languages and environments are available upon request.

CORRECTACAT2

Description

CorrectA wrapper function for integration with environments that prohibit long function argument lists. Includes Stringaddress parameter in the output.

C prototype

```
int CorrectAcat2(     char inputAddress[194],
                      char sentLen[4],
                      char errcode[30],
                      char firmname[40],
                      char urbanization[28],
                      char stringout[39460]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
stringout
(OUTPUT) pipe-delimited output results

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development

directory on UNIX\Linux distributions). The following examples are currently available for this function:

Ruby

Examples for other languages and environments are available upon request.

CORRECTACASSORACLE

Description

CorrectACASS wrapper function for integration with Oracle database systems.

C prototype

```
int CorrectACASSOracle(     char inputAddress[194],
                            char sentLen[4],
                            char errcode[30],
                            char firmname[40],
                            char urbanization[28],
                            char Dline1[64],
                            char Dline2[64],
                            char LastLine[64],
                            char Stringaddress[260],
                            char DPC[2],
                            char Checkdigit[1],
                            char cityname[28],
                            char stcode[2],
                            char zip[5],
                            char  addon[4],
                            char  croute[4],
                            char LACS[1],
                            char LOTsequence[4],
                            char LOTcode[1],
                            char PMB[12],
                            char results[100][194],
                            char strnum[10],
                            char secname[4],
                            char secnum[8],
                            char countyname[25],
                            char countynum[3],
                            char path[256]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery line

2 information

(OUTPUT) city, state, ZIP information
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code
Checkdigit
(OUTPUT) Delivery point check digit
cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum
path
(INPUT) location of a text file to store address statistics for generating CASS report

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

CorrectAddress Graphical User Interface (see Chapter 6) is normally used for batch database jobs and can be used to create a facsimile copy of PS Form 3553 on your default printer. PS Form 3553 can be used as proof of valid addresses when approaching the United States Postal Service for a rate discount on bulk mail. In the case that you are integrating *CorrectAddress* into an application and not using the GUI for your batch jobs, the function **CorrectACASS** allows you to keep track of your records and create a text file with a facsimile PS Form 3553 that you can later print out.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Oracle

Examples for other languages and environments are available upon request.

CORRECTACOBOL

Description

CorrectA wrapper function for integration with environments that do not allow integer return codes.

Includes optional flag to return secondary information on delivery line 2.

C prototype

```
void CorrectACobol(char inputAddress[194],
                   char sentLen[4],
                   char errcode[30],
                   char firmname[40],
                   char urbanization[28],
                   char Dline1[64],
                   char Dline2[64],
                   char LastLine[64],
                   char Stringaddress[260],
                   char DPC[2],
                   char Checkdigit[1],
                   char cityname[28],
                   char stcode[2],
                   char zip[5],
                   char  addon[4],
                   char  croute[4],
                   char LACS[1],
                   char LOTsequence[4],
                   char LOTcode[1],
                   char PMB[12],
                   char results[200][194],
                   char strnum[10],
                   char secname[4],
                   char secnum[8],
                   charcountyname[25],
                   char countynum[3],
                   char retcode[10],
                   char dline2Flag[1]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery line

2 information

(OUTPUT) city, state, ZIP information
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code
Checkdigit
(OUTPUT) Delivery point check digit
cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code (OUTPUT) carrier route

code and number

(OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum
retcode dline2flag

(OUTPUT) character representation of the return code

(INPUT) switch to return secondary information on line 2 0 – default
1 – always return secondary information on line 1 2 – always return
secondary information on line 2

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

COBOL (see Lawson Integration later in this chapter), PERL

Examples for other languages and environments are available upon request.

CORRECTAN

Description

CorrectA variant function that allows setting the maximum number of results per address.

C prototype

```
int CorrectAN(      char inputAddress[194],
                    char sentLen[4],
                    char errcode[30],
                    char firmname[40],
                    char urbanization[28],
                    char Dline1[64],
                    char Dline2[64],
                    char LastLine[64],
                    char Stringaddress[260],
                    char DPC[2],
                    char Checkdigit[1],
                    char cityname[28],
                    char stcode[2],
                    char zip[5],
                    char  addon[4],
                    char  croute[4],
                    char LACS[1],
                    char LOTsequence[4],
                    char LOTcode[1],
                    char PMB[12],
                    char results[][194],
                    int rsize;
                    char  strnum[10],
                    char  secname[4],
                    char secnum[8],
                    char countyname[25],
                    char countynum[3]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery line

2 information

(OUTPUT) city, state, ZIP information
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code
Checkdigit
(OUTPUT) Delivery point check digit

cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results rsize strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (INPUT) maximum

number of results to return

(OUTPUT) primary (street) number

(OUTPUT) secondary (unit) designator (OUTPUT) secondary

(unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C

Examples for other languages and environments are available upon request.

CORRECTAORACLE

Description

CorrectAWorld wrapper function for integration with Oracle database systems.

Includes optional flag to return secondary information on delivery line 2.

C prototype

```
int CorrectAOracle( char inputAddress[194],
                    char sentLen[4],
                    char errcode[30],
                    char firmname[40],
                    char urbanization[28],
                    char Dline1[64],
                    char Dline2[64],
                    char LastLine[64],
                    char Stringaddress[260],
                    char DPC[2],
                    char Checkdigit[1],
                    char cityname[28],
                    char stcode[2],
                    char zip[5],
                    char  addon[4],
                    char croute[4],
                    char LACS[1],
                    char LOTsequence[4],
                    char LOTcode[1],
                    char PMB[12],
                    char results[100][194],
                    char strnum[10],
                    char secname[4],
                    char secnum[8],
                    char countyname[25],
                    char countynum[3],
                    char dline2flag[1]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery line

2 information

(OUTPUT) city, state, ZIP information
Stringaddress
(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)
DPC
(OUTPUT) Delivery point code

Checkdigit
(OUTPUT) Delivery point check digit
cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code (OUTPUT) carrier route

code and number

(OUTPUT) LACS code
LOTsequence
(OUTPUT) Line of Travel sequence number
LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number
countyname
(OUTPUT) county name
countynum
(OUTPUT) countynum
dline2flag

(INPUT) switch to return secondary information on line 2 0 – default
1 – always return secondary information on line 1 2 – always return
secondary information on line 2

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Users may specify a country-specific search database by passing the appropriate flag in the *errcode* field:

"Us" – Search only the USPS database (United States and territories)
"Cd" – Search only the Canadian address database

The results string may contain multiple records, but users must check the *errcode* string to determine the record size and layout, which is different for each country. See Appendix C for information about country-specific error codes. See Appendix F for information about country-specific record formats.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Oracle

Examples for other languages and environments are available upon request.

FINDCITYCOUNTYCOBOL

Description

Accepts ZIP code as input.

Returns preferred city name, state, county information.

C prototype

```
void FindCityCounty(    char ZIP[5],
                        char cityname[28],
                        char state[2],
                        char countyname[25],
                        char countynum[3],
                        char retcode[10]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

ZIP

cityname state

input ZIP code

(OUTPUT) city name (OUTPUT) state abbreviation

countyname
(OUTPUT) county name

countynum
(OUTPUT) county number

retcode

Return codes

(OUTPUT) numeric representation of the return code

Remarks

Examples

0 – completed successfully
-2 – no valid license key
-3 – trial expired
-99 – no matches found

All input parameters must be initialized prior to calling the function.

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

COBOL

Examples for other languages and environments are available upon request.

TIGERCACAT

Description

TigerCA wrapper function for integration with environments that prohibit long function argument lists. Excludes Stringaddress parameter from the output.

C prototype

```
int TigerCAcat(     char inputAddress[194],
                    char sentLen[4],
                    char errcode[30],
                    char firmname[40],
                    char urbanization[28],
                    char stringout[40920]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
stringout
(OUTPUT) pipe-delimited output results

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Ruby

Examples for other languages and environments are available upon request.

TIGERCACAT2

Description

TigerCA wrapper function for integration with environments that prohibit long function argument lists. Includes Stringaddress parameter in the output.

C prototype

```
int TigerCAcat2(    char inputAddress[194],
                    char sentLen[4],
                    char errcode[30],
                    char firmname[40],
                    char urbanization[28],
                    char stringout[41920]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below) (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient
urbanization
(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)
stringout
(OUTPUT) pipe-delimited output results

Return codes and error codes

See Appendix C.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a “pipe” character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to “194 “ (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Ruby

Examples for other languages and environments are available upon request.

TIGERCAN

Description

TigerCA variant function that allows setting the maximum number of results per address.

C prototype

```
int TigerCAN(     char inputAddress[194],
                  char sentLen[4],
                  char errcode[30],
                  char firmname[40],
                  char urbanization[28],
                  char Dline1[64],
                  char Dline2[64],
                  char LastLine[64],
                  char Stringaddress[260],
                  char DPC[2],
                  char Checkdigit[1],
                  char cityname[28],
                  char stcode[2],
                  char zip[5],
                  char  addon[4],
                  char  croute[4],
                  char LACS[1],
                  char LOTsequence[4],
                  char LOTcode[1],
                  char PMB[12],
                  char results[][194],
                  int rsize,
                  char  strnum[10],
                  char  secname[4],
                  char secnum[8],
                  char countyname[25],
                  char countynum[3],
                  char TLID[20][10],
                  char tigererrcode[30],
                  char tigerstcode[20][2],
                  char tigercroute[20][4],
                  char tigercounty[20][3],
                  char MiscData[20][1],
                  char tract[20][6],
                  char block[20][4],
                  char fLat[20][11],
                  char tLat[20][11],
                  char fLong[20][12],
                  char tLong[20][12],
                  char AddonStart[20][4],
                  char AddonEnd[20][4],
                  char tigerRet[10]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below*)* (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient

urbanization

(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)

Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery

line 2 information

(OUTPUT) city, state, ZIP information (municipality, province, postal code for Canada)

Stringaddress

(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)

DPC

(OUTPUT) Delivery point code

Checkdigit

(OUTPUT) Delivery point check digit

cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code

LOTsequence

(OUTPUT) Line of Travel sequence number

LOTcode PMB

results rsize strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (INPUT) maximum

number of results to return

(OUTPUT) primary (street) number (OUTPUT) secondary (unit)

designator

(OUTPUT) secondary (unit) number

countyname

(OUTPUT) county name

countynum

(OUTPUT) countynum

TLID

(OUTPUT) TIGER/Line identification number for use with Census Bureau files

tigererrcode

(OUTPUT) TIGER/Line specific error code (see Appendix H)

tigerstcode

(OUTPUT) FIPS state code

tigercroute

(OUTPUT) TIGER/Line carrier route number

tigercounty

(OUTPUT) FIPS county number

MiscData
tract block fLat tLat fLong tLong

(OUTPUT) Contains CBSA indicator (5 bytes), followed by FIPS congressional district code (7 bytes) ("exact match indicator" in the earlier, ZIP+4 level version of the geocoder)

(OUTPUT) Census tract number (OUTPUT) Census block number

(OUTPUT) Latitude ("from latitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version) (OUTPUT) Longitude

("from longitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version)
AddonStart
(OUTPUT) {not used in the rooftop version} ("ZIP+4 start range" in the earlier, ZIP+4 level version)
AddonEnd tigerRet

(OUTPUT) {not used in the rooftop version}("ZIP+4 end range" in the earlier, ZIP+4 level version) (OUTPUT) Geocoder return code,

equal to the number of Census records returned

Return codes and error codes

See Appendices C and H.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

The results string may contain multiple records, but users must check the *errcode* string to determine the record size and layout, which is different for each country. See Appendix C for information about country-specific error codes. See Appendix F for information about country-specific record formats.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

C

Examples for other languages and environments are available upon request.

TIGERCAORACLE

Description

TigerCA wrapper function for integration with Oracle database systems.

C prototype

```
int TigerCAOracle( char inputAddress[194],
                   char sentLen[4],
                   char errcode[30],
                   char firmname[40],
                   char urbanization[28],
                   char Dline1[64],
                   char Dline2[64],
                   char LastLine[64],
                   char Stringaddress[260],
                   char DPC[2],
                   char Checkdigit[1],
                   char cityname[28],
                   char stcode[2],
                   char zip[5],
                   char  addon[4],
                   char  croute[4],
                   char LACS[1],
                   char LOTsequence[4],
                   char LOTcode[1],
                   char PMB[12],
                   char results[100][194],
                   char strnum[10],
                   char secname[4],
                   char secnum[8],
                   char countyname[25],
                   char countynum[3],
                   char TLID[20][10],
                   char tigererrcode[30],
                   char tigerstcode[20][2],
                   char tigercroute[20][4],
                   char tigercounty[20][3],
                   char MiscData[20][1],
                   char tract[20][6],
                   char block[20][4],
                   char fLat[20][11],
                   char tLat[20][11],
                   char fLong[20][12],
                   char tLong[20][12],
                   char AddonStart[20][4],
                   char AddonEnd[20][4],
                   char tigerRet[10]);
```

Parameters *(for descriptions of postal terms and abbreviations, refer to Appendix B)*

inputAddress
input address, formatted according to postal specifications (see *Remarks* below)
sentLen errcode firmname

input address length (see *Remarks* below*)* (OUTPUT) error codes

(INPUT/OUTPUT) firm name / recipient

urbanization

(INPUT/OUTPUT) urbanization name (Puerto Rico addresses only)

Dline1 Dline2 LastLine

(OUTPUT) delivery line 1 information (OUTPUT) delivery

line 2 information

(OUTPUT) city, state, ZIP information (municipality, province, postal code for Canada)

Stringaddress

(OUTPUT) buffer to store DPV, LACS, SuiteLink, RDI indicators, and special purpose data (see Appendices I-M)

DPC

(OUTPUT) Delivery point code

Checkdigit

(OUTPUT) Delivery point check digit

cityname stcode zip addon croute LACS

(OUTPUT) City name (OUTPUT) State abbreviation

(OUTPUT) 5-digit ZIP code

(OUTPUT) +4 extension for the ZIP code

(OUTPUT) carrier route code and number (OUTPUT) LACS code

LOTsequence

(OUTPUT) Line of Travel sequence number

LOTcode PMB

results strnum secname secnum

(OUTPUT) Line of travel sequence code (OUTPUT) Private mail

box number

(OUTPUT) buffer containing postal records (see Appendix F for layout details) (OUTPUT) primary

(street) number

(OUTPUT) secondary (unit) designator

(OUTPUT) secondary (unit) number

countyname

(OUTPUT) county name

countynum

(OUTPUT) countynum

TLID

(OUTPUT) TIGER/Line identification number for use with Census Bureau files

tigererrcode

(OUTPUT) TIGER/Line specific error code (see Appendix H)

tigerstcode

(OUTPUT) FIPS state code

tigercroute

(OUTPUT) TIGER/Line carrier route number

tigercounty

(OUTPUT) FIPS county number

MiscData

tract block fLat tLat fLong tLong

(OUTPUT) Contains CBSA indicator (5 bytes) , followed by FIPS congressional district code (7 bytes) ("exact match indicator" in the earlier, ZIP+4 level version of the geocoder)

(OUTPUT) Census tract number (OUTPUT) Census block number

(OUTPUT) Latitude ("from latitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version) (OUTPUT) Longitude

("from longitude" in the earlier, ZIP+4 level version)

(OUTPUT) {not used in the rooftop version} ("to latitude" in the earlier, ZIP+4 level version)
AddonStart
(OUTPUT) {not used in the rooftop version} ("ZIP+4 start range" in the earlier, ZIP+4 level version)
AddonEnd tigerRet

(OUTPUT) {not used in the rooftop version}("ZIP+4 end range" in the earlier, ZIP+4 level version) (OUTPUT) Geocoder return code,

equal to the number of Census records returned

Return codes and error codes

See Appendices C and H.

Remarks

All input parameters must be initialized prior to calling the function.

Inputaddress parameter contains three sections of information, formatted as follows:

DELIVERY LINE 1 (64 bytes) | DELIVERY LINE 2 (64 bytes) | LAST LINE (64 bytes)

Each section is separated with a "pipe" character (|), and the remainder of each section must be padded with blanks up to the section length. Input delivery line 2 is intended for dual address information.

sentLen parameter must be set to "194 " (with a space) on all calls to this function.

By default, output address is returned in capital letters. Mixed case output switch and other custom options are described in Appendix I.

The results string may contain multiple records, but users must check the *errcode* string to determine the record size and layout, which is different for each country. See Appendix C for information about country-specific error codes. See Appendix F for information about country-specific record formats.

Examples

Code samples can be found in the \Development Kits folder in the *CorrectAddress* installation directory (\Development directory on UNIX\Linux distributions). The following examples are currently available for this function:

Oracle

Examples for other languages and environments are available upon request.

CALLING CORRECTADDRESS FROM C

Examples of calling *CorrectAddress* functions from C programs are located in the \Development Kits\C directory under *CorrectAddress* installation. In addition, C prototypes for all supported API functions are provided earlier in this chapter.

CALLING CORRECTADDRESS FROM .NET

CALLING CORRECTADDRESS FROM C#

Examples of calling *CorrectAddress* functions from C# programs are located in the \Development Kits\C# directory under *CorrectAddress* installation.

CALLING CORRECTADDRESS FROM VB. NET

Examples of calling *CorrectAddress* functions from VB.NET programs are located in the \Development Kits\VB.NET directory under *CorrectAddress* installation.

CALLING CORRECTADDRESS FROM JAVA

Examples of calling *CorrectAddress* functions from Java programs are located in the \Development Kits\Java directory under *CorrectAddress* installation.

CorrectAddress comes with its own Java class (**javaCANativeDispatcher)** which provides access to the main *CorrectAddress* API functions. This Java code can run on any platform due to its use of the Java Native Interface (JNI). Documentation for the use of the class can be found in the \Development Kits\Java **\Javadoc** directory.

To incorporate this class into a package, you must use **"correcta"** as the package name.

CALLING CORRECTADDRESS FROM PERL

WINDOWS SYSTEMS

No specialized wrappers are required to access *CorrectAddress* functions from PERL scripts on Windows. All calls are made directly to **CorrectA.dll** library. A number of PERL demo files are available in your local **\Development Kits\PERL\Windows** directory in the default *CorrectAddress* installation.

UNIX/ L INUX SYSTEMS

To call *CorrectAddress* functions from a PERL script, you must use a specialized shared object (**CAPerl.so**). To create this file, follow the instructions below.

1. Build a standard *CorrectAddress* shared object (**libCorrectA.so**) as described in SHARED OBJECTS section. At the end of the process, when prompted to delete the object files, select NO.
2. Locate the file **CAPerl_wrap.c** in the *CorrectAddress* PERL directory (**/PERL/<version>**) on your installation. Place this file in the directory where **libCorrectA.so** was built (all **.o** files must still be there).
3. Use the following command to compile **CAPerl_wrap.c:**

   ```
   $CC –O –c –I/$PERL_INCLUDE_DIRECTORY CAPerl_wrap.c
   ```

   where $CC is your default compiler and **$PERL_INCLUDE_DIRECTORY** is the path to the directory containing **perl.h, EXTERN.h**, and **XSUB.h** files. To find out where the directory is loaded, execute:

   ```
   perl -e 'use Config; print $Config{archlib};'
   ```

   Wrapper compile example:

   ```
   gcc –O –c –fPIC –I/usr/lib/perl5/5.8.0/i386-linux-thread-multi/CORE

   CAPerl_wrap.c
   ```

4. Create the shared object by issuing the command:

   ```
   ld –G –o CAPerl.so *.o
   ```

5. Copy the files **CAPerl.pm** and **test.pl** into the directory containing your shared object. These files can be found in your *CorrectAddress* **PERL/<version>** directory. The **test.pl** file contains a demonstration of calling the **CorrectA** and **FindCityCounty** functions from the shared object.

CALLING CORRECTADDRESS FROM RUBY

Ruby is the interpreted scripting language for quick and easy object-oriented programming. It has many features to process text files and to do system management tasks (as in Perl). It is simple, straight-forward, extensible, and portable.

RUBY WRAPPERS FOR CO RRECTADDRESS

1. The “DL” LIBRARY

   Ruby comes with a Standard Library called “DL”. This library provides interfaces to the underlying operating system’s dynamic loading capabilities. Using this library, Ruby code can be written to interface with functions in .dll files on Windows. It can also be used to load shared objects (.so files) on UNIX.

2. Description

   The DL library will be used to load the **libCorrectA.so** file. Wrappers will be created for the following three functions:

   - FindCityCounty
   - FindZipCity
   - CorrectAcat

   These wrapper functions will be part of a class called "CorrectAddress". The details of these functions are shown below:

   a. **def cityCounty(zip)**

This function calls the **FindCityCounty** function in the shared object. It takes the Zip Code as Input and prints out the **City**, **State**, **County Name**, and **County Number**. A sample output is shown below:

```
=====================================================
Input Zip: 10509
=====================================================
City: BREWSTER
State: NY
County Name: PUTNAM
County Number: 079
```

b. def zipCity(city, state)

This function calls the **FindCityCity** function in the shared object. It takes the name of the City and the State as Input and prints out the name of the **New City** and the Zip codes associated with that city. A sample output is shown below:

```
  =====================================================
Input City - State: Boston - MA
=====================================================
New City: BOSTON
Zip: 021080210902110021110211202113021140211502116021170211802l
190212002121021220212302124021250212602127021280212902130021310
213202133021340213502136021370216302196021990220102203022040220
502206022070221002211022120221502216022170222202228022410226602
283022840229302295022970229 8
```

c. def correctA(address)

This function calls the **CorrectAcat** function in the shared object. It takes the address as Input and prints out the results as shown below:

The "CorrectA" function is normally used for this. However, due to some limitations either in Ruby or in the "DL" library, a function cannot be called which takes more than 15 arguments. Hence, the "CorrectAcat" function is used, which takes 6 arguments instead of 26 arguments in the case of "CorrectA".

The *CorrectAddress* class has a Constructor and the three wrapper functions mentioned above.

3. Constructor

The code for the Constructor is shown below:

```
def initialize(path)
    @ca = DL.dlopen(path)
  end
```

The Constructor takes the Path of the **libCorrectA.so** file as an argument. It uses the dlopen method to load the shared object. The handle to the opened shared object is stored in the instance variable (@ca) as shown.

