Building Data Sets for Portfolio Explorer

# Introduction

PortfolioExplorer needs a certain, specific set of data files, and parsing tables in order to correctly perform its functionality. While the parsing tables are basically built by hand, the data files need to be created using an executable built by the geocoder\_loaders sub project in the open source repository. Using a Microsoft Visual Studio IDE and compiler, the resulting executable is named **PortfolioExplorerLoaders.exe**. (Should you be building on another platform, or using a different IDE/compiler, you may name your resulting executable differently, if desired.) This executable requires ten (10) comma delimited files (.csv) as input, to produce the correct output. Each input file is run separately using the executable, with parameters to be discussed later. The beginning of this document will explain those ten .csv files, describing the fields required, and what should be contained within those fields.

Note that all of the following files **must** have the field names as the first line within the files.

# The Layout of the Loader Input Files

As previously stated, there are ten (10) comma delimited text files required as input to create the correct number of data files for PortfolioExplorer. They are as follows:

* **CityStatePostcode** This file contains one record for each combination of a city, state or province, and a postal code, along with other information;
* **PostalCentriods** This file contains one record for each postal code, along with the latitude and longitude of its center position (centroid);
* **Coordinates** This file contains one record for each stored latitude and longitude as referenced by street segments, along with the sequence number of the record;
* **FinanceCityIndex** This file contains one record for each Finance Area, and the ID of its associated CityStatePoscode record;
* **FinanceStreetSoundex** This file contains one record for each street name, where that street name's Soundex (along with other information) is included;
* **PostcodeGroups** This file contains information about possible grouping of postal codes;
* **StateCitySoundex** This file contains the Soundex for a give city, along with its associated state or province, and the ID of its associated CityStatePostcode record;
* **StreetIntersectionSoundex** This file contains one record for each intersecting street segment pair, and necessary information about those street segments;
* **StreetName** This file contains one record for each street **name**, along with information to further help identify the exact street to which it refers;
* **StreetSegment** This file contains one record for each street **segment**, including its address range, the ID of its associated coordinates, and the ID of its associated street name;

## Field Descriptions for each Input File

We will now describe what the fields are for in each one of the above listed loader input files.

### CityStatePostcode

Before assigning the each record its ID (**CITY\_STATE\_POSTCODE\_ID**), the file should be sorted, as follow:  
**POSTCODE** —in ascending order;  
**STATE** —in ascending order;  
**CITY\_NAME** —in ascending order;  
Here is the list of fields required for this file:

1. OLD\_ID
2. COUNTRY
3. POSTCODE
4. STATE
5. CITY\_NAME
6. FINANCE
7. STREET\_NAME\_ID\_FIRST
8. STREET\_NAME\_ID\_LAST
9. CITY\_STATE\_POSTCODE\_ID

#### Field Descriptions:

**OLD\_ID** This field is merely a place holder that is used to store a previous value for a CityStatePostcode ID. Its contents are irrelevant to the loader executable. (See the section on Open Source Alteryx Modules Notes)

**COUNTRY** A two character country code; e.g. "US" for the United States"; "CA" for Canada; Each record within a given data set will have the same value;

**POSTCODE** The postal code for this CityStatePostcode record;

**STATE** The number code for the given state or province; e.g. For US data the USPS "FIPS" codes are used—Colorado (CO) is "08";

**CITY\_NAME** The city name for this CityStatePostcode record, in all caps;

**FINANCE** The Finance Area in which this CityStatePostcode record resides; (See the section describing the **FinanceCityIndex** file for a description of Finance Areas.)

**STREET\_NAME\_ID\_FIRST** The lowest ID of a StreetName record that is physically within this record's city, state or province, and postcode;

**STREET\_NAME\_ID\_LAST** The highest ID of a StreetName record that is physically within this record's city, state or province, and postcode;

**CITY\_STATE\_POSTCODE\_ID** The zero based ID to identify this record within other files' records;

### ****PostalCentriods****

The file should be sorted by the **POSTCODE**, in ascending order.  
Here is the list of fields required for this file:

1. LATITUDE
2. LONGITUDE
3. POSTCODE

#### Field Descriptions:

**LATITUDE** The latitude of the centroid of the postcode, expressed as a decimal value, with a six digit precision;

**LONGITUDE** The longitude of the centroid of the postcode, expressed as a decimal value, with a six digit precision;

**POSTCODE** The postcode in question;

### ****Coordinates****

This file is used by the StreetSegments file, to obtain actual coordinates for a physical street segment. Basically, a street segment is a line that contains a number of points. The minimum number of points that will make up a segment is two, if the segment is relatively short, and is in a straight line. However, a segment may be made up of many point, should it contain curves, and be relatively long in between intersecting street segments.

The sequence member of a record from this file is the offset from the first point of the segment. The first point of the segment is generally associated with the low address number of the address range for a given segment. So, the sequence will start at zero, and go to one less the total number of points that are from an original street segment. (In this case original refers to a street segment before any copies of its street segment record may be made which represent alternative names for the given street.) The coordinate record following the last record for an original street segment will be the first coordinate record for the next original street segment (with a sequence of zero). Therefore, once the coordinate points from the various street segments are separated from their original segment record (depending on how your beginning data is arranged), they need to be kept together, so that this sequencing can make sense.