```
def zipCity(city, state)
   zc = @ca["FindZipCity", "ISsSs"]
   @newcity = DL.malloc(28 * DL.sizeof("C"))
   @zip = DL.malloc(1000 * DL.sizeof("C"))
   ret_val, rs = zc.call(city, @newcity, state, @zip)
   puts "\n\n================================="
   puts "Input City - State: #{city} - #{state}"
   puts "================================="
   puts "New City: #{rs[1]}"
   puts "Zip: #{rs[3].rstrip}"
   DL::FREE
 end
```

The details of the code are as follows:

a. **zc = @ca["FindZipCity", "ISsSs"]**

In this line, the @ca object created in the Constructor is used to get a handle to the required function. The first argument is the name of the function, **FindZipCity**. The second argument is the method signature. The first "I" stands for the return type, which is an Integer. The next four characters are the arguments needed to call the function. "S" stands for a Character String and "s" stands for a Mutable Character String. This is an equivalent way of writing:

int FindZipCity(const char*, char*, const char*, char*)

This function returns, what is essentially, a function pointer to the **FindZipCity** function.

b. **@newcity = DL.malloc(28 *DL.sizeof("C"))**

@zip = DL.malloc(1000 * DL.sizeof("C"))

The above two lines allocate memory for the variables in which the results will be stored. The "C" argument in DL.sizeof indicates a "char".

c. **ret_val, rs = zc.call(city, @newcity, state, @zip)**

The above line makes the actual function call by passing in the needed arguments. It returns two things – the return value and the result set. The result set is an Array having the four parameters passed to the function.

d. **puts "\n\n================================="**

puts "Input City - State: #{city} - #{state}"

puts "================================="

puts "New City: #{rs[1]}"

puts "Zip: #{rs[3].rstrip}"

The above lines are used for printing the Input passed to the function along with the Output.

e. **DL::FREE**

This line makes a call to the Garbage Collection Routines and frees the allocated memory.

CALLING CORRECTADDRESS FROM PHP

WINDOWS SYSTEMS

CorrectAddress comes with a PHP development kit. It includes an example of a PHP page that references functions exported by specialized wrapper library CorrectAPHP.dll. The library itself is located in the **C:\Program Files\Intelligent Search Technology\CorrectAddress\Development Kits\PHP\Windows** directory. In order to load *CorrectAddress* functions from a PHP application, you should make the following modifications to your **php.ini** file:

Add an appropriate extension entry: **extension=CorrectAPHP.dll**

Specify path to loadable extensions: **extension_dir = "<directory_of_wrapper_dll>"**

UNIX/ L INUX SYSTEMS

1. To access *CorrectAddress* from PHP on a Linux/UNIX based system, we must first create *CorrectAddress* object files. Follow the process described in SHARED OBJECTS section to create **libCorrectA** object. When prompted to delete the object files at the end of the process, select NO.
2. After creating the object files, you must copy the **CAphp.c** file into the directory with all your other source code. This file can be found in your default **istCorrectAddress\PHP** directory where you initially installed the software. Next, compile the file to create the **CAphp** object file using the following command:

   ```
   cc -fpic -DCOMPILE_DL=1 -I/usr/local/include -I/usr/include/php -
   I/usr/include/php/Zend -I/usr/include/php/main -I/usr/include/php/TSRM -O -c
   CAphp.c
   ```

   The –I option flags used in the above compile command denote necessary include directories that the compiler will need in order to create the object file. These directories are **Zend**, **php**, **main**, and **TSRM**. These directories will only be present if you have already installed the PHP developer's package on your system.
3. Next, we must link the object file created with the other *CorrectAddress* objects created in step 1. To link the files into **ca_module.so** use: **cc –shared –L/usr/local/lib –rdynamic –o ca_module.so *.o.** This will create **ca_module.so** which can then be called from a PHP page.
4. In order for PHP to be able to find the shared object, it must reside in your PHP's library path. The PHP library directory by default should be in **/usr/lib** and its name should be php and the version number. For instance, on a machine with PHP 4 installed the directory would be **/usr/lib/php4**.
5. Any PHP web page that you will use to call *CorrectAddress* must be located in a directory that Apache will recognize, for instance **/var/www/html**. To test **ca_module.so**, copy the **CAdemo.php** test file located in your default **istCorrectAddress\PHP** directory into **/var/www/html**.
6. If everything is set up correctly, you will be able to view *CorrectAddress* demo by browsing to the URL: **http://localhost/CAdemo.php** The PHP code to make calls to our **CorrectA** and **FindCityCounty** functions are accessible in **CAdemo.php**.

INTERFACING CORRECTADDRESS VIA LAWSON

Calls to *CorrectAddress* libraries can be integrated into Lawson Software's enterprise resource planning environment. The following procedure must be followed in order to invoke address validation routines from COBOL and Java under Lawson ERP.

COBOL

Installing user must have the authority to run C and COBOL compiler commands.

1. Extract contents of the source archive into an empty directory on your disk (source archive is the `source_vXX.zip` file, where 'XX' stands for *CorrectAddress* version number). This file is located in the root catalog of the installation disk.
2. Compile testCA.c file by running the following command (the file is provided by Experian upon request):

   ```
   cc -O -bmaxdata:0x11E1A300 -c testCA.c
   ```

3. Make sure the command generates object file (testCA.o).
4. Run the build utility:

   ```
   java –jar BuildLib.jar
   ```

5. At the wizard prompt for 'link command', type.

   ```
   cob -zo libCorrectA.so *.o testCA.o -e loadshObj
   ```

If you had previously built the object and retained configuration file (conf.txt), you may instead run 'java –jar BuildLibAuto.jar'

This will create **libCorrectA.so** file ready to be called from Lawson COBOL programs.

6. Copy **libCorrectA.so** into the $GENDIR/lib/shared directory on the Lawson system.
7. To load *CorrectAddress* library dynamically, run: (e.g., `CALL "/app/lawson9/gen/lib/shared/libCorrectA.so"`)

   ```
   CALL "{location_of_the_object}/libCorrectA.so"
   ```

There are currently two procedures available for address validation from COBOL.

CorrectACobol

All parameter sizes are the same as in the **CorrectA** function call:

This procedure performs CASS-processing, standardization, and validation of input addresses.

Input parameters:

```
INPUT-ADDRESS
       DLINE1IN            PIC X(64)
       DELIM1              PIC X(01) VALUE "|".
       DLINE2IN            PIC X(64)
       DELIM2              PIC X(01) VALUE "|".
       LASTLINEIN          PIC X(64)
SENTLEN                 PIC X(4) VALUE "194 "
FIRMNAME                PIC X(40)
URBANIZATION            PIC X(28)
DLINE2FLAG              PIC X(01) VALUE "0"
```

Call example:

```
CALL "CorrectACobol"
USING             INPUT-ADDRESS
                     SENTLEN
                     ERRCODE
                     FIRMNAME
                     URBANIZATION
                     DLINE1
                     DLINE2
                     LASTLINE
                     STRINGADDRESS
                     DPC
                     CHECKDIGIT
                     CITYNAME
                     STCODE
                     ZIP
                     ADDON
                     CROUTE
                     LACS
                     LOTSEQUENCE
                     LOTCODE
                     PMB
                     RESULTS-ARRAY
                     STRNUM
                     SECNAME
                     SECNUM
                     COUNTYNAME
                     COUNTYNUM
                     RET-CODE
                     DLINE2FLAG.
```

FindCityCountyCobol

All parameter sizes are the same as in **FindCityCounty** function:

This procedure retrieves city/state/county information for a particular ZIP code:

Input parameter:

```
ZIP                          PIC X(05)
```

Call example:

```
CALL "FindCityCountyCobol"
USING ZIP,
       CITYNAME,
       STCODE,
       COUNTYNAME,
       COUNTYNUM,
       RET-CODE
```

JAVA

You may use a standard shared object (.so) from the installation disk, or a library previously compiled for COBOL. For the Java development kit, please refer to the /Java directory in the root catalog of the installation disk.

The object linked with the COBOL linker ('cob') in the COBOL section in this document may not be callable from Java. To make a Java-enabled object, use the standard C linker command ('ld', 'cc' etc) to create .so out of object code.

Example (AIX):

Execute the following commands **after** you build shared object for use with COBOL (backup or rename COBOL .so before proceeding). Make sure that all the object files are still in the directory.

```
rm testCA.o

cc -o libCorrectA.so *.o -bE:CorrectA.exp -bM:SRE -bnoentry
```

This will build a shared object for use with Java.

Refer to *Installation* on *Page 2-1* for instructions on installing the product and applying monthly postal data updates.

If you require COBOL integration on UNIX/Linux platforms, follow the source archive approach in the *UNIX/Linux Installation section(for the first three steps only).* Replace step 4 in that procedure with COBOL instructions above.

INTERFACING CORRECTADDRESS VIA ORACLE

CorrectAddress functions can be integrated into Oracle applications. Oracle provides the capability of calling external functions from within PL/SQL. This is accomplished through the creation of extended stored procedures that reference the dynamically executable function code. An external procedure is a third-generation-language routine stored in a dynamic link library, registered with PL/SQL, and called by you to do special purpose processing. *CorrectAddress* routines can be accessed via **CorrectA.dll** (Windows) or **libCorrectA.so** (UNIX/Linux). At run time, PL/SQL loads the library dynamically, and then calls the routine as if it were a PL/SQL subprogram. To safeguard your database, the routine runs in a separate address space.

Development kit showing how to register and execute *CorrectAddress* functions using Oracle remote procedure calls is located under **\Development Kits\Oracle** in your default *CorrectAddress* directory.

Outlined below is the basic process of calling *CorrectAddress* functions:

- Create alias library (**CREATE LIBRARY…**)
- Register *CorrectAddress* function (**CREATE OR REPLACE FUNCTION…**)

  CorrectAOracle example in the development kit creates a function called pCorrectA that calls address correcting routine **CorrectAOracle** (an Oracle-specific version of **CorrectA**). After the successful creation of the pCorrectA function, you will be able to create PL/SQL scripts, triggers, procedures and other functions that call **CorrectA**.
- Create procedure to invoke external function (**CREATE OR REPLACE PROCEDURE…**)
- Enable server output (**SET SERVEROUTPUT ON**)
- Execute procedure (**EXECUTE …**)

INTERFACING CORRECTADDRESS VIA MICROSOFT SQL SERVER

CorrectAddress functions can be executed from within Microsoft SQL Server environment by means of extended stored procedures or CLR assemblies (SQL Server 2005 and higher). The latter approach uses .NET languages (see .NET integration section earlier in this chapter).

To call *CorrectAddress* functions using extended stored procedures requires a specialized wrapper library (**xpCorrectA.dll**). The development kit, along with this library, is located under **\Development Kits\SQL Server** in your default *CorrectAddress* directory.

Outlined below is the process of registering SQL Server extended stored procedures.

- Copy xpCorrectA.dll from your **CorrectAddress\Development Kits\SQL Server** directory to your SQL Server's \Binn directory.

- Open Microsoft SQL Server Enterprise Manager

- Connect to the server that you will be using

- Open DATABASES folder and go to the System database 'master'

- (SQL2000) Right-click on EXTENDED STORED PROCEDURES

- (SQL2005) Right click on PROGRAMMABILITY->EXTENDED STORED PROCEDURES

- Click on NEW EXTENDED STORED PROCEDURE

- In the 'Name' field, type the name of the exported functions (e.g., "xp_CorrectA")

- In the 'Path' field, type xpCorrectA.dll

You can also perform the steps above programmatically by executing the following SQL code:

sp_addextendedproc '{exported_function_name}', 'c:\Program Files\Microsoft SQL Server\MSSQL\Binn\xpCorrectA.dll'

Below is the list of available exported functions (see definitions for highlighted functions at the beginning of the chapter):

1. xp_CorrectA - wrapper for **CorrectA** excluding Stringaddress parameter
2. xp_CorrectACASS - wrapper for **CorrectACASS**
3. xp_CorrectASA - wrapper for **CorrectA** including Stringaddress parameter
4. xp_CorrectAWorld - wrapper for **CorrectAWorld**
5. xp_capconv - wrapper for **capconv**
6. xp_FindCityCounty - wrapper for **FindCityCounty**
7. xp_FindZipCity - wrapper for **FindZipCity**
8. xp_GetBuildDate - wrapper for **GetBuildDate**
9. xp_ParseAddress - wrapper for **ParseAddress**
10. xp_PrintPSForm3553 - wrapper for **PrintPSForm3553**
11. xp_TigerCA - wrapper for **TigerCA** excluding Stringaddress parameter
12. xp_TigerCASA - wrapper for **TigerCA** including Stringaddress parameter
13. xp_UnloadC - procedure to unload **xpCorrectA.dll** from memory

INTERFACING CORRECTADDRESS VIA MYSQL

CorrectAddress functions can be called from within the MySQL environment via user-defined functions, or UDFs. A user-defined function (UDF) is a way to extend MySQL with a new function that works like native (built in) MySQL functions. User-defined functions can be passed a number of arguments (character strings), and return a value. They must be written in C or C++.

Creating User-Defined Functions (UDFs)

Calling conventions for user-defined functions in the MySQL environment:

char *xxx(UDF_INIT *initid, UDF_ARGS *args,char *result, unsigned long *length,char *is_null, char *error);

Wrappers demonstrating how to call some of the *CorrectAddress* functions from within MySQL are available in the **\Development Kits\MySQL** folder.

Building a Shared Object

To build a shared object on a Linux/UNIX platform, copy the file **CorrectA_udf.cc** to the location of your main *CorrectAddress* object files and compile it using the command:

gcc –O –c CorrectA_udf.cc

This will create **CorrectA_udf.o**, follow the instructions in the SHARED OBJECTS chapter to create the main object files for *CorrectAddress* and then link all the object files using the command:

ld –G –o libCorrectA.so *.o

This command will link the object files into a shared object.

You should then copy the object into a directory where it will be recognized by an OS (such as **/usr/lib**).

Executing User-Defined Functions

Calling **CorrectASQL** from MySQL:

To execute user-defined function, first launch MySQL by typing **mysql** at the terminal window prompt. You will see the following (or similar) message displayed.

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 3.23.41

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

To call user-defined function **CorrectASQL**, type the following at the **mysql> prompt** :

create function CorrectASQL returns string soname "libCorrectA.so";

After the function is created successfully, you can call it from the mysql> prompt as follows:

```
set @ca1 = space(40000)

set @ca1 = CorrectASQL(' ', ' ', '445 Hamilton Ave Ste 608', ' ','White
Plains NY 10601',space(40000));
```

The first parameter holds the firm name of the address. This is an optional parameter. Next comes the urbanization name of an address, this is optional and only applies to addresses in Puerto Rico. The third parameter contains primary delivery line information and is required. The fourth parameter contains secondary delivery line information and is used only for dual address cases and apartment abbreviation/numbers. The fifth parameter holds the city, state, and ZIP code for an address. The sixth parameter is purely used to allocate memory for the internal workspace of the function; this parameter should always be padded with 40000 blanks. This function will save the entire results of **CorrectA** into the **@ca1** variable. To parse the information into a more user friendly format use the function parseCA.

Create **parseCA** as follows:

create function parseCA returns string soname "libCorrectA.so";

Call the function by passing it two parameters, the variable containing a results set from a previous call to CorrectASQL and a character string containing a number representing what field you want to parse out. Example given is for parsing out the return code:

Example from MySQL:

```
mysql> select parseCA(@ca1,'0');
+--------------------------------+
| parseCA(@ca1,'0') |
+--------------------------------+
| 1 |
1 row in set (0.03 sec)

mysql> exit

Bye
```

Below are all the possible character string values and the fields they parse:

Field	Returned Parameter
Return Code	'0'
Error Codes	'1'
Firm Name	'2'
Urbanization	'3'
Delivery Line 1	'4'
Delivery Line 2	'5'
Last Line(City/State/ZIP)	'6'
Delivery Point Code	'7'
Checkdigit	'8'
City Name	'9'
State Abbreviation	'10'
ZIP Code(5 digit)	'11'
ZIP Add-on(4 digit)	'12'
Carrier Route	'13'
LACS Indicator	'14'
LOT Sequence	'15'
LOT Code	'16'
PMB Designation	'17'
Result Record	'18,n' where n is equal to the record number to be returned. Valid values are between 0and 200.
Street Number	'19'
Secondary Abbreviation	'20'
Secondary Number	'21'
County Name	'22'
County Number	'23'
ZIP + 4	'24'
LOT Number	'25'

Calling **FindCityCountySQL** from MySQL:

To create a user-defined function **FindCityCounty**SQL, launch MySQL, and at the prompt type:

create function FindCityCountySQL returns string soname "libCorrectA.so";

To call **FindCityCounty**SQL, type the following at the mysql> prompt :

set @ca2 = space(60)

set @ca2 = FindCityCountySQL('10509');

The parameter taken is a 5-digit zip code. This function returns the preferred city name, state abbreviation, county name, and FIPS county number to the variable **@ca2**. To view the information within **@ca2**, use the function parseCityCounty.

To create the UDF parseCityCounty:

create function parseCityCounty returns string soname "libCorrectA.so";

To call parseCityCounty, type the following at the mysql> prompt:

select parseCityCounty(@ca2,'0');

```
mysql> select parseCityCounty(@ca2,'0');
+--------------------------------+
| parseCA(@ca1,'0') |
+--------------------------------+
| Brewster |
+--------------------------------+
1 row in set (0.03 sec)

mysql> exit

Bye
```

The function parseCityCounty takes 2 parameters, the variable returned from a previous call to **FindCityCountySQL** and a character string representing the field to be returned. The table below describes the valid values for parameter 2:

Field Value	Parameter Number
City Name	'0'
State Abbreviation	'1'
County Name	'2'
FIPS County Number	'3'

INTERFACING CORRECTADDRESS VIA POSTGRESQL

CorrectAddress functionality can be accessed from PostgreSQL environment via user-defined C language functions, embedded into the specialized *CorrectAddress* shared object (**libCorrectAPG.so**). The first time a user-defined function in a shared object file is called in a session, the dynamic loader loads that object file into memory so that the function can be called.

1. To access *CorrectAddress* procedures from PostgreSQL on a Linux/UNIX-based system, we must first create *CorrectAddress* object files. Follow the process described in SHARED OBJECTS section to create **libCorrectA.so** object. When prompted to delete the object files at the end of the process, select NO.
2. After creating the object files, copy the **CApgproc.c** file from **/Development/PostgreSQL** directory on your *CorrectAddress* installation disk into the directory with all the object code. Next, compile the file to create the **CApgproc.o** using the following command:

```
$CC -c CApgproc.c –I<local_pgsql_path>/include -
I<local_pgsql_path>/include/server
```

```
(e.g.,gcc-cCApgproc.c-I/usr/local/pgsql/include-
I/usr/local/pgsql/include/server)
```

   The **–I option** flags used in the above compile command denote necessary include directories that the compiler will need in order to create the object file. These directories will only be present if you have PostgreSQL installed on your system.

3. Next, we must link the object file created with the other *CorrectAddress* objects created in step 1. To link the files into **libCorrectAPG.so** use the following command:

```
$CC -shared -o libCorrectAPG.so *.o
```

4. Once the object is created, you can start using *CorrectAddress* functions in your PostgreSQL code.

   Example of using **CorrectA**() procedure as a user-defined function.

```
CREATE FUNCTION PG_CorrectA(character, character,
character,character,character) RETURNS CHARACTER AS
'<path>/libCorrectAPG.so', 'PG_CorrectA' LANGUAGE C STRICT;

SELECT PG_CorrectA ('445 Hamilton Ave Ste 608','','White Planes
NY','','');

DROP FUNCTION PG_CorrectA(character, character,
character,character,character);
```

Description of arguments:

ARG 1 – Delivery Line 1

ARG 2 – Delivery Line 2

ARG 3 – Last Line

ARG 4 – Error code (reserved for special processing)

ARG 5 – Stringaddress (reserved for special processing)

Output record layout:

FIELD NAME	POSITION	LENGTH
RETURN CODE	1	4
ERROR CODE	5	30
FIRM NAME	35	40
URBANIZATION	75	28
DELIVERY LINE 1	103	64
DELIVERY LINE 2	167	64
LAST LINE	231	64
DPC	295	2
CHECKDIGIT	297	1
STATE CODE	298	2
ZIP	300	5
ADDON	305	4
CARRIER ROUTE	309	4
LACS	313	1
LOT SEQUENCE	314	4
LOT CODE	318	1
PMB	319	12
STREET NUMBER	331	10
SECONDARY NAME	341	4
SECONDARY NUMBER	345	8
COUNTY NAME	353	25
COUNTY NUMBER	378	3
STRINGADDRESS	381	260
RESULTS	641	50*194

Example of using **FindCityCity()** procedure as a user-defined function:

```
CREATE FUNCTION PG_FindZipCity(character, character) RETURNS
CHARACTER AS '<path>/libCorrectAPG.so', 'PG_FindZipCity'
LANGUAGE C STRICT;

SELECT PG_FindZipCity ('White Planes','NY');

DROP FUNCTION PG_FindZipCity(character, character);
```

Description of arguments:

ARG 1 – City

ARG 2 – State

Output record layout:

FIELD NAME	POSITION	LENGTH
RETURN CODE	1	4
NEW CITY NAME	5	28
ZIP CODES	33	1000

INTERFACING CORRECTADDRESS VIA DB2

CorrectAddress libraries ship with specialized wrapper functions which allow them to be invoked from DB2 scripts by means of stored procedures. Below is an example of how address validation methods can be accessed directly from DB2.

Wrappers demonstrating how to call some of the *CorrectAddress* functions from within DB2 are available in the **\Development Kits\DB2** folder.

Example of registering **db2CorrectA** stored procedure.

For this example, statement terminator character has been set to @.

```
CREATE PROCEDURE db2CorrectA(IN inputAddress varchar(255),IN sentLen
char(4),INOUT errcode char(30),INOUT firmname char(40),INOUT urbanization
char(28),OUT dline1 char(64), OUT dline2 char(64), OUT lastline char(64), OUT
stringaddress varchar(260), OUT DPC char(2), OUT checkdigit char(1), OUT
cityname char(28), OUT stcode char(2), OUT zip char(5), OUT addon char(4), OUT
croute char(4), OUT LACS char(1), OUT LOTsequence char(4), OUT
LOTcode char(1), OUT PMB char(12), OUT results varchar(19900), OUT strnum
char(10), OUT secname char(4), OUT secnum char(8), OUT countyname char(25), OUT
countynum char(3), OUT retcode char(10))
DYNAMIC RESULT SETS 0
LANGUAGE C
PARAMETER STYLE GENERAL
NO DBINFO
FENCED
NOT DETERMINISTIC
PROGRAM TYPE SUB
EXTERNAL NAME 'CorrectA!db2CorrectA' @
```

Code samples in the **\Development Kits\DB2** folder demonstrate how to register a sample stored procedure **CallCorrectA** which accepts three lines of address information and makes a call to **db2CorrectA.** The results can then be inserted into the output table.

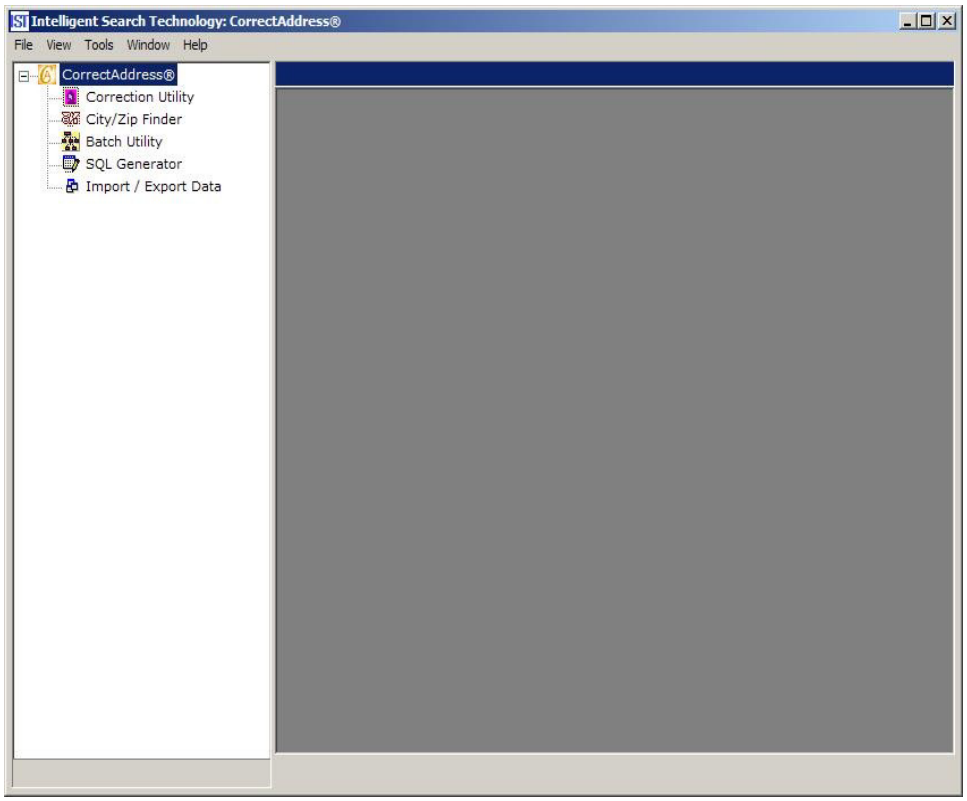
Running stored procedure **CallCorrectA**

```
call CallCorrectA('445 Hamilton Ave','Ste 608','10601') @
```

The result of this execution will be the validated and standardized address and all of its supplemental fields. For complete field descriptions, refer to **CorrectA** function specification at the beginning of this chapter.

Chapter6- CorrectAddress Graphical User Interface (GUI)

This section will detail the various uses of the *CorrectAddress* Graphical User Interface, included with the Windows version of the product. If you installed the GUI with the default settings, then select **Start→Programs→Intelligent Search Technology→CorrectAddress.** The following will be displayed:

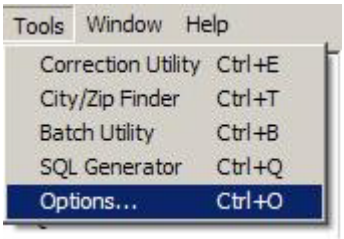


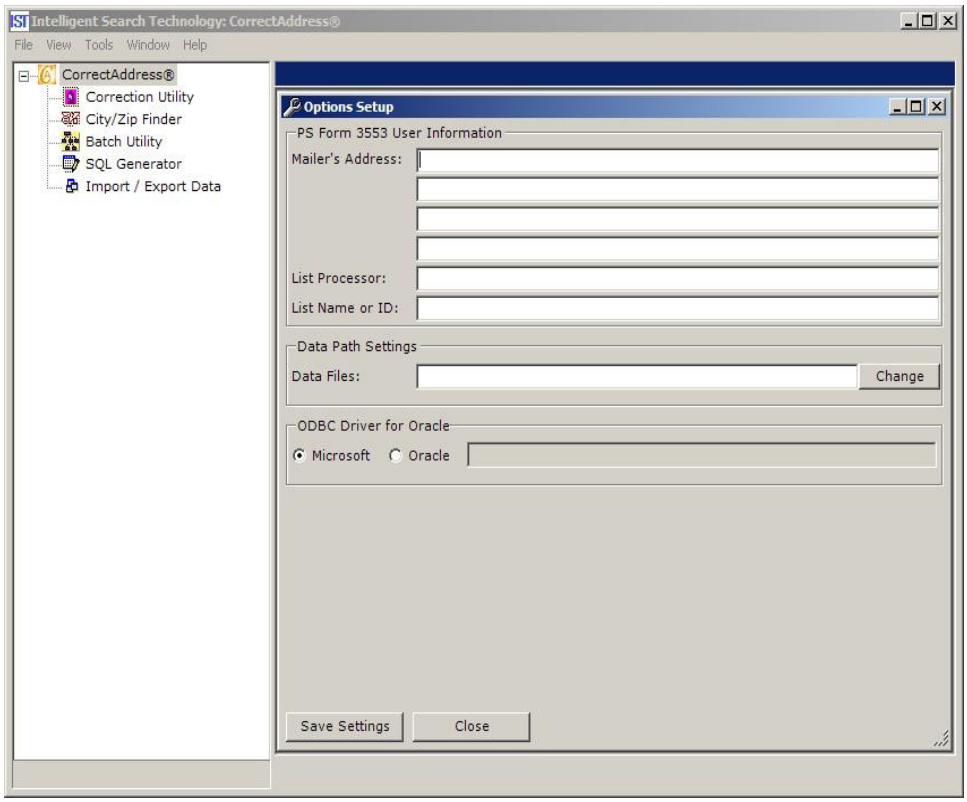
From this window, you can access all the components of the *CorrectAddress* product.

CHANGING YOUR SETUP INFORMATION

Default setup information was recorded when you initially installed the *CorrectAddress* product, most importantly the location of the data files on your system. If the need ever arises where you have to move your data files or wish to change your user or printer information, access the menu by selecting **Tools →Options Setup**.

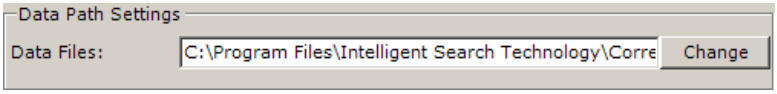
Opening the **Options Setup** menu will display this window where you can change your user information that is used when executing the *CorrectAddress* engine.





The first frame in the window, entitled PS Form 3553 User Information, contains identification information that is required for the generation of Form 3553. Form 3553 is a CASS Summary Report that must be included when applying for any sort of postal discount from the United States Postal Service. Our batch processor can create its own facsimile of this form and will input the information here into it before printing it out. You may also write your own summary report by using the **E3553.pdf** file that is included with this product. The **E3553.pdf** file should reside in your [**CorrectAddress Home]** directory (**Program Files\Intelligent Search Technology\CorrectAddress** by default). This PDF file required Adobe Acrobat to read and is editable; simply place the correct values for the fields into the form and print it. More information on what the fields of PS Form 3553 mean is given in *PS Form 3553 on Page A-1*.

The frame entitled Data File Paths contains all the necessary path information that is contained in your **CorrectA.ini** file. This file was created when *CorrectAddress* was installed. If the location of the data files ever changes, you can input the new paths here by pressing the change button next to the path you wish to change. For instance, pressing the Change button next to the State Files path results in the following window being brought up:

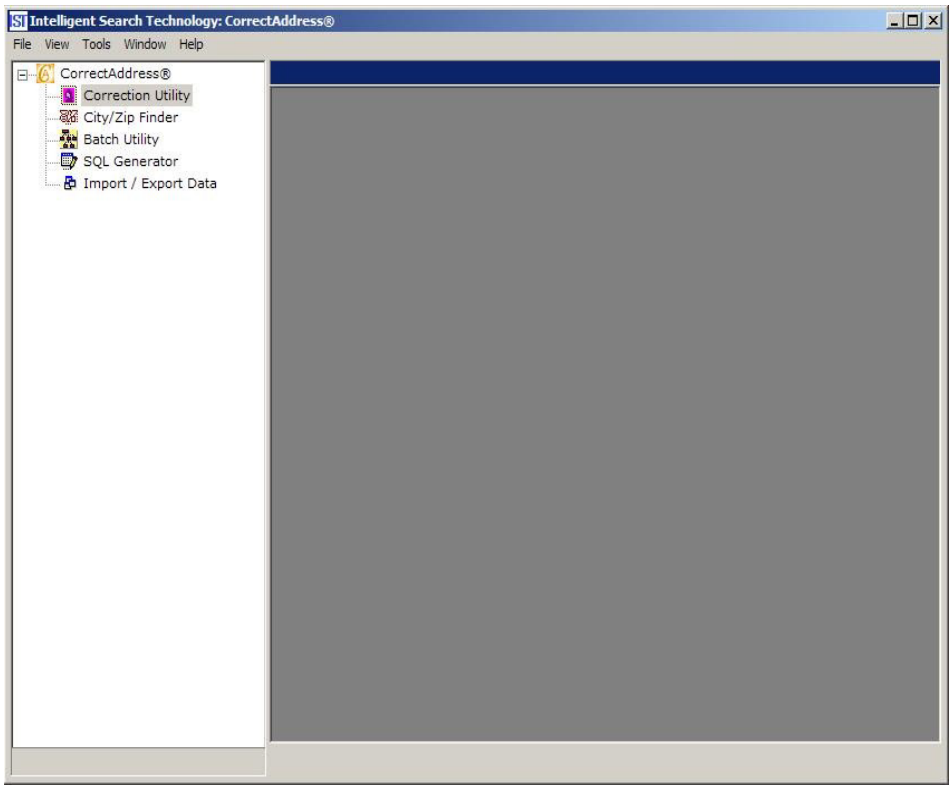


You can navigate the various drives and directories on your computer until you have the path you want displayed in the top-most text box. When done, click on the **OK** button; this will update the information on the Setup window. For ease of use, when initially installed all data files are copied to the same directory; it is recommended that these files not be split up among different directories.

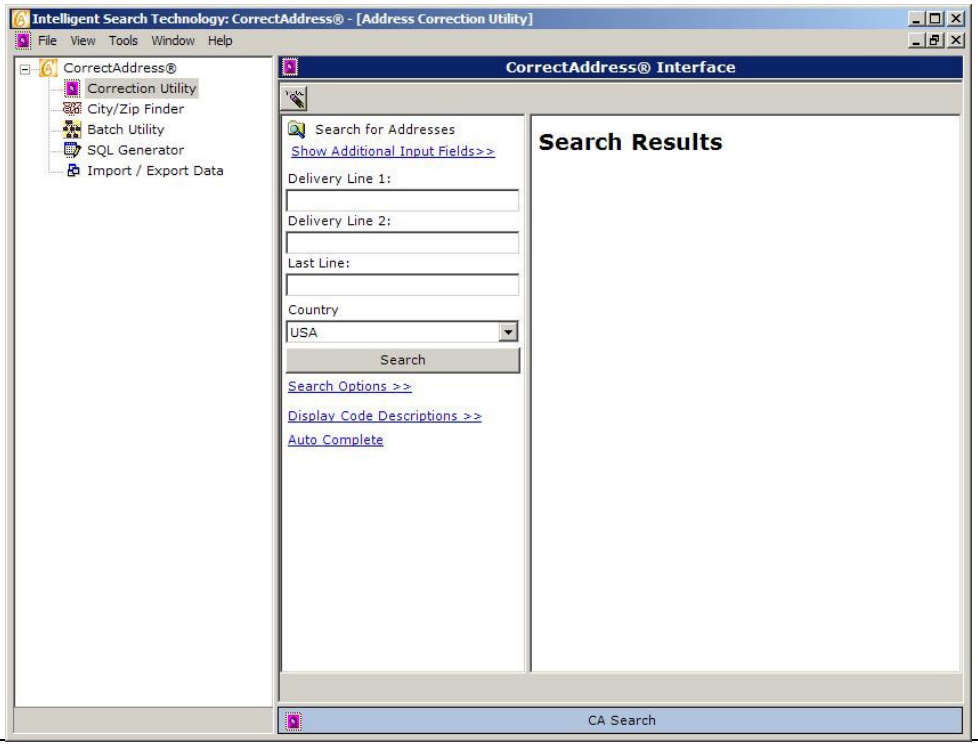
Upon finishing your changes, click on the **Save Settings** button to save all your changes. Neglecting to save before closing this window will undo any changes you have made. See *Listing of CorrectAddress Data Files* on *Page D-1* for a listing of the data files.

USING THE CORRECTION UTILITY

CorrectAddress comes with a Correction Utility that allows the user to input a single address and get a corrected response with additional information such as Line of Travel numbers and Delivery Point Codes. To access the Correction Utility, select it from the list of available utilities on the left side of your *CorrectAddress* screen.



This brings you to the Correction Utility window, as follows:



To input an address for correction, enter the appropriate values in the 5 fields and press **Search Now**.

The Fields on the Search for Addresses screen are as follows:

***Delivery Line 1** – Street address.

Delivery Line 2 – Secondary street address if present. Only exists in a dual addressing case.

Last Line – The city, state, and ZIP, as they would appear on a piece of mail.

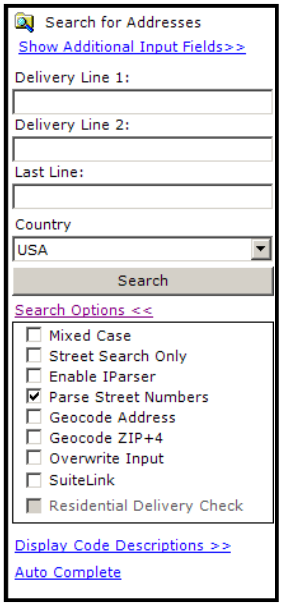
Country – The Country of the searched address.

*Fields marked with an * are required.*

***CorrectAddress* will correct misspellings to street addresses and city names. If a valid ZIP Code is supplied, city and state can be omitted. If the valid city and state are supplied, a ZIP Code can be omitted**.

SHOW ADD ITIONAL INPU T FIELDS

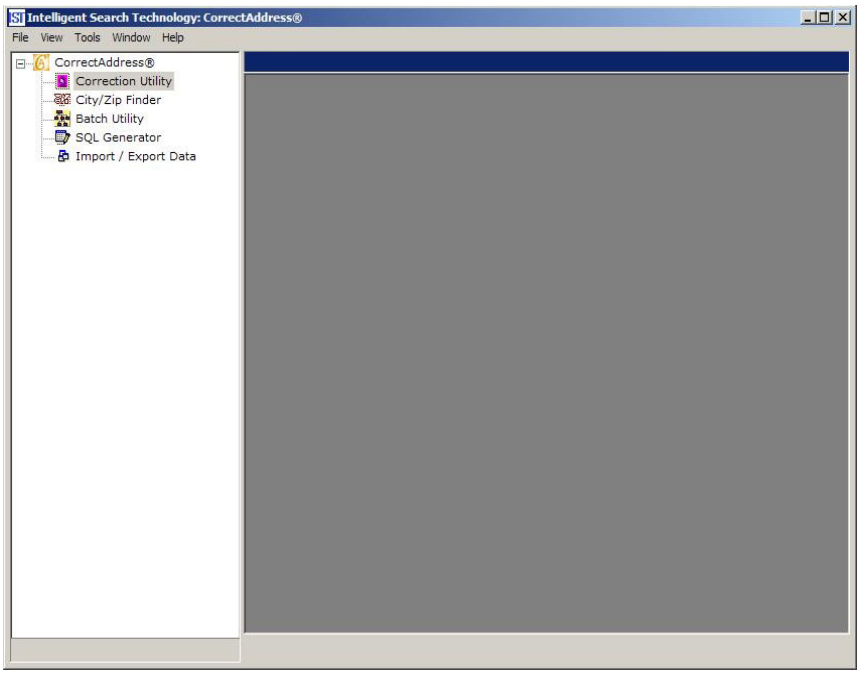
Clicking on the **Show Additional Input Fields** hyperlink will display a **Firm Name** field and an **Urbanization** field.



Firm Name - Firm name if one exists or is available, otherwise leave blank.

Urbanization - Urbanization name if one exists (Puerto Rico only) or is available, otherwise leave blank.

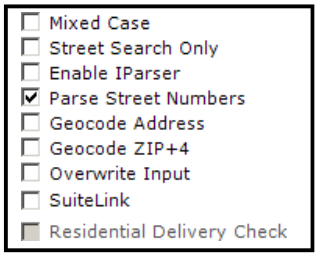
In the next screen we see a misspelled street address being input to the Correction Utility.



After clicking on the **Search Now** button, the corrected address fields are displayed on the left. Additional information about the address is displayed in the results grid at the right. If an address returns a multiple number of results, the address will not be corrected but each result will be displayed in the results grid. If no exact match could be found, the original input is returned, and nothing is displayed in the grid. Clicking on the **Clear** button nullifies the input and results. The **Export Results** button opens a new window containing the corrected address and results grid so you can keep old searches visible.

SEARCH OPTIONS

The Search Options section contains a list of advanced features that can be added to an address search. Below is a list of advanced options:



Mixed Case – Output from *CorrectAddress* is reported in all capitals unless this option is selected.

Street Search Only – This option allows you to search for similar street names in a given state; this will return records with a score based on how close they match to a given street name.

Enable IParser – This switch makes the software attempt to pre-parse an address before matching is attempted. Only effective in cases where delivery line information is split up in multiple fields, otherwise this option should be left off.

Geocode Address – This will return Longitude/Latitude and other geocoding information as well as the usual address information on a successful search. This option can only be used if you have the Geocoding add-in for *CorrectAddress*.

Geocode Only – This will take a ZIP+4 code alone and attempt to match it to geographic information. This option can only be used if you have the ZIP+4 level Geocoding add-in for *CorrectAddress*.

Auto–complete – this feature allows you to enter a partial city and state or partial zip code and a series of potential matches will be displayed and can be entered into the search fields.

Parse Street Numbers – this feature will display the CASS street number range in readable format.

Overwrite Input – this feature will allow data to be overwritten with CASS-certified output if the address in question is CASS-certified.

Residential Delivery Check (add on feature) - this add on feature will verify delivery type status and determine with delivery is to a residence or a business.

Suite^{Link} - this feature allows users to append secondary (suite) information to a business address (see Appendix M)

DISPLAY COD E D ESCRIPTIONS

Display Code options will be displayed below the search results. The radio buttons on the Display Code Descriptions are keys that allow you to view possible error codes or return codes that are returned with the Search Results. The Display Code Options are as follows:

ADDRESS CODES

An example of the Address Code option is shown below. This is a key for the Return Code returned with the Search Results.

Address Codes

Code	Description
Return Code: 1	Match found, four-digit ZIP add-on assigned.

RETURN CODES

An example of the Return Codes options is shown below. This is a complete listing key for possible Return Codes returned with the Search Results. Refer to *Return Codes* on *page C-5* for descriptions.

ERROR CODES

An example of the Error Code option is shown below. This is a complete listing key for possible Error Codes return with the Search Results. Refer to *Error Codes* on *Page C-5* for descriptions.

GEOCODES

An example of the Geocodes options is shown below. This is a listing of possible Geo Return Codes and Geo Error Codes that can be returned with Search Results.

Geo Codes

Code	Description
Geo Return Codes	==========
0	No match for specified address or ZIP+4
1	Match for specified address or ZIP+4
>1	Multiple matches for specified address or ZIP+4
99	Internal error.
Geo Error Codes	==========
00	Zip code is invalid
01	Internal error
02	Internal error
03	Internal error
04	No record for specified ZIP+4
05	No record for specified ZIP+4; near ZIP+4 match
06	No record for specified ZIP+4; Highrise default ZIP+4 used
07	No record for specified ZIP+4; near Street ZIP+4 used
08	No record for specified ZIP+4; near street ZIP+4 used
09	No address level record exists; attempting to use ZIP+4
99	Internal error

DELIVERY POINT VALIDATION (DPV) CODES

An example of the DPV Codes option is shown below. This is a listing of possible DPV Codes Confirmation, CMRA Indicators, False Positive Indicators, No Stat Indicator, and DPV Footnotes that may be returned with Search results.

An example of the LACS Codes is shown below. This is a listing of possible LACS Indicator Codes and LACS Return Codes that could be returned with Search Results.

DPV Codes

Code	Description
Delivery Point Validation	==========
Confirmation Indicator	==========
Y	Address was DPV confirmed for both primary and (if present) secondary numbers.
D	Address was DPV confirmed for the primary number only, and the secondary number infor
S	Address was DPV confirmed for primary number only, and secondary number information p
N	Primary number failed to DPV confirm.
Enhanced Return Code	==========
Y	Address was DPV confirmed for primary and secondary numbers necessary to determine a
D	Address was DPV confirmed for the primary number only. Secondary information was missi
S	Address was DPV confirmed for the primary number only, the secondary number informatio
N	Primary number failed to DPV confirm.
R	Address confirmed but assigned to phantom route R777 and R779 and USPS delivery is not
CMRA Indicator	==========
Y	Address found in CMRA Table.
N	Address not found in CMRA Table.
False Pos Indicator	==========
Y	Address found in False Positive table. Please contact IST to reenable DPV.
N	Address not found in False Positive table.
No Stat Indicator	==========
Y	Address found in NO-STAT table.
N	Address not found in NO-STAT table.
No Stat Reason Code	==========
01	IDA - Internal Drop Address. These are addresses that do not receive mail delivery directly
02	CDS - The delivery is new construction and delivery has not been established or on a Rural/
03	Collision - These addresses do not actually DPV® confirm. In this case, the 'Y' should be se
04	CMZ - College/Military Zone, & Other types. These are ZIP + 4® records USPS has incorpor
05	Regular No-Stat - The address is no longer a possible delivery, the address is on an R777 r
06	Secondary Required - The address requires secondary information.
DPV Footnotes	==========
AA	Input address matched to the ZIP+4 file
A1	Input address not matched to the ZIP+4 file
BB	Input address matched to DPV (all components)
CC	Input address primary number matched, secondary number not matched; secondary numb
C1	Input address primary number matched, secondary number not matched; secondary numb
F1	Input address matched to a military address
G1	Input address matched to a general delivery address
IA	Informed address identified
M1	Input address primary number missing
M3	Input address primary number invalid
N1	Input address primary number matched to DPV but address missing required secondary nu
PB	Identified PO Box street address
P1	Input address RR or HC box number missing
P3	Input address PO, RR, or HC box number invalid
R1	Input address matched to CMRA but PMB designator not present (PMB 123 or #123)
RR	Input address matched to CMRA and PMB designator present (PMB 123 or #123)
R7	Addresses that are assigned to a phantom route of R777 or R779
TA	Input address primary number matched to DPV by dropping trailing alpha
U1	Input address matched to a unique ZIP Code
Vacant Indicator	==========
Y	Address listed in the table of vacant addresses.
N	Address not found in the table of vacant addresses.
PBSA Indicator	==========
Y	Address listed in the table of post office box street addresses.
N	Address not found in the table of post office box street addresses.
Drop Indicator	==========
Y	Address was found in the table.
N	Address was not found in the table.
PO Box Throwback Indicator	==========
Y	Address was found in the table of residences and businesses that choose to receive delivery
N	Address was not found in the table of residences and businesses that choose to receive deli
Non-Delivery Day Indicator	==========
Y	Address was found in the table.
N	Address was not found in the table.
Non-Delivery Day Code	==========
NYNYNYY	Y is set for each day that represents the day(s) that do not receive delivery when the addre
No Secure Location	==========
Y	Address was physically accessibly, but cannot leave a package due to security concerns.
N	Address was not found in the table of residences and buildings with no security.
Door Not Accesible	==========
Y	Address was found in the table of physically inaccessible residences and buildings.
N	Address was not found in the table of physically inaccessible residences and buildings.

LACS CODES

LACS Codes

	Code	Description
▶	LACS Indicator	==========
	Y	New Address provided
	N	New Address not provided
	F	LACS False Positive encountered. Please contact IST to reenable LACS Link.
	S	Secondary number dropped from input address. New address provided
	LACS Return Code	==========
	A	LACS Record Match - the input record matched to a record in the master file. A new address could be furnished.
	00	No Match - the input record COULD NOT BE matched to a record in the master file. A new address could not be fur
	14	Found LACS Record: New Address Would Not Convert at Run Time - the input record matched to a record in the ma
	92	LACS Record: Secondary Number Dropped from Input Address - the input record matched to a master file record,

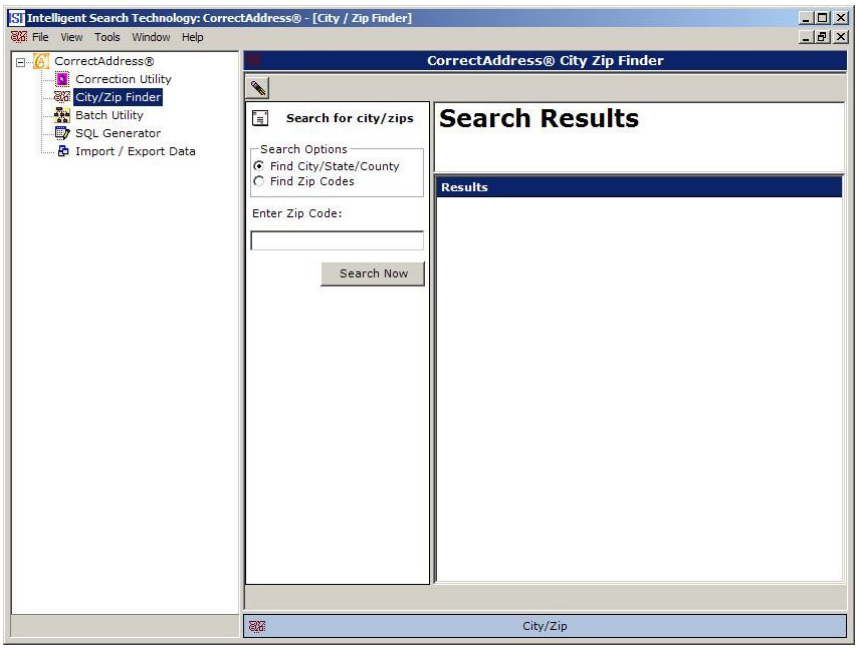
AUTO COMPLETE

The Auto-completion Wizard will be activated when you click on the Auto-complete hyperlink in the Search for Addresses area of the Correction Utility windows. Enter a city, state or zip and a list of possible matches will be displayed. Clicking on the **Finish** button will enter the information on the appropriate Search Fields.

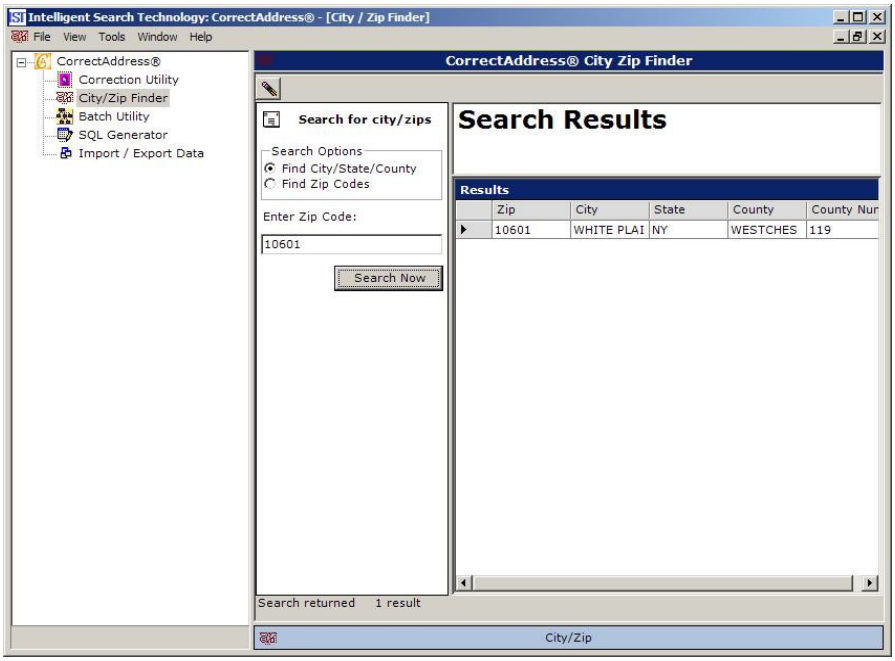
USING THE CITY/ZIP FINDER

The **City/ZIP Finder** program allows the user to input a ZIP Code and get back the preferred city name that corresponds to that address, the county it resides in, and the county number. It also allows the generation of a list of valid ZIP Code for any city/state combination.

To use the **City/ZIP Finder**, select **City/ZIP Finder** from the list of utilities on the left, as shown below.



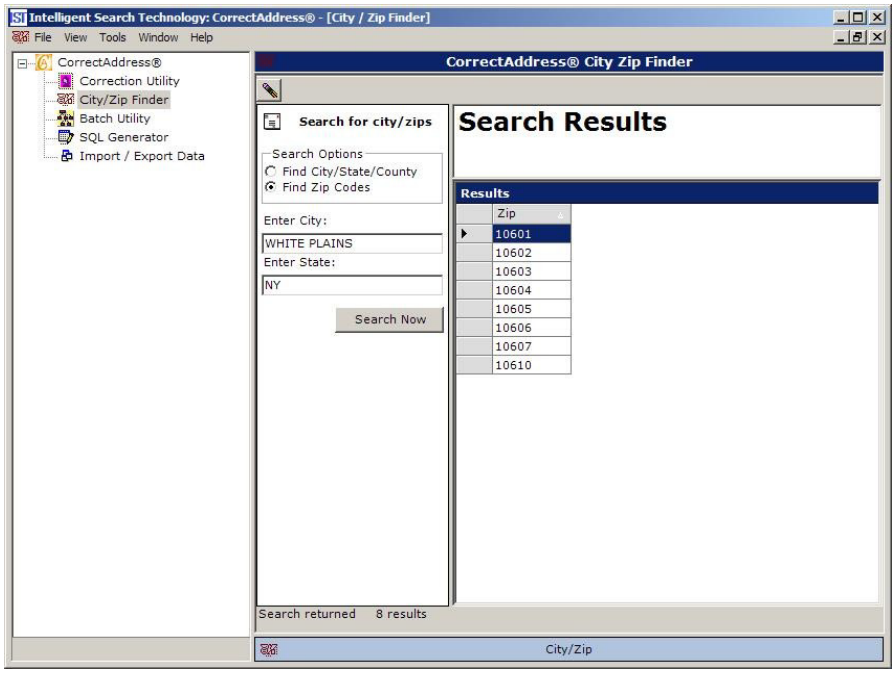
This brings up the City/ZIP Finder window, as shown below.



To use the City Finder, simply input a 5-digit ZIP Code as directed and click on **Search Now**.

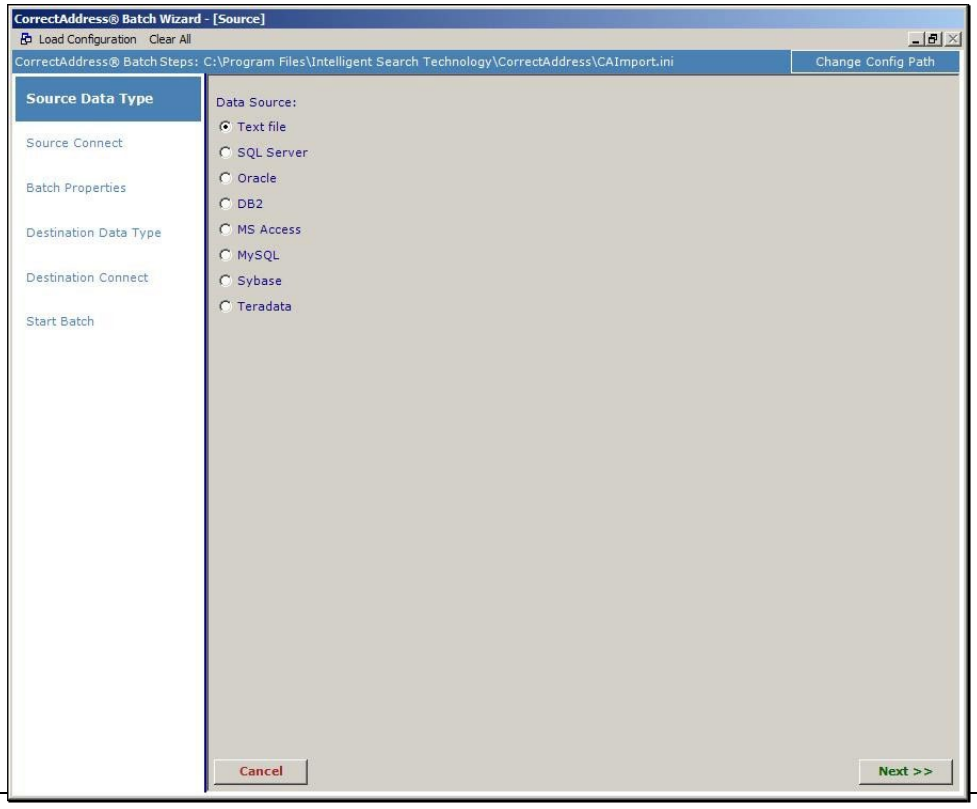
The resulting city name, state, county and county number are all displayed in the results grid. Clicking on the **Erase** button will clear all results and the input ZIP Code.

To get a list of ZIP Codes in a valid city/state combination, select Find ZIP Codes from the **Search Options** box. The city and state boxes will become enabled, after inputting a city and state, press Search Now. The results will be displayed in the grid as shown below:



USING THE BATCH PROCESSOR WIZARD

CorrectAddress comes equipped with a Batch Processor wizard which can take addresses contained in text files, Microsoft Access Databases, SQL Server Databases, Oracle, and other database systems and return the corrected addresses in any of those formats. The process can run in both single-threaded and multi-threaded modes (see Appendix G for configuration

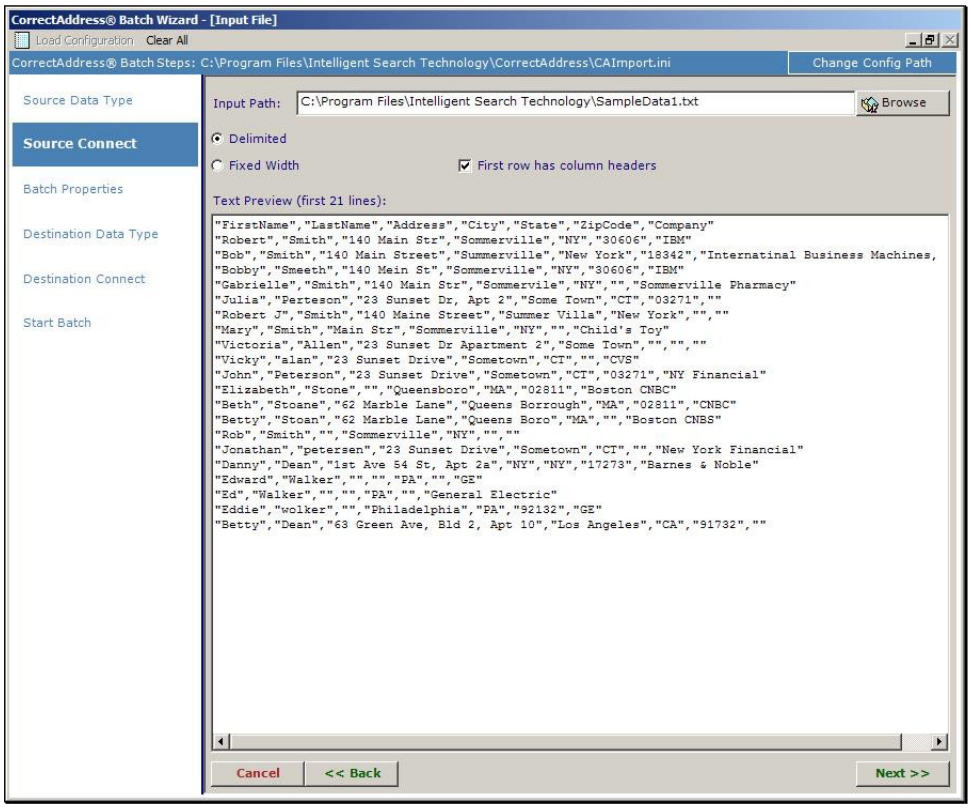


details). To enter the Batch Processor, choose the **Batch Utility** from the list of utilities on the left menu tree. This will activate the Batch Processing screen as shown below. By clicking on the appropriate radio button, you will need to specify the Source Data Type to be processed.

Click on the **Next** button to display the **Source Connect** window.

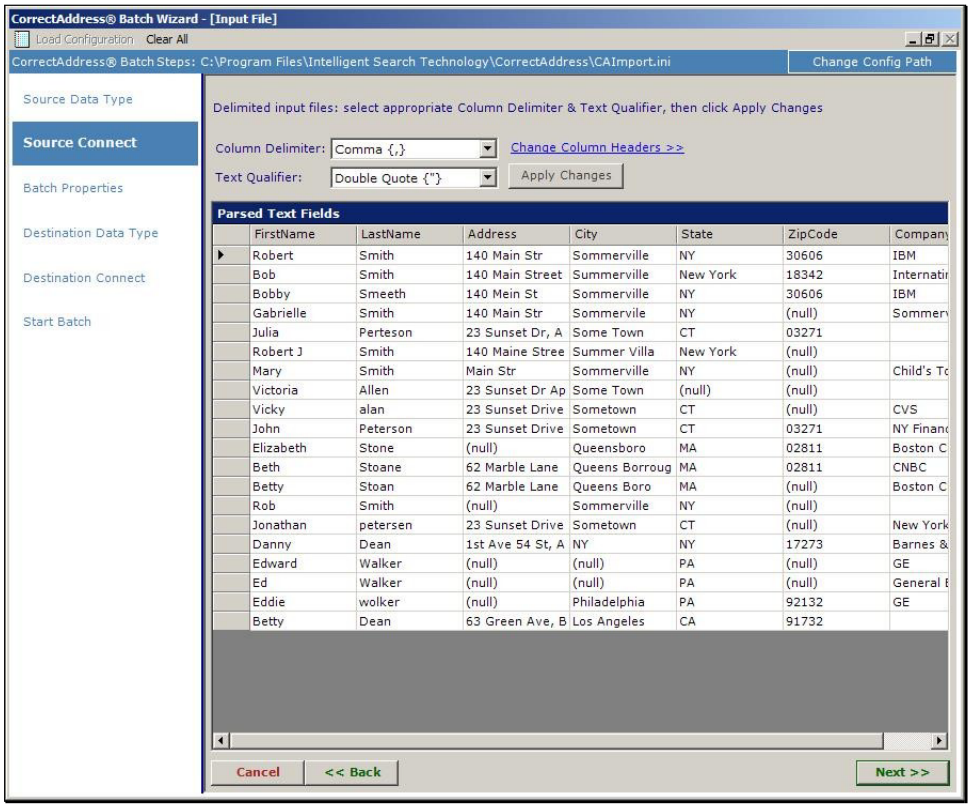
SOURCE CONNECT

1. Click on the **Browse** button to specify a path to a file. In this case a delimited text file will be selected. Click on the **First row has column headers** checkbox to select whether the first row of text contains column headers.

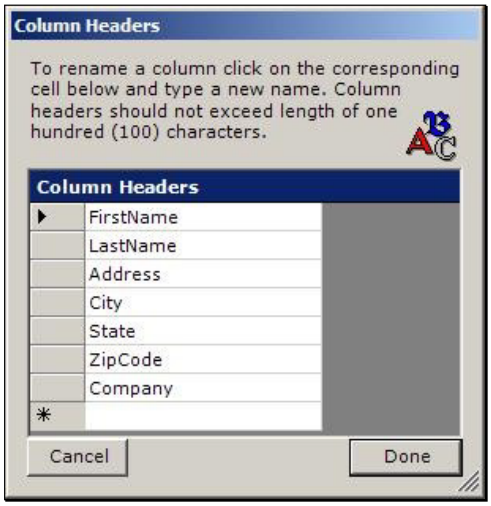


2. Click on the desired radio button to specify whether the .text file is **Delimited** or is **Fixed Width**. Click on the **Next** button to proceed to the next screen.

3. The next screen that will be active is the **Parsed Text Field** screen which displays the parsed fields. You will notice the **Column Delimeter** and **Text Qualifier** fields that were used to parse the text file are displayed. You can change this by using the appropriate drop-down list, making the desired changes to the parsing mechanism and then click on the **Apply Changes button.**



a. Clicking on the **Change Column Headers** hyperlink will display the following. From the **Column Headers** window, you can rename columns. Click on the **Done** button when finished.



b. Clicking on the **Next** button will display the **Batch Properties** window.

BATCH PROPERTIES

The list box on the left (underneath the table name) shows the available fields in the input connection. In order for a job to be run, *CorrectAddress* must know what fields contain the pertinent address information it needs.

There are 5 types of data that it uses.

Retained Fields

Firm/Recipient Name

Urbanization Name

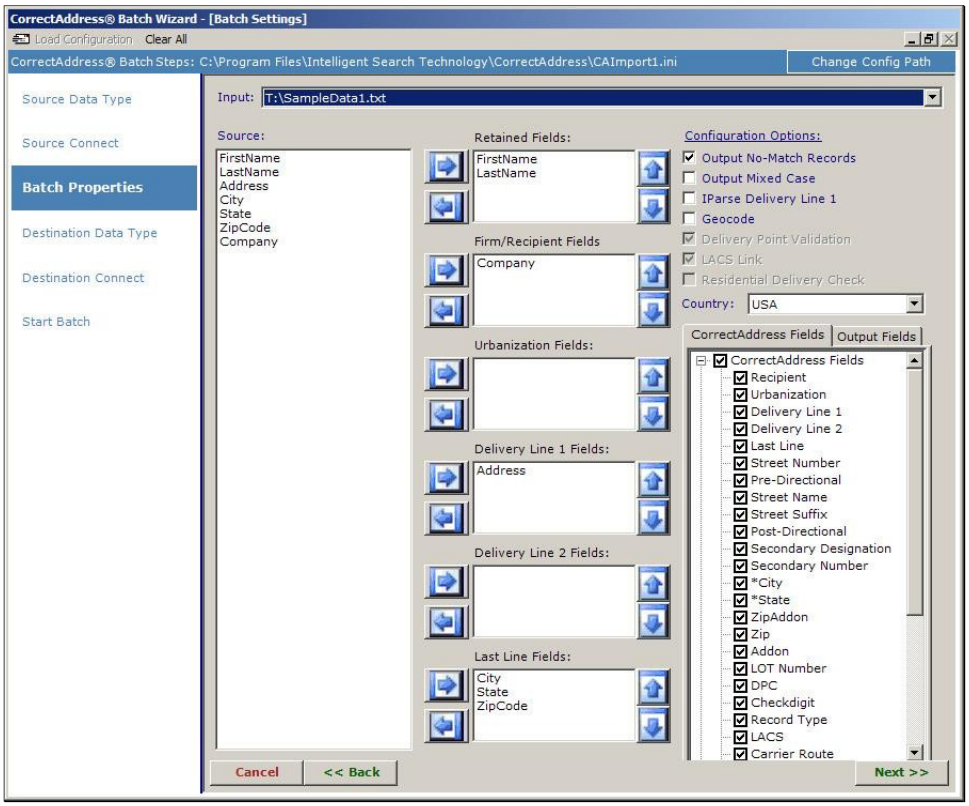
Delivery Line 1

Delivery Line 2

Last Line

Any fields that are placed in the **Retained Fields** category allow for the retention of useful record information such as row **ID** numbers. Note that if you are updating an old table you will not be able to set any retained fields this way.

Refer to *Using the Correction Utility* on *Page 6-3* for descriptions of these fields.



CONFIGURATION OPTION S

Output No Match Records - Unless this box is checked, only addresses that match uniquely will be output.

Output Mixed Case - By default, *CorrectAddress* returns information capitalized. If this setting is selected, address information will come out in mixed case.

Iparse Delivery Line 1 - Use only if you have a table which may have address information that is erratically placed in different fields. By selecting this and concatenating the suspect fields into the Delivery Line 1 fields, Iparser will attempt to pre-parse the address before sending it to *CorrectAddress*. If one of the fields contains **Firm Name** you can select the Extract Firm/Recipient sub option. This will attempt to parse out the firm name as well and send it to *CorrectAddress*.

Geocode Address - This option can only be used if you also have purchased the *CorrectAddress* Geocoding add-in (*See Appendix C*). If you have, you may select this option to append geocoding information to the regular address information. Along with a validated address, the Geocode Address module returns geographical coordinates (latitude and longitude), Census tract and block numbers, and more. Enabling the Geocode Address Option will activate additional fields in the *CorrectAddress* Fields list.

Delivery Point Validation – With DPV checking enabled, capable of confirming over 145 million physical mail delivery points throughout the United States and its territories. The DPV component will also determine if the address belongs to a Commercial Mail Receiving Agency (CMRA) and provide other useful information to indicate match quality. Enabling the DPV Configuration Option will activate additional DPV fields in the *CorrectAddress* Fields list.

LACS$^{Link™}$ - This option allows addresses that have been converted due to various USPS changes to be linked with their new addresses. This affects many of rural-style U.S. addresses that have been assigned city-style street names for 911 emergency response systems. Additionally, LACS^{Link} covers street names that have been modified by municipalities in recognition of an individual or an event. Enabling the LACS^{Link} Configuration Option will activate 2 additional LACS^{Link} fields in the *CorrectAddress* Fields list.

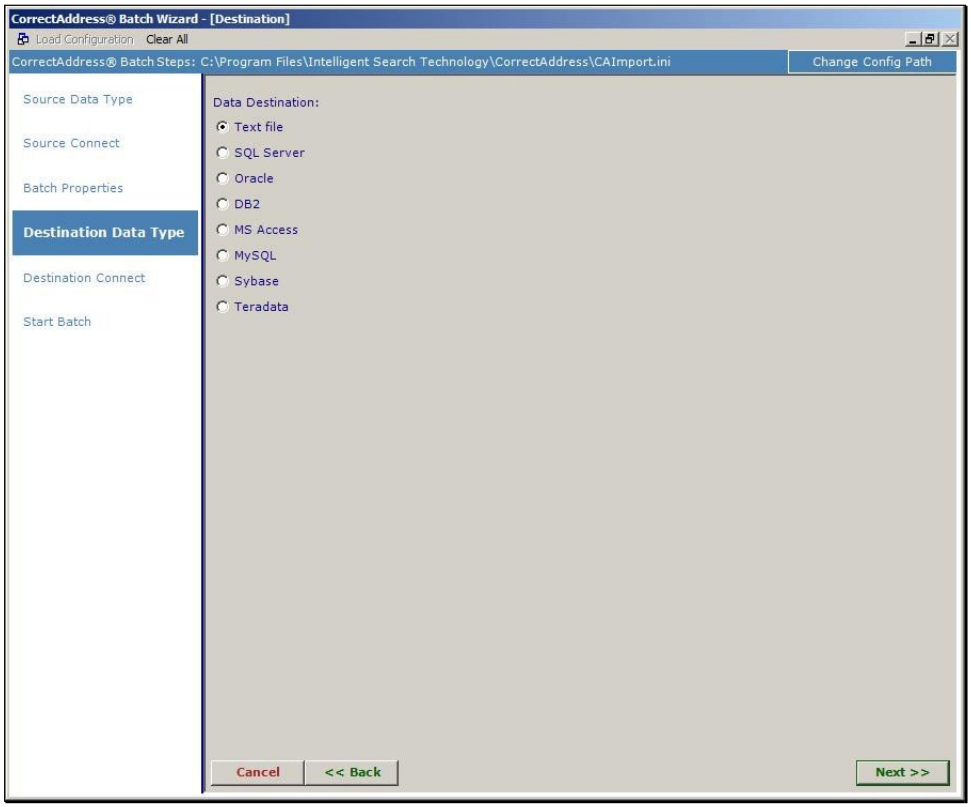
The Field Output tab lists the address information that you can have output to your destination table. When you select a field, it will be added to the fields list below and the order displayed in that field box will mirror the output table. Using the arrow keys, the order of the fields may be changed.

Clicking on the **Next** button will display the **Data Destination window** as shown below. In this window you will have the options to determine the type of data that will be sent to the *CorrectAddress* batch processor. Click on the desired radio button to enable your selection.

Click on the **Next** button to proceed to the **Destination Connect** window.

DESTINATION DATA TYPE

The **Destination Connect** window is used to specify connection details for the *CorrectAddress* data connection. At any time you may clear the configuration by clicking on **Clear All** on the menu bar and return to **Source Data Type** and begin the wizard from the start.



By default the **Configuration Path** is set to a path of **C:\Program Files\Intelligent Search**

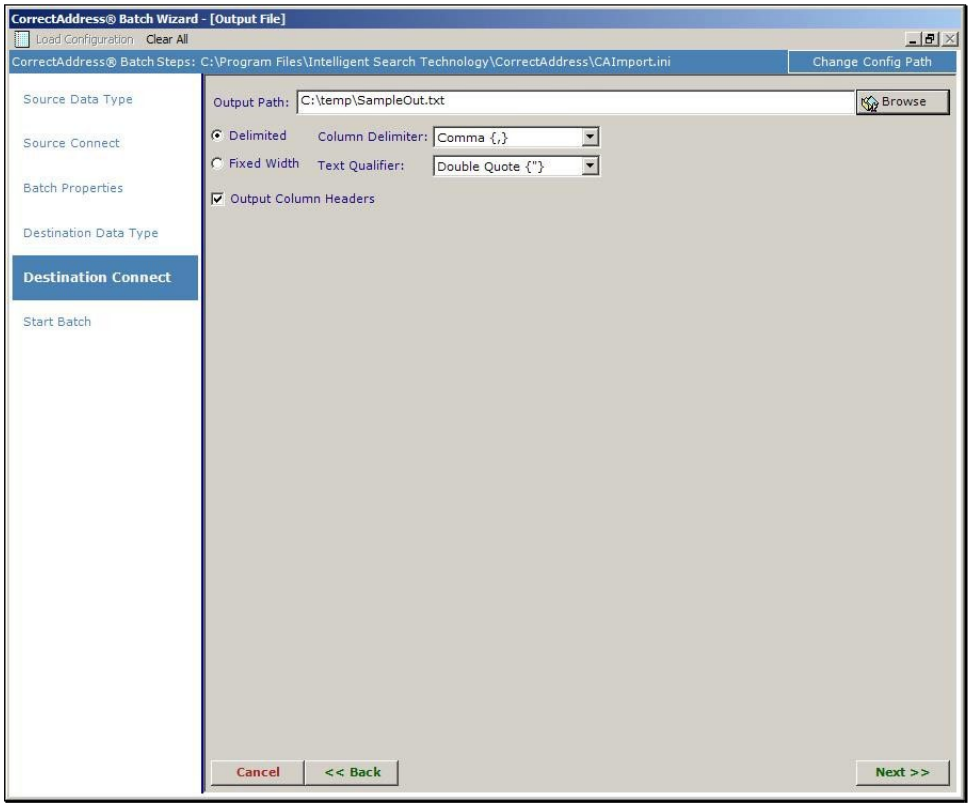
Technology\CorrectAddress. This can be changed by clicking on the **Change Config Path** button and selecting a new directory to store the .ini configuration file.

You can also recall a previous configuration by clicking on the **Load Configuration** on the menu bar of the Batch Processor and selecting a previously run configuration .ini file.

Click on the **Next** button to continue.

DESTINATION CONNECT

Select a file name for your output. Using the radio buttons you must specify whether you would like the text file, in this case, to be Delimited and what the Column Delimeter would be or if you would like that same text file to be Fixed Width and what the Text Qualifier would be. A checkbox can also specific whether to output Column Headers.



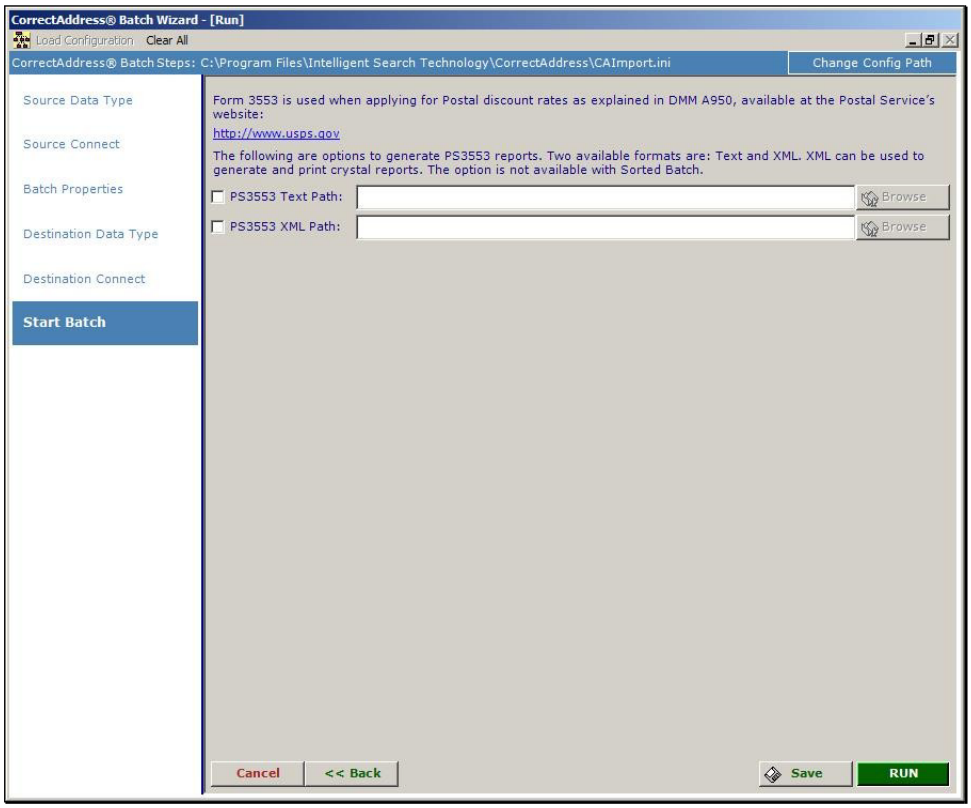
Click on the **Next** button to start the batch process.

START BATCH

You will be given the option to generate PS3553 reports. Follow the prompts to generate an XML file to generate a crystal report or click on the **PS3553 Text Path** check box and select a file name to generate a text file.

Click on the **RUN** button to start the batch process. If you choose, you can click on the **Save** button to save the configuration.

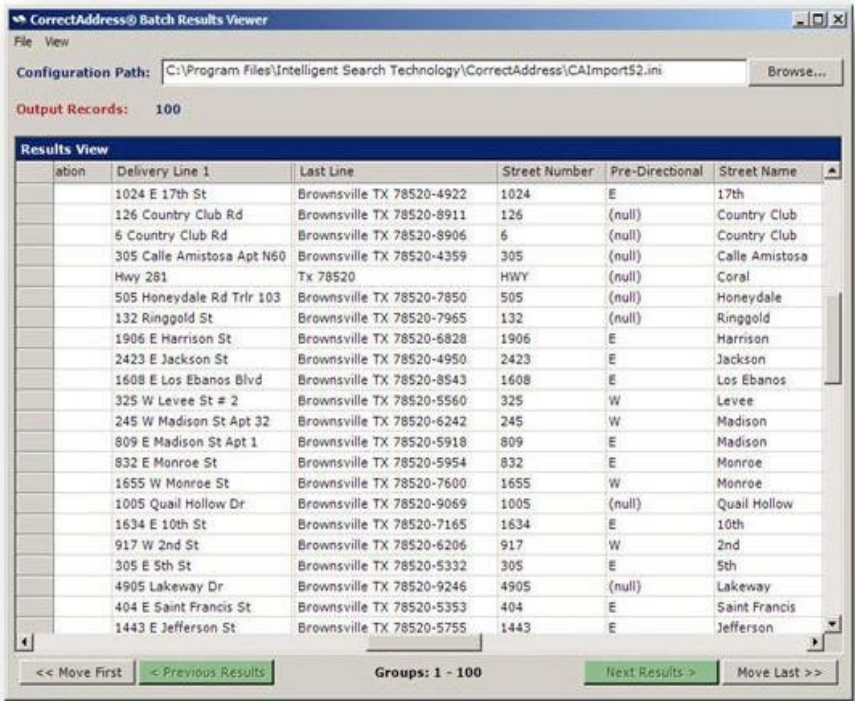
An option to process using the Sorted Batch Processor to process flat text files as available by clicking on the Process using Sorted Batch Processor check box at the bottom of the screen.



Clicking on the **Save** button will save the *.ini file specified in configuration path to be loaded and run later. A configuration may be loaded by clicking on **File→Load Configuration** from the Batch Wizard.

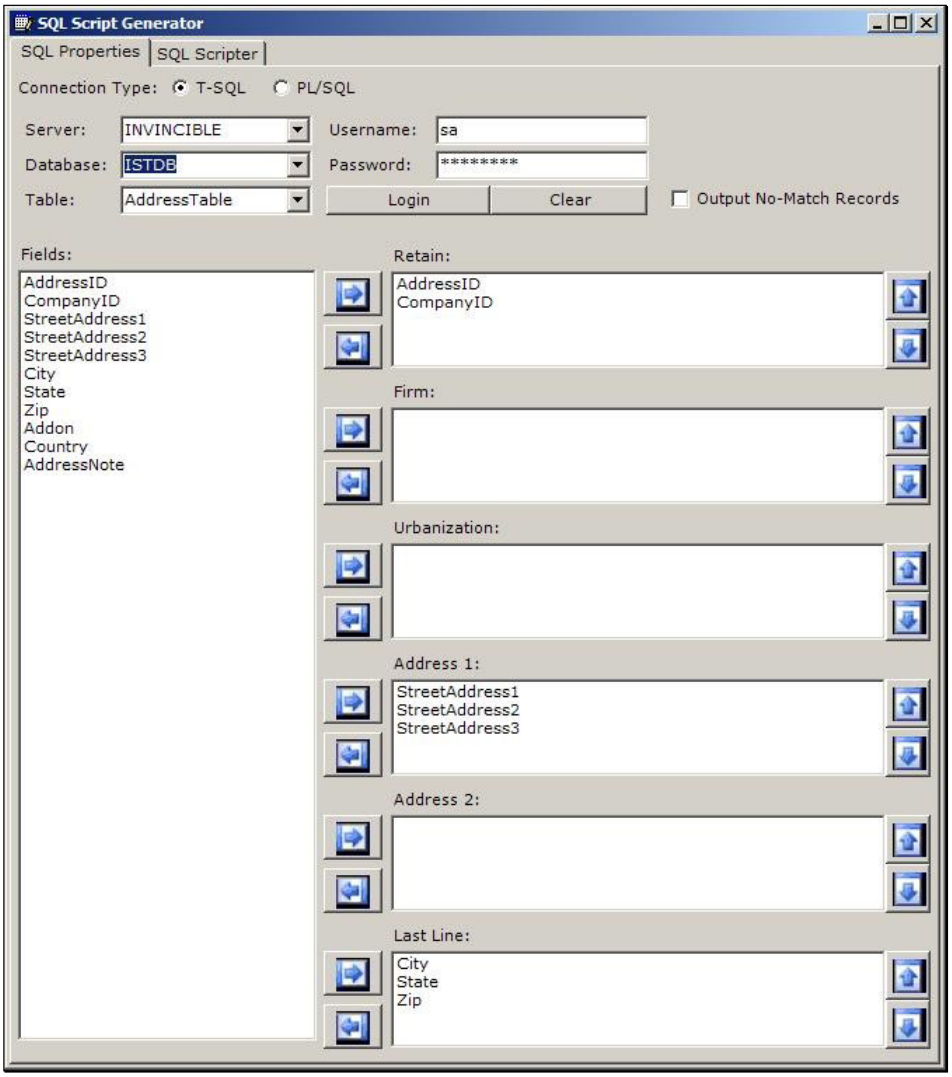
An In-progress window will be visible showing the batch process. When complete, a Batch complete indication will be shown in the **Stage:** field.

The .ini file created with all of your specifications can be viewed by clicking on **View→View Results** on the *CorrectAddress* menu bar. An example is shown below.



USING THE SQL GENERATOR

The **SQL Generator** program allows the user to create and save a T-SQL or PL/SQL batch script for jobs using *CorrectAddress* stored procedures. To access the **SQL Generator**, select it from the list on the left as shown below. This will bring up a new window where you can login to the Oracle or SQL Server you wish to run a batch job in, as shown below:



The SQL Script Generator will automatically attempt to bring up a list of SQL Server connections when it is first loaded. If you need to regenerate the SQL Server list at any time, click on the **Login** button. If you wish to login to an Oracle server, select PL/SQL for the Language Type and click on the **Login** button. The program will ask you for your service name, database, and User ID and password and attempt to connect. Upon successfully connecting, the **Fields** box will contain a list of all columns in a given table. You can set which fields pertain to the various address information types by moving them from the **Fields** box to the appropriate destination field on the right.

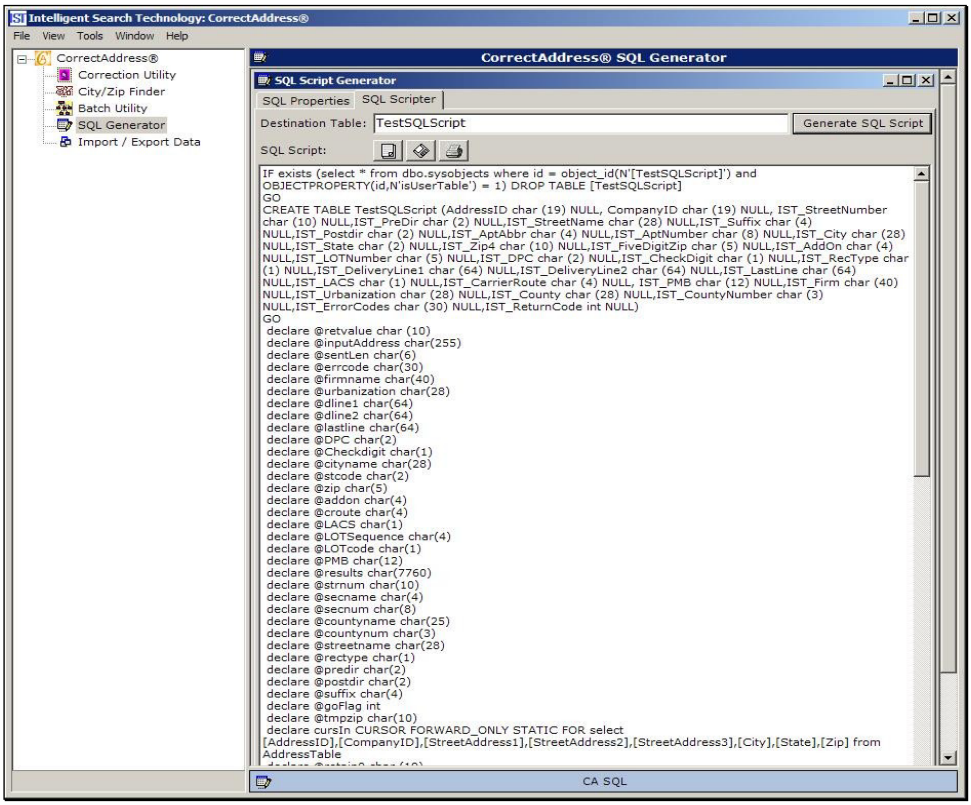
Below is a list of the fields and their definitions:

1. **Firm Name**
2. **Urbanization Name**
3. **Address 1**
4. **Address 2**
5. **Last Line**

The Firm Name is any company name for the addresses being input, if applicable. It is not required but does allow *CorrectAddress* to match the address closely to firm records if they exist. **Urbanization** is only used in Puerto Rican addresses and is not required; it simply helps narrow down the location of the address. **Delivery Line 1** is the first address line that would appear on a piece of mail, the actual street address. For example, 445 Hamilton Ave Ste 608. **Delivery Line 1** is mandatory and cannot be left blank. **Delivery Line 2** is the second street address line if present. A **Delivery Line 2** is only present in the case of dual addressing where a street address and a PO Box number or Rural Route both appear on a piece of mail at the same time; most often it will be left completely blank. Last Line information is the same as the last line of a piece of mail, precisely the city, state, and ZIP Code of the address. For example: White Plains, NY 10601. This field is also mandatory and cannot be left out.

If your information (such as city/state/ZIP) are separated into multiple fields rather than one single line, you can move them into their appropriate fields in order and they will be concatenated together to produce the proper line. For example, the fields **City**, **State**, and **ZIP** can all be moved via the right arrow button next to the **City**, **State**, and **ZIP** Fields box. The up and down buttons to the right of that box allow you to select a field and move it around so that they are in the proper order. This way, *CorrectAddress* will concatenate the **City** field with the **State** field and the **ZIP** field. This produces the appropriate **Last Line** field.

After setting your field definitions, you can select the **SQL Scripter** tab to input your output table name. After you have named your destination table, clicking the **Generate SQL Script** button will create a script that can be executed from your SQL Query Analyzer or SQL*Plus window. Note that for PL/SQL the script will actually create a stored procedure which must then be executed. An example is shown below:



Clicking on the **Copy** button will copy your SQL Script to the Microsoft® Windows® clipboard.

Clicking on the **Save** button will save your SQL Script to a .sql file.

Clicking on the **Print** button will print your SQL Script to a printer of your choice.

Chapter7- Troubleshooting

GENERAL TROUBLESHOOTING ISSUES

1. *Every time I run CorrectAddress, I get a return code of-99.*

   Your shared object or library cannot find the data files. Check your **CADataPath.h** or **ISTfpath.h** (if exists) that was used during the object build; make sure it points to the right data location.

   There is an additional environment variable, **CA_DATA**, that may be set to overwrite settings.

2. *I get "Segmentation Fault" when trying to run CorrectAddress*

   Your shared object version does not match the data. Check release months for both .so source and the data files, make sure they match.

   Data files are corrupt. If you FTP your data files, make sure they were transferred in binary mode. Also check that all data files are from the same month (except **tiger0..9.txt** files).

PLATFORM-SPECIFIC ISSUES

AIX- SPECIFIC ISSUES

1. *When using the shared object, I receive errors about undefined symbols.*

   Resolution:

   Make sure the **.exp** file is attached when .so is built and contains all export symbols (function names).

2. *Architecture Issue: AIX 5.2 64bit: When linking with ld, I receive the following error:*

   Resolution:

   If you see this error, you may be exceeding the default 256MB process limit. If you have at least 1 GB of memory on your AIX system, you can complete the following steps to increase the maxDATA from 256MB to 1GB:

   a. Login as root.
   b. Change to the /usr/ccs/bin directory.
   c. Make a copy of the original executable file 'bind'.
   d. Create the script file maxdata, by saving this text file to your hard drive. Be sure to save the file as "maxdata" with no extension.
   e. Enter chmod 755 maxdata to make the script file executable.
   f. Enter maxdata bind 4 to modify the o_maxdata file from the default 0x00000000 to 0x40000000.
   g. To check the maxDATA, enter dump -ov bind. The output should be 0x40000000.

```
ld: 0711-101 FATAL ERROR: Allocation of 2738976 bytes failed in routine
get_RLDs.

There is not enough memory available.
Please check your ulimit or paging space or use local problem reporting
procedures.

make: 1254-004 The error code from the last command is 12.
```

3. *Architecture Issue: AIX 5.2 64bit: I'm having problems using the '-q64' option with gcc.*

   Resolution:

   Use **-maix64** flag instead of **-q64.**

```
MESSAGE>gcc: unrecognized option `-q64'
MESSAGE>gcc: unrecognized option `-q64'
MESSAGE>gcc: unrecognized option `-q64'
MESSAGE>gcc: unrecognized option `-q64'
Compilation completed
```

Linking files into the shared object...

*MESSAGE>gcc: `-b' must come at the start of the command line. The command 'gcc -o libCorrectA.so *.o -bE:CorrectA.exp -bM:SRE -bnoentry' was NOT successful! ExitValue: 1*

Resolution:

Use **-maix64** flag instead of **-q64.**

Resolution:

Use **-maix64** flag instead of **-q64.**

LINUX- SPECIFIC ISSUE S

1. *General Architecture Issue: I receive the following error when trying to run CallCorrectA:*

   Resolution:

   Shared object was not compiled with optimizations. Recompile with **–O flag.**

```
./CallCorrectA: relocation error: ./libCorrectA.so: undefined symbol: fstat
```

LINUX IBM POWERPC - SPECIFIC ISSUES

1. *Architecture Issue: IBM PowerPC 64-bit: What are the compiler/ link commands for a 64-bit Linux IBM PowerPC machine?*

   Resolution:

   Compile Command: **gcc -m64 -mpowerpc64 -shared –fPIC**

   Link Command: **gcc -m64 -mpowerpc64 -shared -fPIC -o libCorrectA.so *.o**

SOLARIS - SPECIFIC ISS UES

1. *Architecture and Compiler Issue: Solaris 9 with gcc 3.4.2: We used the following compile and link commands:*

   Resolution:

   You need to pass the **–fPIC** flag to **gcc.** See the compile and link commands below:

   gcc -c -g -O2 -I. -fPIC *.c

gcc -shared -fPIC -o libCorrectA.so *.o

```
---------------------------------------

gcc -O -c *.c

gcc -c *.c

ld -G -o libCorrectA.so *.o

---------------------------------------
```

The compile and link completes okay, but when I check out the shared object it complains about symbol not found (see below).

```
# ldd -u libCorrectA.so
      libucb.so.1 => /usr/ucblib/libucb.so.1
      libresolv.so.2 => /usr/lib/libresolv.so.2
      libsocket.so.1 => /usr/lib/libsocket.so.1
      libnsl.so.1 => /usr/lib/libnsl.so.1
      libelf.so.1 => /usr/lib/libelf.so.1
      libc.so.1 => /usr/lib/libc.so.1
      libdl.so.1 => /usr/lib/libdl.so.1
      libmp.so.2 => /usr/lib/libmp.so.2
      /usr/platform/SUNW,Sun-Fire-V240/lib/libc_psr.so.1
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:__muldi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_udivdi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)
      symbol not found:_umoddi3 (./libCorrectA.so)

unused object=/usr/lib/libelf.so.1

unused object=/usr/lib/libmp.so.2
```

2. *General Architecture Issue: We receive the following compilerwarnings:*

Resolution:

Upgrade gcc to 3.0 or above.

```
MESSAGE >DPV.c:912: warning: malformed `#pragma pack'
stgresin1 src]# gcc -o CallCorrectA CallCorrectA.c libCorrectA.so
libCorrectA.so: undefined reference to `ks'
libCorrectA.so: undefined reference to `resetModDate'
libCorrectA.so: undefined reference to `doDPV'
libCorrectA.so: undefined reference to `rpa'
libCorrectA.so: undefined reference to `dpvfile'
libCorrectA.so: undefined reference to `rmZeros'
libCorrectA.so: undefined reference to `kr'
collect2: ld returned 1 exit status

DPV.o size is very small.
```

LANGUAGE-SPECIFIC ISSUES

JAVA- SPECIFIC ISSUES

1. *Exception occurred during event dispatching:*

   Resolution:

   Make sure **libCorrectA.so** is in the **java.library.path**. To display the contents of the path, add:

   System.out.println(System.getProperty("java.library.path")) to your java file.

   You may also choose to append the location of the **.so** to the **LIBPATH** variable.

```
java.lang.UnsatisfiedLinkError: no CorrectA in java.library.path

      at java.lang.Throwable.fillInStackTrace(Native Method)

      at java.lang.Throwable.fillInStackTrace(Compiled Code)

      at java.lang.Throwable.<init>(Compiled Code)

      at java.lang.Error.<init>(Error.java:43)

      at java.lang.LinkageError.
```

2. *How do I modify java.library.path dynamically?*

   Resolution:

   Use -D option as follows:

   ```
   javac javaCANativeDispatcher.java javaCallFI.java

   java –Djava.library.path=<location of your libCorrectA.so> CallFI CAImport.ini
   ```

3. *I get the following error message.*

   Resolution:

Your Java must be 32-bit and is unable to load a 64-bit shared object. Rebuild the object without 64-bit flags, or use 64-bit JRE.

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
/local/greg/IstCorrectAddress/libCorrectA.so: ld.so.1: java: fatal:
/local/greg/IstCorrectAddress/libCorrectA.so: wrong ELF class: ELFCLASS64
      at java.lang.ClassLoader$NativeLibrary.load(Native Method)
      at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1586)
      at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1511)
      at java.lang.Runtime.loadLibrary0(Runtime.java:788)
      at java.lang.System.loadLibrary(System.java:834)
      at javaCANativeDispatcher.<clinit>(javaCANativeDispatcher.java:806)

      at Test.main(javaCANativeDispatcher.java:947)
```

4. *I get the following error message when I run the builder, BuildLib.jar:*

   Resolution:

   Use **/usr/java/j2re1.x.x/bin/java -jar BuildLib.jar**

   It is possible that you are picking up the java runtime for gcj the Gnu Compile for Java. This is a Java front end.

   Make a symbolic link for **/usr/bin/java** to point to the correct directory.

```
<prompt># java -jar ./BuildLib.jar
Warning: -jar not understood. Ignoring.
Exception in thread "main" java.lang.NoClassDefFoundError: ..BuildLib.jar
      at gnu.gcj.runtime.FirstThread.run() (/usr/lib/libgcj.so.5.0.0)
      at _Jv_ThreadRun(java.lang.Thread) (/usr/lib/libgcj.so.5.0.0)
      at _Jv_RunMain(java.lang.Class, byte const, int, byte const, boolean)
      (/usr/lib/libgcj.so.5.0.0)
      at_gcj_personality_v0 (<IstCorrectAddresspath>/java.version=1.4.2) at
__libc_start_main (/lib/tls/libc-2.3.4.so)
at _Jv_RegisterClasses (<IstCorrectAddresspath>/java.version=1.4.2)
```

5. *I receive the following error message while running BuildLib.jar:*

   Resolution:

   CAPort.zip is corrupt, most likely due to incorrect file transfer (FTP in ASCII mode).

   Compare file sizes of **CAPort.zip** before and after the transfer, and re-copy if they do not match.

```
Preparing modules...

Extracting CorrectAddress(r) modules from archive...

Preparing CorrectAddress(r) modules for compilation...

An error occurred! Please contact Support.
java.util.zip.ZipException: error in opening zip file
      at java.util.zip.ZipFile.open(Native Method)
      at java.util.zip.ZipFile.<init>(ZipFile.java:204)
      at java.util.zip.ZipFile.<init>(ZipFile.java:85)
      at Create.if(Unknown Source)
      at Create.main(Unknown Source)

File cleanup: Retain object files?(Y or N)
```

6. *The build (BuildLib.jar) hangs after Extracting CorrectAddress(r) modules from archive...*

   Resolution:

   Delete **CAPort1.zip**, and re-run the build. This is an internal system issue.

7. *I get the following message:*

   Resolution:

   Specify the fPIC flag in the compile command and use the regular ld command for linking (without fPIC option)

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
<path>/libCorrectA.so: ld.so.1: <javapath>/java: fatal: relocation error: file
<path>/libCorrectA.so: symbol_udivdi3: referenced symbol not found
      at java.lang.ClassLoader$NativeLibrary.load(Native Method)
      at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1751)
      at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1676)
      at java.lang.Runtime.loadLibrary0(Runtime.java:822)
      at java.lang.System.loadLibrary(System.java:992)
      at javaCANativeDispatcher.<clinit>(javaCANativeDispatcher.java:348)

      at Test.main(javaCANativeDispatcher.java:373)
```

PERL- SPECIFIC ISSUES

1. *What is the procedure to re-build the PERL wrappers?*

   Resolution:

   a. Make sure **SWIG** is installed on your system.

   b. Type: **swig -perl5 CAPerl.i** (or other interface file). This will create **CAPerl_wrap.c**.

   c. Compile the wrapper C file **gcc -O -c -I<include dir>/CORE CAPerl_wrap.c** where <include dir> is location of PERL includes and can be retrieved by:**perl -e 'use Config; print $Config{archilib};'**

   d. Create shared object with wrapper object file ld **-G -o CAPerl.so *.o.**

   e. Run a PERL test that has a 'use CAPerl' and accesses CAPerl:: functions.

   IMPORTANT: Make sure two-dim arrays are declared in 1 dimension in the interface (.i file) (e.g. results[200][194] should be results[38800])

PHP- SPECIFIC ISSUES

1. *DPV processing does not work when invoked from the web browser. Error code 21 is returned.*

   Resolution:

   Set read/write permissions for user **apache** (or equivalent) on the *CorrectAddress* data directory. Particularly, **AFstreetsort.txt** is affected.

SQL- SPECIFIC ISSUES

1. *When running SQL Server extended stored procedures, addresses do not correct and an error code of "66" (out of memory) is returned.*

   Resolution:

   This may be due to the way SQL Server handles memory in excess of 2 GB. SQL Server sets aside by default at most 512 MB of RAM for internal processes and extended stored procedure DLLs, it then occupies the remaining available memory for its buffer pool. When running *CorrectAddress* in batch or with multiple processes, *CorrectAddress* may require more than the base 512 MBs set aside. To increase the amount of reserved memory, open Enterprise Manager and right click on the server you wish to configure, selecting Properties. Under the General tab there is a button marked Startup Parameters, press this to bring up a screen where you can input new parameters for SQL Server. To increase the memory, use the –g parameter which takes a number in MBs to determine the amount of RAM to reserve. Setting the memory to 1024 MBs (e.g. –g1024) will prevent future occurrences of this error.

Appendix A- PS Form 3553

Form 3553 is used when applying for Postal discount rates as explained in DMM A950, available at the Postal Service's website http://www.usps.gov. A sample of this file is located in your *CorrectAddress* installation directory in PS3553.pdf.

The fields and information required are as follows:

A1: CASS Certified Company Name	Experian Ltd
A2: CASS Certified Software Name and Version	*CorrectAddress* v9.0X.XX.A
A3: Configuration	Our configuration is STD (standard)
B1: List Processor's Name	The name of the company running this software.
B2: Date List Processed	The date the addresses were corrected, only input the master file and LOT section.
B3: Date of Database Product Used	The date for the database product, this can be found on a label of the *CorrectAddress* CD or DVD used for installation.
B4: List Name or ID	The internal identification for the list.
B5: Number of Lists Processed	How many lists of addresses were submitted for correction
B6: Total Records Processed	Total number of records.
C: Output	The sections here will ask for the total number of coded ZIPs, add-ons, carrier routes, etc. and their validation dates. The total numbers are shown when running the batch and the dates for each section are as follows. All From" dates are the date of the database product being used. Below are the "To" dates for each field. ZIP+4: 180 days from "From" date DPBC: 180 days from "From" date Five Digit: 365 days from "From" date Carrier Route: 90 days from "From" date LOT: 90 days from "From" date
D: Mailer	Contains mailing information, the address, and name of the company using this product, date this form was created, and a signature of an authorized company official.
E: Qualitative Statistical Summary	Contains values that can be found displayed after running the batch, such as total number of LACS converted addresses, total number of highrise default addresses, highrise exact addresses, rural route default and exact addresses.

Appendix B- Glossary of Postal Terms

Carrier Route

Code assigned by the USPS to a group of addresses to aid mail delivery within a ZIP code. Consists of a carrier route type and carrier route code (length: 4 bytes). For example, “C001”.

CASS

Coding Accuracy Support System. The Postal Service’s guidelines for address correction through which all address correction software must be certified.

Check Digit

Delivery point check digit is a number that is added to the sum of the other digits in the DPBC to yield a number that is a multiple of ten.

Cityname

Address city name (maximum length: 28 bytes).

CBSA

Core Bases (Metropolitan) Statistical Area, maintained by the U.S. Census Bureau . CBSAs are categorized as Metropolitan (at least one urbanized area of 50,000+ inhabitants) or Micropolitan (at least one urban cluster of between 10,000 and 50,000 inhabitants).

CMRA

Commercial Mail Receiving Agency. CMRAs are companies who offer mail services commercially to customers and are authorized to receive mail on behalf of their customer. Also see PBSA.

Congressional District Code

See FIPS codes

County Name

Name of county that the address belongs to (maximum length: 25 bytes). County is a local level of government below the state.

County Number

County number within a state (maximum length: 3 bytes).

Delivery Line 1

Primary delivery address line (maximum length: 64 bytes). Contains the house number, pre-directional, street name, street suffix, post-directional, secondary abbreviation, and secondary number. For example, “445 N Hamilton Ave Ste 608”.

Delivery Line 2

Second delivery address line (maximum length: 64 bytes). This line is usually reserved for dual addresses.

Delivery Point Alternate Records

Special address records containing alternate delivery points. E.g., different street name with the same address number as the base record, or different firm name associated with the base record at the same address as the base record. Similar to aliases.

DMM

Domestic Mail Manual. Contains all postal regulations for domestic mail. Available for browsing at www.usps.gov.

DPC
Delivery Point Code (length: 2 bytes). This field contains the last two digits of the house/box number, or if the match is made to a highrise record, the secondary unit number representing the delivery point information to form the 11-digit or delivery point barcode (DPBC). Possible values: “00” through “99” or spaces.

DPBC
Delivery Point Barcode. Created from the nine-digit ZIP code combined with the Delivery Point Code (DPC) and the Checkdigit sum. Used on mail pieces for hand held scanning.

DPV
Delivery Point Validation. See Appendix J.

Dual address

Address containing both a street portion and a PO Box or Rural Route address.

(E.g., 100 Main St, PO Box 123, Sometown, NY 11111)

Early Warning System

EWS is a file that lists by ZIP Code new street names that are not yet available within the ZIP + 4 product. Today, ZIP + 4 product is extracted from the Address Management System (AMS) approximately 30 days before its “official” release date. When address matches a record in the EWS file, a no-match is returned.

eLOT

Extended LOT.

False Positive Records

Control records placed by the USPS into the DPV and LACSlink databases to prevent unauthorized tampering (e.g., creation of lists containing every single delivery point in a geographical region). A match against a False Positive record activates a Stop Processing request, causing the software to disable DPV or LACSlink processes for all future addresses, until the process has been reactivated by the CASS vendor.

Error Codes

A string of one or more informational codes returned from a call to *CorrectAddress* function. Listing of error codes for variants of the standard CorrectA() function is provided in Appendix C.

Finance Number

Code assigned to USPS facilities (primarily post offices) to collect cost and statistical data and compile revenue and expense data. Used to better match addresses in which input ZIP codes are missing or incorrect, especially for cities that span multiple ZIP codes.

Firm/Recipient Name

Name of individual, company, building, apartment complex, shopping center, or other entity identifier (maximum length: 40 bytes). *CorrectAddress* returns corrected firm name or input firm name if no match found.

FIPS codes (state, county, congressional district)

Federal information processing standards codes (FIPS codes) are a standardized set of numeric or alphabetic codes issued by the National Institute of Standards and Technology (NIST) to ensure uniform identification of geographic entities through all federal government agencies. The entities covered include: states and statistically equivalent entities, counties and statistically equivalent entities, named populated and related location entities (such as, places and county subdivisions), and American Indian and Alaska Native areas.

Congressional district codes are 7 digit codes that consist of 2 digits representing the state, 2 digits representing the congressional district, and 3 digits designating the number of the Congress.

FSA

Forward Sortation Area. The first three characters of the Canadian postal code. FSA pinpoints a general area to which mail is delivered. The first character represents a province or territory or a portion within either.

General Delivery addresses

When postal customers pick up their mail at the local post office, it can be addressed as General Delivery. The USPS database contains special records for general delivery. Street name is “GENERAL DELIVERY” and the directional, suffix and secondary information fields are all blank. The add-on code for general delivery addresses is usually 9999.

Highrise addresses

Records that may be used to identify a commercial building, apartment complex, highrise, wing or floor of a building, group of apartment mail boxes, or physical location other than a street.

LACS

Locatable Address Conversion System. Data set provided to allow addresses that have been converted due to USPS changes or for 911 emergency systems to be linked with their new address. (See Appendix K for more details.)

LDU

Local Delivery Unit. The last three characters of the Canadian postal code. The LDU reveals a specific delivery point, such as a building, a large-volume receiver of mail, or a range of addresses on a street.

LastLine

Output city, state, and ZIP/+4 (maximum length: 64 bytes). For example: “White Plains, NY 10601-1827”.

LOT

Line of Travel. A code composed of a four digit sequence number and an associated code (“A” or “D” for ascending and descending respectively). Denotes the direction that mail is delivered in for an address (length: 4 bytes)

LVR

Large Volume Receiver. Term used by Canada Post to designate an entity subject to special address validation rules (length: 60 bytes).

Municipality

Term used by Canada Post to designate a city, town, village, township, borough, district or county (length: 30 bytes).

PBSA

P.O.Box Street Address. USPS street addresses that are equivalent to traditional P.O.Box-style addresses. They are flagged as CMRAs (see above) when verifying deliverability.

PMB

Private mailbox designation within a CMRA (maximum length: 12 bytes).

Point of Call (PoC) Address Data

Range-based data set with specific deliverable addresses within given Canadian postal codes. More accurate than Postal Code Address Data (PCAD)

Postal Code

Code system used by Canada Post (length: 6 bytes).

Postal Code Address Data (PCAD)

Range-based data set based on Canadian postal codes. Less accurate than Point of Call (PoC) Address Data.

Post-Directional

A directional abbreviation specified after the street name in an address. For example, 445 Hamilton Ave **S** (maximum length: 2 bytes).

Pre-Directional

A directional abbreviation specified before the street name in an address. For example, 445 **N** Hamilton Ave (maximum length: 2 bytes).

Preferred City Name

City name identified by the Postal Service as either the preferred city name at the 5-digit ZIP Code level, or the ZIP+4 level. In case of conflict between the two, the latter name takes precedence.

Primary Number

See Street Number.

Province

Administrative division within Canada. Analogous to **state** in the U.S.

Questionable Address

Canada Post defines a "questionable" address as one which is not complete or fully accurate, but in some instances may still be deliverable. An apartment building address is questionable if the mailing address does not have a unit number and there are no unit numbers available in the Canada Post database. A rural address is questionable if it cannot be validated based on all of the civic address components present and is therefore validated based on the Postal Code only.

Return Code

Integer return value of a *CorrectAddress* function. Listing of return codes for variants of the standard CorrectA() function is provided in Appendix C.

Secondary Abbreviation

The abbreviation of an apartment, suite, or other secondary housing name. Examples include: Apt, Ste, Fl, Rm (maximum length: 4 bytes).

Secondary Name

See Secondary Abbreviation

Secondary Number

The apartment/suite number of the address. For example, 445 Hamilton Ave Ste **608** (maximum length: 8 bytes).

State Code

State abbreviation (length: 2 bytes) within the U.S.

Street Name

The actual name of the street. For example, 445 N **Hamilton** Ave (maximum length: 28 bytes).

Street Number

The house number for the street (Primary number). For example, **445** Hamilton Ave (maximum length: 10 bytes).

Street Suffix

The abbreviation of the type of street. Examples include: Ave, St, Blvd (maximum length: 4 bytes).

TIGER/Line

Topographically Integrated Geographic Encoding and Referencing System – U.S. Census Bureau's geocoding database, containing features and statistical geographic areas

Unique ZIP code

Special ZIP code associated with a single high volume address where mail distribution is handled internally. Unique ZIP codes are subject to special USPS CASS rules.

Urbanization

Urban development within a geographic area (maximum length: 28 bytes). This is an additional delivery line input for addresses in Puerto Rico.

ZIP code

Zone Improvement Plan code (length: 5 digits) Postal code system used by the United States Postal Service.

ZIP Addon (+4)

Extension of a ZIP code which determines a more precise location within the ZIP. (length: 4 bytes)

Appendix C - Return Codes and Error Codes

The following return codes and error codes are used by variants of the standard **CorrectA** function, including **TigerCA**. Non-USPS codes are returned only by **CorrectAWorld** and **TigerCA.**

To determine whether a foreign address has been encountered, users must check the first 2 bytes of the errcode string for a match against a foreign country code below.

"30" – Canadian address

If the first two bytes match a foreign country code, the return code and remaining codes in the errcode strings will correspond to the codes particular to that country, as described in the tables below. If a foreign country code does not exist in the first two bytes of the errcode string, all return codes and error codes will match the descriptions for the USPS **CorrectA** codes provided in the section below.

USPS RETURN CODES

The Return Code is the integer value returned at completion of the validation process.

Return Code	Description
1	Match found; four-digit ZIP add-on assigned.
>1	Multiple possible results, but no exact match made. Number of results is the value of return code.
<0	Multiple possible results only when error code contains 11; no exact match made. Number of results is the absolute value of the return code.
-1	When error code contains 07, delivery point validation failed; five-digit ZIP returned.
-3	When error code contains 05 - PO Box, Rural Route or Highway contract; street name normalized though no match found.
-99	No match found, and the original input has been returned.

USPS ERROR CODES

The error codes field generated by *CorrectAddress* is a 30-character string in which 2-byte codes are placed going from left to right. Each 2-byte number refers to a specific code as detailed below.

Error Code	Description
00	**Address is Default Highrise or Rural Route.** This address matched to a default delivery record in a multi-unit building, or a rural/highway contract record with route number in the street name field. See Appendix B for information on highrise addresses.
01	**No match in 5-digit ZIP Code; match found in finance number.** Input ZIP code was incorrect. Correction applied successfully using city/state information provided. See Appendix B for information on finance numbers.
02	**ZIP Code add-on not found; replaced with correct add-on.** Input +4 code was incorrect. Correction applied successfully. See Appendix B for information on ZIP-Addon codes.
04	**City name corrected.** Input city name was incorrect. Correction applied successfully.
05	**PO Box, Rural Route or Highway Contract address standardized.** Input address was a box, rural route or highway contract address in a non-standard form (e.g., P.O.Box, or POBOX). Standardization performed successfully.
06	**Street number not precise match in street range; e.g. alphanumeric 10A within numeric range 1-99.** Input address contained extra characters in the street number. These characters were retained in the output.
07	**Address non-deliverable; no add-on assigned.** Delivery Point Validation (DPV) check failed.
08	**Secondary number is not precise match in secondary range; e.g. alphanumeric 10A within numeric range 1-99.** Input address contained extra characters in the secondary number. These characters were retained in the output.
09	**Address is Delivery Point Alternate.** See Appendix B for information on Delivery Point Alternate records.
10	**City is part of multiple counties.** More than one county name is listed for the address city name. Preferred county name is returned.
11	**No match; failed CASS multi-component rule; number of results is absolute value of return code.** Several problems were found in the input address. Unable to match using CASS logic. Number of near matches returned is equal to the absolute value of the return code.
12	**All highrise records returned; first result is the CASS-certified address; Z's follow last result.** Multiple records containing secondary ranges (apartment low – high numbers) returned in the *results* parameter. End of results is indicated by a string of ten "Z"s (ZZZZZZZZZZ). This error code always appears with error code "00". See Appendix B for information on highrise addresses. See Appendix F for result record layout.
13	**Military address.**
14	**Street address with appended apartment number.**

Error Code	Description
	Input address contained secondary information that could not be resolved through the CASS process. Secondary number was retained in the output.
15	**No match; near matches placed in results field; Z's follow last result.** Multiple records containing possible match candidates returned in the *results* parameter. End of results is indicated by a string of ten "Z"s (ZZZZZZZZZZ). This error code always appears with error code "99". See Appendix F for result record layout.
16	**Preferred city name used.** Input city name was changed to a preferred city name for this address. See Appendix B for information on preferred city names.
17	**The PO Box Only Delivery Zones** indicates there is only one ZIP for a given facility and that facility has no other form of postal delivery other than PO Box deliveries.
18	**DPV False Positive.** DPV process stopped, ZIP+4 codes will no longer be assigned until DPV has been reenabled. Each subsequent call will return error code "21". See Appendix B for information on False Positive records.
19	**No match; address found in Early Warning System database.** See Appendix B for information on the Early Warning System (EWS) database.
20	**Street name modified.** Input street name was incorrect. Correction applied successfully.
21	**DPV processing already stopped; please contact Support to restart the module.** See error code "18".
22	**LACS^{Link} processing already stopped; please contact Support to restart the module.** See error code "24".
23	**No match; no correlation between city and unique ZIP Code; 5-digit ZIP Code deleted.** Input record contained a unique ZIP code, input city did not match the ZIP. According to CASS rules for handling unique ZIP codes, 5-digit ZIP code was deleted. See Appendix B for information on Unique ZIP codes.
24	**LACS^{Link} False Positive.** LACSlink process stopped, ZIP+4 codes will no longer be assigned until LACSlink has been reenabled. Each subsequent call will return error code "22". See Appendix B for information on False Positive records.
30	**Foreign address.**
31	**Geocoder files missing or corrupt.**
39	**Failed 5-digit ZIP validation using city/state information** The output City, State and ZIP Code in LastLine do not correspond
40	**Multiple matches; incorrect post-directional.** Input address contained an incorrect post-directional abbreviation. Unable to correct automatically. Candidate records returned. See Appendix B for information on post-directionals.
41	**Multiple matches; incorrect pre-directional.** Input address contained an incorrect pre-directional abbreviation. Unable to correct automatically. Candidate records returned. See Appendix B for information on pre-directionals.

42	**Multiple matches; incorrect street suffix.** Input address contained an incorrect street suffix. Unable to correct automatically. Candidate records returned. See Appendix B for information on street suffixes.
43	**ZIP Code is PO Box or Rural Route only.** ZIP code contains no street records. Match is made to a PO Box, route or general delivery record.
44	**Apparent extraneous information removed.**

Error Code	Description
	Some input information was considered unnecessary and was removed.
45	**Street suffix modified.** Input address contained an incorrect street suffix. Correction applied successfully. See Appendix B for information on street suffixes.
46	**Street directional modified.** Input address contained an incorrect pre- or post-directional abbreviation. Correction applied successfully. See Appendix B for information on pre- and post-directionals.
47	**Address requires apartment/suite number; none input.** Missing secondary number.
48	**Multiple matches; would resolve with pre-directional.** Input address contained no pre-directional abbreviation. Unable to correct automatically. Candidate records returned. See Appendix B for information on pre-directionals.
49	**Multiple matches; would resolve with post-directional.** Input address contained no post-directional abbreviation. Unable to correct automatically. Candidate records returned. See Appendix B for information on post-directionals.
50	**Multiple matches; would resolve with street suffix.** Input address contained no street suffix. Unable to correct automatically. Candidate records returned. See Appendix B for information on street suffixes.
51	**Address does not require apartment/suite; none input.** This address matched to a default delivery record in a multi-unit building, or a rural/highway contract record with route number in the street name field. (See error code "00".) No secondary information was required. See Appendix B for information on highrise addresses.
52	**Address does not require apartment/suite; incorrect input.** This address matched to a default delivery record in a multi-unit building, or a rural/highway contract record with route number in the street name field. (See error code "00".) Secondary information was provided, but incorrect. See Appendix B for information on highrise addresses.
57	**Address not standardized; not enough information provided.**
58	**Address not standardized; invalid ZIP Code.** Input ZIP code was incorrect. Unable to standardize address.
59	**Address not standardized; belongs to US territory.** Unable to standardize address. State abbreviation indicates that the address belongs to one of the territories.
60	**Expired verification database; DPV/LACS^{LINK} processing disabled.** Postal data files expired. Address validation halted. Data expires 105 days after the build date. Build date can be obtained by calling **GetBuildDate** API function.

61	**House number not on street.** Input street number was incorrect.
64	**Data mismatch.** Postal data files do not match *CorrectAddress* library.
65	**Unable to open data file(s).** Postal data files are missing or corrupt.

Error Code	Description
66	**Out of memory.**
67	**Trial expired.**
68	**Invalid or missing license key.**
80	**RDI error: unable to open data files.** Residential Delivery Indicator lookup files are missing or corrupt.
81	**RDI error: out of memory.**

CANADA POST RETURN CODES

Return Code	Description
1	A non-ambiguous match was made to a Canadian address.
0	No match could be made for this address, nor could any near matches be found.
<0	No exact match could be resolved for this address. The number of near matches returned is equal to the absolute value of the return code.

CANADA POST ERROR CODES

Error Code	Description
01	Valid address
02	Correctable address - reformatted
03	Invalid / noncorrectable address
05	Civic address
06	Civic address with route service
07	PO Box address
08	Route Service address
09	General Delivery address
11	Large Volume Receiver address type A
12	Large Volume Receiver address type B
13	Large Volume Receiver address type C
14	Large Volume Receiver address type D
15	Large Volume Receiver address type E
16	Large Volume Receiver address type F
18	Municipality incorrect

Error Code	Description
19	Postal code invalid or missing and matching address has wrong municipality
20	Invalid province name / abbreviation
21	Postal code invalid
23	Civic number out of range
24	Missing unit number
25	Invalid unit number
26	Invalid civic suffix
27	Invalid street
28	Street name typo
29	Unit number in front of civic without a dash
30	Civic suffix with space
31	Invalid unit designator
32	Street type missing / invalid
33	Street type typo
34	Street direction missing
35	Street direction invalid

36	Street direction typo
37	Route information missing on civic route service address
38	Route service removed from PoCAD managed rural civic
39	PO Box number out of range
40	PO Box identifier typo
42	Route type identifier typo
43	Route type incorrect
44	Route Service number incorrect and missing postal code
45	Route Service number incorrect
47	General Delivery indicator typo
49	Delivery Installation type typo
50	Delivery Installation name missing
51	Delivery Installation type missing
52	Delivery Installation name incorrect
53	Delivery Installation info missing
55	Pound sign removed
56	Postal code missing
58	Warning - address invalid for corresponding LVR postal code - address deemed valid based on postal code
59	Warning - correctable address for corresponding LVR postal code
61	Warning - address invalid for corresponding rural postal code

Error Code	Description
62	Warning - correctable address for corresponding rural postal code - address deemed valid based on postal code
63	PoCAD excluded rural civic
64	Postal Code changed from Rural to Urban
65	Postal Code corrected
66	Postal Code LDU changed
67	Postal Code FSA changed
68	Valid Questionable - incoming address has no unit number and there is no unit number in CPC
69	Valid Questionable - incoming address does not contain a recognizable delivery mode
70	Valid Questionable - incoming address delivery mode does not exist in Postal Code
71	Processed using Point of Call (PoC) Address Data
72	Processed using Postal Code Address Data (PCAD), even though Point of Call (PoC) data was requested

Appendix D - Listing of Correct Address Data Files

STANDARD

ZIP+4 data files:

strname.txt, firm.txt, nskey.txt, zByFin.dat, zByFin.idx, zToKey.idx, keyToStreet[0-9].idx, streetToZ4[0-9].idx, z4[0-9].dat, z4f.dat, unique.txt

City/State data files:

alias.dat, aliasname.txt, county.txt, countyByState.txt, ctystate.idx, detail.dat, findurbkey.dat, findurbname.dat, pobzone.txt

eLOT data file:

ltravel.wrk

EWS data file:

ews.txt

DPV data files:

dph.hsa.z[00-99], dph.hsc, dph.hsf, dph.hsp, dph.hsv, dph.hsx, lcdadd.dat, lcdzip.idx, dph.hsn,dph.hsu,dph.hst,dph.hsy,dph.hsz,dph.hsr.zall

LACS^{Link} data files:

llk.hs[1-6], llk.hsl, llk_leftrite.txt, llk_pno.dat, llk_sno.dat, llk_dsc.dat, llk_hint.txt, llk_nam.dat, llk_suf.dat, llk_crd.dat, llk_lln.dat, llk_rv9.esd, llk_strname.dat

SuiteLink data files:

slk.asc, slknine.lst, slknoise.lst, slknormal.lst

ZIPmove data files:

zipmove.idx, zipmove.txt

ADD-ONS

Geocoding Files:

ZIP+4 level - tiger[0-9]c.txt

Address (rooftop) level (pre-2010) - geocoder\[strdir.txt, strsuf.txt, streets.txt, gc[01-72].bin, and gc[01-72].bnd]

Address (rooftop) level - geocoder\2010\[addr_[0-9].idx, addr_[0-9].dat,coords_[0-9].dat],zip_centroid.dat,zip_centroid_codes.dat

Canadian data files:

dell_inst_mult, lvr_zip_add[1-6], lvr_zip_add[1-6].ind, ref_add[1-5], ref_add[1-5].ind, zip_add[1-5], zip_add[1-5].ind, mcr, mcr.ind, scr, scr.ind, buildinfo, pcm, ref_poc[1-5],ref_poc[1-5].ind, zip_poc[1-5],zip_poc[1-5].ind

fmt\
dell_inst_mult.fmt, dcr_[A-F].fmt, ref_add[1-5].fmt, zip_add[1-5].fmt, mcr.fmt, scr.fmt,pcm.fmt,ref_poc[1-5].fmt, zip_poc[1-5].fmt

sets\
altstreetcodes.set, dellinsttypes.set, dir.common.set, dir.set, dir.strict.set, dircodes.set, errors.set, gd.set, pobox.set, provcodes.set, provinces.set, provinces.strict.set, rr.set, streetcodes.common.set, streetcodes.set, streetcodes.strict.set, type2.set, unitcodes.set, unitcodes.strict.set

Appendix E - Postal Discount Rates

An online tutorial to the preparation of business mail can be found at the following web site:

https://pe.usps.com/BusinessMail101/index

For a complete price list, go to :

http://pe.usps.com/cpim/ftp/manuals/dmm300/ratesandfees.pdf

Appendix F - Results Record Layout

U.S. RESULTS LAYOUT

Field Number	Field Description	Bytes	Position Start/End	Notes
1	Detail Code	01	01/01	D = Detail
2	ZIP Code	05	02/06	
3	Record Type Code	01	18/18	F = Firm G = General Delivery H = Highrise P = PO Box R = Rural Route/Highway Contract S = Street
4	Carrier Route ID	04	19/22	
5	Street Pre-directional Abbreviation	02	23/24	N,W,S,E,NW,SW,NE or SE
6	Street Name	28	25/52	
7	Street Suffix Abbreviation	04	53/56	
8	Street Post-directional Abbreviation	02	57/58	N,W,S,E,NW,SW,NE or SE
9	Street Number Range Low (From)	10	59/68	
10	Street Number Range High (To)	10	69/78	
11	Street Number Odd/Even Code	01	79/79	(O)dd, (E)ven or (B)oth
12	Building or Firm Name	40	80/119	
13	Secondary Abbreviation	04	120/123	Only record types F & H
14	Secondary Number Range Low (From)	08	124/131	Only record types F & H

Field Number	Field Description	Bytes	Position Start/End	Notes
15	Secondary Number Range High (To)	08	132/139	Only record types F & H
16	Secondary Number Odd/Even Code	01	140/140	Only record types F & H
17	ZIP Addon Number Range Low (From)	04	141/144	
18	ZIP Addon Number Range High (To)	04	145/148	

19	Base/Alternate Record Code	01	149/149	B = Base; A = Alternate
20	LACS Status Indicator	01	150/150	L - LACS Converted Blank - not available
21	Finance Number	06	152/157	
22	State Abbreviation	02	158/159	
23	County Number	03	160/162	
24	Urbanization City State Key	06	171/176	
25	Preferred Last Line City State Key	06	177/182	
26	Filler	12	183/194	Filler-Do Not Use

CANADIAN RESULTS LAYOUT

Field Number	Field Description	Bytes	Position Start/End	Notes
1	Province Code	02	1/2	
2	Municipality Name	30	3/32	
3	Postal Code	06	33/38	
4	Street Name	30	39/68	
5	Street Type Code	06	69/74	
6	Street Directional Code	02	75/76	
7	Street Address Sequence Code	01	77/77	
8	Street Address High Number (To)	06	78/83	
9	Street Address Number Suffix Low Number (To)	01	84/84	
10	Suite High Number (To)	06	85/90	
11	Street Address Low Number (From)	06	91/96	
12	Street Address Number Suffix High Number (From)	01	97/97	
13	Suite Low Number (From)	06	98/103	
14	Lock Box Bag High Num (To)	06	104/109	

15	Lock Box Bag Low Number (From)	06	110/115	
16	Route Service Type Description	02	116/117	
17	Route Service Number	04	118/121	
18	Delivery Installation Type Description	05	122/126	
19	Delivery Installation Qualifier Name	15	127/141	
20	Building Name	30	142/171	
21	Building Type Code	01	172/172	
22	LVR Name	60	173/232	

23	Department Name	30	233/262	
24	Branch Name	30	263/292	
25	Language Code	01	293/293	
26	General Delivery Description	60	294/353	
27	Filler	01	354/354	

Appendix G - Batch Processor Configuration

This section describes configuration options for running the Windows batch processor executable (CABatch.exe). The program can run in both single-threaded and multi-threaded modes. For latter mode, a processor configuration file (CABatch.exe.config) must be present in the local directory.

PROCESSOR CONFIGURATION FILE

The default processor configuration file located in the CorrectAddress installation directory is set up as follows:

```
<userSettings>
  <CABatch.My.MySettings>
    <setting name="MaxAddressQueueLists" serializeAs="String">
      <value>10</value>
    </setting>
    <setting name="MaxAddressesPerQueueList" serializeAs="String">
      <value>100</value>
    </setting>
    <setting name="MaxConsumerThreads" serializeAs="String">
      <value>0</value>
    </setting>
    <setting name="RunThreaded" serializeAs="String">
      <value>False</value>
    </setting>
  </CABatch.My.MySettings>
</userSettings>
```

Setting **RunThreaded** to **True** enables multi-threaded processing (this option is currently available for TEXT FILE PROCESSING ONLY).

MaxAddressQueueLists specifies the number of groups of addresses to keep in memory. This value should be set to the number of cores + 2.

MaxAddressesPerQueueList specifies the number of addresses in each address queue list in memory. This value should be kept at 100.

MaxConsumerThreads specifies the number of threads to run. This value should be set to the number of cores or (number of cores – 2).

RUNNING BATCH PROCESSOR AT COMMAND-LINE PROMPT:

cd **C:\Program Files\Intelligent Search Technology\CorrectAddress**

C:\Program Files\Intelligent Search Technology\CorrectAddress>CABatch.exe MyConfigFile.ini

Batch job configuration file (e.g., MyConfigFile.ini) can be created using the Batch Utility wizard in the Graphical Interface (CorrectGUI.exe), or manually by the user. Below is a user's guide that describes job configuration files in detail.

BATCH JOB CONFIGURATION FILES

GENERAL ORGANIZATION

[CorrectAddress Configuration File]

[THREAD] - contains paths to store temporary process statistics

[INPUT] - contains information about the data source

[OPTIONS] - contains run-time options, such as add-ons, mixed case conversion etc.

[OUTPUT] - contains information about the data destination

[ADDRESSVALUES] - section header

[RET FIELDS] - contains list of retained fields (optional)

[FIRM] - list of fields containing firm name / recipient matching (optional)

[URBANIZATION] - list of fields containing urbanization information (optional)

[DLINE1] - list of fields containing delivery line information

[DLINE2] - list of fields containing additional delivery line information (optional)

[LASTLINE] - list of fields containing last line information (e.g., city, state, ZIP)

[END ADDRESSVALUES] - section terminator

SECTIONS

[THREAD]

```
THREAD0
0
0
UpdateProcessPath: C:\Program Files\Intelligent Search
Technology\CorrectAddress\UpdateProcessPath1.ini
CancelProcessPath: C:\Program Files\Intelligent Search
Technology\CorrectAddress\CancelProcessPath1.ini
ErrorLogPath: C:\Program Files\Intelligent Search
Technology\CorrectAddress\ErrLogPath0.ini
```

This section contains thread name, start and end record counts and paths to three files containing temporary process statistics..

[INPUT]

Sample - text input

```
[INPUT]
Type: Text
Input: C:\SampleRecords\MyInputFile.txt
Format: Delimited
TEXTDELIM: "
DELIM: ,
ColHeader: True
Cols: 8
EmployeeID
```

```
FirstName
MiddleName
LastName
Address1
City
State
ZIP
```

Sample - database input

```
[INPUT]
Type: SQLServer
ConnType: 2
Server: (local)
Port: 1433
Database: SAMPLEDATABASE
Username: sa
Password: mypassword
OLEDBname: sqloledb
ODBCname: SQL Server
ConnString: Data Source=MyServerName;Initial Catalog=MyDatabaseName;User
ID=MyUsername;Password=MyPassword
IPPortBool: False
WindowsBool: False
ConnStringBool: False
TableType: Table
TableName: INPUTADDRESSES
Cols: 12
EmployeeID=0-9
LastName=10-29
FirstName=30-39
Title=40-69
TitleOfCourtesy=70-94
BirthDate=95-117
HireDate=118-140
Address1=141-200
City=201-215
State=216-230
ZIP=231-240
Country=241-255
```

The input section contains the following settings:

- Input type (Text, SQLServer, Oracle, Access)
- Input file or database location
- File format : DELIMITED or FIXEDWIDTH
- Text delimiters and qualifiers (for delimited files)
- Column headers (true/false)
- Total number of fields/columns and their listing

Database input fields and fixed-width text input must contain field start-end positions as shown in the database example above.

[OPTIONS]

```
[OPTIONS]
Geo: False
NoMatch: True
MixedCase: False
IParse: False
PS3553 TEXT: False
PS3553 XML: False
Codes:
Country: 1
```

The following run-time switches are available: Geocoder (GEO), Output No-Match records (NOMATCH), proper case conversion (MIXEDCASE), IParser (IPARSE), generation of text-based and XML-based CASS reports (forms PS3553), custom options described in Appendix I (CODES) and the country indicator (0 for auto-detect, 1 for USA, 2 for Canada).

[OUTPUT]

Sample - text output

```
[OUTPUT]
Type: Text
Output: C:\SampleRecords\MyOutputFile.txt
Format: Delimited
TextDelim: "
Delim: ,
ColHeader: true
Cols: 31
RET 0:EmployeeID
RET 0:LastName
RET 0:FirstName
CA 0:Recipient
CA 1:Urbanization
CA 2:Delivery Line 1
CA 3:Delivery Line 2
CA 4:Last Line
CA 5:Street Number
CA 6:Pre-Directional
CA 7:Street Name
CA 8:Street Suffix
CA 9:Post-Directional
CA 10:Secondary Designation
CA 11:Secondary Number
CA 12:City Name
CA 13:State Abbreviation
CA 14:ZipWithAddon
CA 15:Zip5
CA 16:Addon
```

```
CA 17:LOT Number
CA 18:DPC
CA 19:Checkdigit
CA 20:Record Type
CA 21:LACS
CA 22:Carrier Route
CA 23:PMB
CA 24:County Name
CA 25:County Number
CA 26:Return Code
CA 27:Error Codes
```

Sample - database output

```
[OUTPUT]
Type: SQLServer
ConnType: 2
Server: (local)
Port: 1433
Database: SAMPLEDATABASE
Username: sa
Password: mypassword
OLEDBname: sqloledb
ODBCname: SQL Server
ConnString: Data Source=MyServerName;Initial Catalog=MyDatabaseName;User
ID=MyUsername;Password=MyPassword
IPPortBool: False
WindowsBool: False
ConnStringBool: False
TableType: Table
TableName: OUTPUTADDRESSES
Cols: 31
RET 0:EmployeeID
RET 0:LastName
RET 0:FirstName
CA 0:Recipient
CA 1:Urbanization
CA 2:Delivery Line 1
CA 3:Delivery Line 2
CA 4:Last Line
…
{et cetera, as shown in the text output above)
```

This section is similar to the input section with a notable exception of field naming conventions. Retained fields are preceded with "RET 0", *CorrectAddress* output fields are preceded with "CA #", where # uniquely identifies field contents. E.g., "CA 2" will always mean "Delivery Line 1", but the field can be renamed to something else at user's discretion. To do so, simply change the description following the colon. For example, change "CA 2: Delivery Line 1" to

CA 2: AddressLine1

The effect of this change is that the output file/table will have a field name "AddressLine1" containing primary delivery line information.

[ADDRESSVALUES]

```
[ADDRESSVALUES]
[RET FIELDS]
EmployeeID
LastName
FirstName
[FIRM]
[URBANIZATION]
[DLINE1]
Address1
[DLINE2]
[LASTLINE]
City
ZIP
[END ADDRESSVALUES]
```

This section contains information about input field mappings. [RET FIELDS] subsection contains a listing of retained fields, [FIRM] subsection contains list of fields to be used for firm/recipient matching, [URBANIZATION] contains fields with urbanization information (used only in Puerto Rican addresses), [DLINE1] and [DLINE2] subsections contain delivery line information, and the [LASTLINE] contains city, state and postal code (ZIP) fields. [END ADDRESSVALUES] serves as section terminator.

ADDITIONAL OUTPUT FIELDS

Geocoder fields

GEO 0:Geo TLID

GEO 1:Geo Misc Data

GEO 2:Geo Tract

GEO 3:Geo Block

GEO 4:Geo From Latitude

GEO 5:Geo To Latitude

GEO 6:Geo From Longitude

GEO 7:Geo To Longitude

GEO 8:Geo Addon Start

GEO 9:Geo Addon End

GEO 10:Geo Return Code

GEO 11:Geo Error Codes

DPV fields

DPV 0:DPV Flags

DPV 1:DPV Footnotes

DPV 2:DPV Vacant

DPV 3:DPV PBSA

LACSlink fields

LL 0:LACS Code

LL 1:LACS RetCode

RDI fields

RD 0:RDI

SuiteLink fields

SL 0: SuiteLink

ENABLING RUNCABATCH SUPPORT FOR CANADA DATA

This configuration setting allows RunCABatch for Canada data. These settings only apply to RunCABatch API. If the configuration is not in the configuration file, it will use USA as default.

[COUNTRY]
ALL

[USADATAPATH]
<Path to USA Data>

[CANADADATAPATH]
<Path to CAN Data>

[USAGEODATAPATH]
<Path to Tigerline Data>

Note: [COUNTRY] could have values USA, CAN, ALL. (ALL means both CAN and USA)

Appendix H- Geocoding

Geocoding support allows *CorrectAddress* users to retrieve information from the Census Bureau and match it to the USPS data based on address information and ZIP+4. Along with a validated address, the module returns geographical coordinates (latitude and longitude), Census tract and block numbers and more. The geocoding functionality is accessible via options in the Windows Graphical User Interface as well as via three exported functions: **TigerCA**, **getCentroid** and **GeoCode.** Interfacing to these functions is described in further detail in the API chapter earlier in the manual.

GEOCODING ERROR CODES / FOOTNOTES

00 – ZIP code is invalid.

01 – Out of memory.

02 – Failed to load pre-2010 TIGER/Line data

03 – Error opening data connection.

04 – No record for ZIP+4.

05 – No record for specified ZIP+4; near ZIP+4 match.

06 – No record for specified ZIP+4; Highrise default ZIP +4 used.

07 – No record for specified ZIP+4; near Street ZIP +4 used.

08 – No record for specified ZIP+4; near street ZIP +4 used.

09 – No address level record exists; attempting to use ZIP+4.

20 – Match found in latest TIGER/Line data

22 – Failed to load latest TIGER/Line data

26 – No matches found in latest TIGER/Line data

30 – ZIP centroid coordinates returned (no match in street level latest TIGER/Line data)

GEOCODING CONVERSION

With geocoding, the latitudes and longitudes are returned in decimal degree format. You may convert to HR/MIN/SECONDS using the following formula:

1. The whole units of degrees will remain the same (i.e. in +38.897011 longitude, start with 38°).
2. Multiply the decimal by 60 (i.e. .897011 * 60 = 53.82066). The whole number becomes the minutes (53').
3. Take the remaining decimal and multiply by 60. (i.e. .82066 * 60 = 49.2396). The resulting number becomes the seconds (49"). Seconds can remain as a decimal.
4. Take your three sets of numbers and put them together, using the symbols for degrees (°), minutes ('), and seconds (") (i.e. 38°53'49" longitude).

Appendix I - Custom Options

These custom options can be enabled for variants of the **CorrectA** function through the *CorrectAddress* API. To set the various options a flag must be passed in the errcode argument prior to the call to **CorrectA** or its variants. All flags are two-byte alphabetic codes where the first character is always upper case and the second lower case. To be effective, flags must be appended consecutively from the beginning of the errcode string. The relative order of the flags does not matter. Common customization flags and their descriptions are listed below.

Aa Abbreviate always. Always prefer abbreviated name if otherwise specified. The option has no effect by itself but combined with As will return the abbreviated street name, no matter how long the street line in the result is.

Ac Abbreviate city. Return abbreviated city name if available.

As Abbreviate street. Return abbreviated street name if the street line is longer than 30 characters. Otherwise, the form best matching the input is preferred.

Cd Search only the Canadian address database. Can be used in CorrectAWorld and CorrectAOracle functions.

Db Run software in debug mode, generates **CAdebug.log** in the program directory.

Df Flip input delivery lines in a dual address

Ln Disable logging (No log files are created in the \Data directory)

Mc Display output in mixed case format.

Mx Return secondary information on the line it was entered.

My Return secondary information on the second line.

Pc Return preferred city name. (Override input city name, even if valid for mailing.) If preferred name for the specific ZIP+4 record is different from the preferred city name for the ZIP code, the former name takes precedence.

Po Enable Point of Call (PoC) Address Data processing for Canadian addresses.

Ra Return alias street name. (Override preferred street name.)

Rd Enable RDI processing. (RDI data must be installed. See Appendix L.)

Ro Retain dropped address information in a dual address. Discarded address data will be written to the Stringaddress variable starting at position 147, ending at 176 which will lead to replacing Non-Delivery Day Ind, Non-Delivery Day Val, No Secure Location, Door Not Accessible and Enhanced DPV Code.

Ry Parse and standardize address on no-match. Parsed values will be written to the Stringaddress variable starting at position 21 and formatted as follows:

Pre-directional (2 bytes) Street name (28 bytes) Suffix (4 bytes)

Post-directional (2 bytes)

Us Search only the USPS database (United States and territories). Can be used in CorrectAWorld and CorrectAOracle functions.

Zd Read postal data from disk (recommended for machines with < 1GB RAM).

Zf Load all postal data into memory (instead of default load-on-demand). EXAMPLE:

The example below shows how to enable mixed case output and force preferred city name output by passing the appropriate flags in the errcode argument.

```
/* ... */
char errcode[30];
/* ... */
cpy_word(errcode,"McPc",4);

rc = CorrectA(inputAddress, sentLen, errcode,...);
```

AppendixJ- DeliveryPointValidation(DPV™)

In addition to being certified for standard ZIP+4® processing, *CorrectAddress* supports Delivery Point Validation (DPV™). DPV takes the verification process one step further and authenticates the address as an actual delivery point. With DPV checking enabled, *CorrectAddress* is capable of confirming over 145 million physical mail delivery points throughout the United States and its territories. The DPV component will also determine if the address belongs to a Commercial Mail Receiving Agency (CMRA) and provide other useful information to indicate match quality.

DELIVERY POINT VALIDATION INDICATORS

When the validation process is complete, DPV results appear in the Stringaddress variable in the following format:

Field Number	Field Description	Bytes	Position Start/End
1	DPV Confirmation	01	01/01
2	CMRA	01	02/02
3	False Positive	01	03/03
4	No-Stat	01	04/04
5	DPV Footnotes	10	05/14
6	No-Stat Reason Code	02	15/16
7	Vacant	01	17/17
8	PBSA	01	18/18
9	Drop	01	19/19
10	Throwback	01	20/20
11	Non-Delivery Day Ind	01	166/166
12	Non-Delivery Day Val	07	167/173
13	No Secure Location	01	174/174
14	Door Not Accessible	01	175/175
15	Enhanced DPV Code	01	176/176

DELIVERY POINT VALIDATION (DPV) CONFIRMATION INDICATOR

Field contains the results of the call to the DPV Confirmation Hash Table: **dph.hsa**

Return values:

Y – Address DPV confirmed for both primary and (if present) secondary numbers

D – Address DPV confirmed for the primary number only, and secondary number information was missing

S – Address was DPV confirmed for the primary number only and the secondary number information was present but invalid, or a single trailing alpha on a primary number was dropped to make a DPV match.

N – Primary number failed to DPV confirm.

Blank – Address not presented to hash table.

DELIVERY POINT VALIDATION (DPV) CMRA INDICATOR

Field contains the results of the call to the DPV CMRA Hash Table: **dph.hsc**

DPV CMRA Table contains CMRA Addresses (see Glossary).

Return values:

Y – Address found in CMRA table.

N – Address not found in CMRA table.

Blank – Address not presented to hash table.

DELIVERY POINT VALIDATION (DPV) FALSE POSITIVE INDICATOR

Field contains the results of the call to the DPV False Positive Hash Table: **dph.hsf**

Return values:

Y – Address found in False Positive table.

N – Address not found in False Positive table.

Blank – Address not presented to hash table.

DELIVERY POINT VALIDATION (DPV) NO-STAT INDICATOR

Field contains the results of the call to the DPV No-Stat Hash Table: **dph.hsx**

DPV No-Stat Table contains addresses that are not receiving delivery and not counted as a possible delivery. These addresses are not receiving delivery because a) delivery has not been established; b) customer receives mail as a part of a drop; or c) the address is no longer a possible delivery because the carrier destroys or returns all of the mail. Addresses for delivery points in gated communities may also be identified as No-Stats.

Return values:

Y – Address found in No-Stat table.

N – Address not found in No-Stat table.

Blank – Address not presented to hash table.

DESCRIPTION OF DELIVERY POINT VALIDATION (DPV) FOOTNOTES

AA – Input Address Matched to the ZIP+4 file

A1 – Input Address Not Matched to the ZIP+4 file

BB – Input Address Matched to DPV (all components)

CC – Input Address Primary Number Matched to DPV but Secondary Number not Matched (present but invalid)

N1 – Input Address Primary Number Matched to DPV but Address Missing Secondary Number

M1 – Input Address Primary Number Missing

M3 – Input Address Primary Number Invalid

P1 – Input Address RR or HC Box number Missing

P3 – Input Address PO, RR, or HC Box number Invalid

PB – Input Address Matched to a PBSA Record (Carrier Route C770 through C779)

RR – Input Address Matched to CMRA and PMB designator present (PMB 123 or #123)

R1 – Input Address Matched to CMRA but PMB designator not present (PMB 123 or #123)

R7 – Addresses that are assigned to a phantom route of R777 or R779

F1 – Input Address Matched to a Military Address

G1 – Input Address Matched to a General Delivery Address

U1 – Input Address Matched to a Unique ZIP Code

TA – Input address primary number matched by dropping trailing alpha

IA – Informed address identified

C1 – Input address primary number matched, secondary number not matched; secondary number required

DELIVERY POINT VALIDATION (DPV) No-STAT REASON CODE

Field contains the results of the call to the Hash Table: Dph.hsr.zall

Provides details as to why records are flagged as No-Stats.

Return values:

1 – IDA (Internal Drop Address) – Addresses that do not receive mail directly from the USPS, but are delivered to a drop address that services them.

2 – CDS No-Stat – Addresses that have not yet become deliverable. For example, a new subdivision where lots and primary numbers have been determined, but no structure exists yet for occupancy.

3 – Collision – Addresses that do not actually DPV confirm. In this case, the 'Y' should be set to 'N' on the DPV 'A' table and all other table values should be blank.

4 – CMZ (College, Military and Other Types) – ZIP + 4 records USPS has incorporated into the data.

5 – Regular No-Stat – Indicates addresses not receiving delivery and the addresses are not counted as possible deliveries.

6 – Secondary Required - The address requires secondary information.

DELIVERY POINT VALIDATION (DPV) VACANT INDICATOR

Field contains the results of the call to the DPV Vacant Table: **dph.hsv**

DPV Vacant Table contains delivery points that were active in the past, but are currently vacant (in most cases unoccupied over 90 days) and not receiving delivery.

Return values:

Y – Address listed in the table of vacant addresses

N – Address not listed in the table of vacant addresses

Blank – Address not presented to hash table

DELIVERY POINT VALIDATION (DPV) PBSA INDICATOR

Field contains the results of the call to the DPV PBSA Table: **dph.hsp**

DPV PBSA Table contains PO Box Street Addresses, or PBSAs (see Glossary).

Return values:

Y – Address listed in the table of PBSA addresses

N – Address not listed in the table of PBSA addresses

Blank – Address not presented to hash table

DELIVERY POINT VALIDATION (DPV) DROP INDICATOR

Field contains the results of the call to the Hash Table: dph.hsd

Flag indicates mail is delivered to a single receptable at a site

Return values:

Y - Address was found in the table

N - Address was not found in the table

Blank - Address was not presented to the table

DELIVERY POINT VALIDATION (DPV) THROWBACK INDICATOR

Field contains the results of the call to the Hash Table: dph.hst

Mail is not delivered to the street address

Return values:

Y - Address was found in the table

N - Address was not found in the table

Blank - Address was not presented to the table

DELIVERY POINT VALIDATION (DPV) NON-DELIVERY DAY FLAG

Field contains the results of the call to the Hash Table: **dph.hsy**

Flag indicates mail delivery is not performed every day of the week

Return values:

Y - Address was found in the table

N - Address was not found in the table

Blank - Address was not presented to the table

NON-DELIVERY DAY VALUE

Field contains the results of the call to the Hash Table: **dph.hsz**

Indicates which days mail is not delivered to the address

DELIVERY POINT VALIDATION (DPV) NO SECURE LOCATION

Field contains the results of the call to the Hash Table: **dph.hsu**

Flag indicates door is accessible, but package will not be left due to security concerns

Return values:

Y - Address was found in the table

N - Address was not found in the table

Blank - Address was not presented to the table

DELIVERY POINT VALIDATION (DPV) DOOR NOT ACCESSIBLE

Field contains the results of the call to the Hash Table: **dph.hsn**

Flag indicates addresses where USPS cannot knock on a door to deliver mail

Return values:

Y - Address was found in the table

N - Address was not found in the table

Blank - Address was not presented to the table

DELIVERY POINT VALIDATION (DPV) ENHANCED RETURN CODES

Return values:

Y - Address was DPV confirmed for primary/secondary components necessary to determine a valid delivery point.

D - Address was DPV confirmed for the primary number only, and the secondary number information was missing.

S - Address was DPV confirmed for the primary number only, the secondary number information was present but not confirmed or a single trailing alpha on a primary number was dropped to make a DPV match and secondary information required.

N - Primary number failed to DPV confirm.

R - Address confirmed but assigned to phantom route R777 or R779 and USPS delivery is not provided.
Blank - Address not presented to hash table.

Appendix K- LACS$^{Link™}$

CorrectAddress supports LACS$^{Link™}$. LACS^{Link} stands for **L**ocatable **A**ddress **C**onversion **S**ystem **Link**. It allows addresses that have been converted due to various USPS changes to be linked with their new addresses. This affects many of rural-style U.S. addresses that have been assigned city-style street names for 911 emergency response systems. Additionally, LACS^{Link} covers street names that have been modified by municipalities in recognition of an individual or an event.

With LACS^{Link} processing enabled, the *CorrectAddress* engine will standardize and verify the input address, and will return a LACS converted counterpart, whenever it is applicable.

LACS$^{LINK™}$ RETURN CODES

When the validation process is complete, LACS^{Link} return codes appear in the last 3 bytes of the Stringaddress variable (starting at position 254) in the following format:

Field Number	Field Description	Bytes	Position Start/End
1	LACS Indicator	01	254/254
2	LACS Code	02	255/256

Description of Return Codes

LACS Indicator	Code	Description
Y	A	LACS Record Match – The input record matched to a record in the master file. A new address could be furnished.
N	00	No Match – The input record COULD NOT BE matched to a record in the master file. A new address could not be furnished.
Y	14	Found LACS Record: New Address Would Not Convert at Run Time – The input record matched to a record in the master file. The new address could not be converted to a deliverable address.
S	92	LACS Record: Secondary Number Dropped from Input Address – The input record matched to a master file record, but the input address had a secondary number and the master file record did not. The record is a ZIP+4 street level or highrise match.
F		False Positive: Address matched to a false positive.

Appendix L - Residential Delivery Indicator (RDI™)

The Residential Delivery Indicator (RDI™) add-on enables users to determine whether a given address is classified as a residential or a business address.

The RDI process may be run directly or as part of a standard address lookup.

Users can run RDI directly via the isBusinessZip API, as shown in the example below. The required input parameters are a 9-digit ZIP Code or 11-digit DPC, the corresponding length (9 or 11), and the path to the two RDI lookup tables.

```
int rc;
char zip = "106011827"
int length = 9;
char file9_path = "C:/ MyRDIDataPath/rts.hs9";
char file11_path = "C:/MyRDIDataPath/rts.hs11";

rc = isBusinessZip(zip,length,file9_path,file11_path);
```

The return code indicates whether the ZIP Code was determined to be a business address. (See the return code descriptions below.) Negative values indicate processing errors.

0 – Residential

1 – Business

2 – Mixed

-1 – Failed to allocate memory for lookup table.

-2 – Failed to open lookup file.

-3 – Failed to read lookup file.

-4 – Lookup table size invalid.

RDI processing can be enabled as part of a standard address lookup. To enable RDI when making a call to **CorrectA** or a similar function, append the RDI flag "Rd" to the errcode argument before making the call. The result of the RDI lookup will be indicated in the last character of the Stringaddress argument (Stringaddress[259]). A 'Y' indicates a confirmed residential delivery point, and an “N” indicates a non-residential address. The blank character indicates that an RDI lookup could not be performed due to a ZIP + 4 mismatch.

Appendix M - SuiteLink™

The SuiteLink™ add-on product enables *CorrectAddress* users to append secondary (suite) information to a business address provided that the input address is determined to be a highrise default record. (See *Appendix B* for details on highrise default addresses.)

SuiteLink processing is enabled automatically as part of a standard address lookup. The result of the SuiteLink lookup will be indicated in the 246-th byte of the Stringaddress parameter (Stringaddress[245]). A 'Y' indicates a SuiteLink match, where secondary information is automatically appended to the output address. An 'N' indicates no match. A blank (' ') indicates that no lookup was performed.