Before assigning the **COORDINATE\_ID**, the records should be sorted first by the **StreetSegmentID** field, with the **SEQUENCE** field as the second level of sorting. They should be sorted in ascending order. This ensures that all records from the original street segment are kept together, and are in the proper sequence order.

(Note that since one coordinate record can be referred to by multiple street segment records—because of copies of original segment records made for either alternative street names, or for separating left/right street sides from the original data—not all street segment record IDs will be stored in this file.)

Here is the list of fields required for this file:

1. StreetSegmentID
2. COORDINATE\_ID
3. SEQUENCE
4. LONGITUDE
5. LATITUDE

#### Field Descriptions:

**StreetSegmentID** An ID of a street segment which refers to this coordinate record; Note that this field is not actually used by the loader, and also multiple segments may refer to the same coordinate record (See above).

**COORDINATE\_ID** An ID for this coordinate record;

**SEQUENCE** An integer, representing the sequence from the original, specific street segment;

**LONGITUDE** The longitude of the given coordinate, expressed as a decimal value, with a six digit precision;

**LATITUDE** The latitude of the give coordinate, expressed as a decimal value, with a six digit precision;

### ****FinanceCityIndex****

First, a word about what a Finance Area is. This is a U.S. Postal Service (USPS) construction, which allows them to further group postal codes (the U.S. ZIP codes). While all USPS ZIP codes are in Finance Areas, their main use is in urban areas. They tend to group ZIP codes into geographical areas that break down urban areas into groups which have a similar type of zoning, even if that zoning moves across municipality borders. So, for example, in a major metropolitan area, the ZIP codes on one side of the major city in that area, may be grouped with the ZIP codes in some adjoining, smaller municipalities, which have a similar urban layout (being all residential, or business, etc.)

Since this is strictly a USPS construct, original data from other countries will likely not have Finance Areas. However, some way of grouping postal codes in this way helps PortfolioExplorer to find addresses, should the entered search postcode be incorrect, but in the same vicinity as the true postcode. Therefore, you should attempt to mimic Finance Areas in some way. This simplest is to assign IDs to all the given municipalities (thus grouping your postcodes by the cities they are in). However if your data has some other type of grouping (say, through census data, or sub-province level sectioning) which may act in a similar fashion, then you may wish to use that type of data as your Finance Areas. the data used should be six digits, with the left side padded with zeros, if necessary.

The records should be sorted first by the **FINANCE** field, with the **CITY\_STATE\_POSTCODE\_ID** field as the second level of sorting.

Here is the list of fields required for this file:

1. FINANCE
2. CITY\_STATE\_POSTCODE\_ID

#### Field Descriptions:

**FINANCE** The ID of the Finance Area for this record; For USPS data this is a six digit integer. If you mimic this for your input data, it should just be an integer value, that shouldn't go over six digits.

**CITY\_STATE\_POSTCODE\_ID** The CityStatePostodeID record associated with the given Finance Area;

### ****FinanceStreetSoundex****

A Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English. If you are using Alteryx to parse your input data, the Formula tool has a Soundex function, to create a Soundex from a string. If you are not using Alteryx, then, should you not know what a Soundex is, there is an article on the Wikipeda website describing it. Basically, PortfolioExplorer uses the Soundex to help it fuzzy match against input street names.

Before assigning the record ID (**STREET\_INDEX\_ID**), the records in the file should be sorted first by the **FINANCE\_NUMBER**  field, with the **STREET\_SOUNDEX** field as the second level of sorting.

Here is the list of fields required for this file:

1. FINANCE\_NUMBER
2. STREET\_SOUNDEX
3. STREET\_NAME\_ID
4. STREET\_INDEX\_ID

#### Field Descriptions:

**FINANCE\_NUMBER** The ID of the Finance Area for this record; For USPS data this is a six digit integer. If you mimic this for your input data, it should just be an integer value, that shouldn't go over six digits.

**STREET\_SOUNDEX** The Soundex for the street name, as indicated by the **STREET\_NAME\_ID** field;

**STREET\_NAME\_ID** The ID of the corresponding record in the **StreetName** file;

**STREET\_INDEX\_ID** The zero based ID of the given record;

### ****PostcodeGroups****

This is another file which contains information more specific to USPS data, and which may not have a direct correlation to data from other sources. Basically, the USPS assigns main ZIP codes to certain groups of ZIP codes, which almost exclusively in urban areas. Usually, there will only be two to five ZIP codes associated with one main ZIP code. ZIP codes which are not in grouped in this way, are simply in a group by themselves.

If your source data does not contain postcode groups, then you can either create your own, or simply put each postcode in a group by itself. (The two fields in each record would have the same postcode.)

The records should be sorted first by the **POSTCODE\_GROUP** field, with the **POSTCODE** field as the second level of sorting.

Here is the list of fields required for this file:

1. POSTCODE
2. POSTCODE\_GROUP

#### Field Descriptions:

**POSTCODE**  The postcode in question;

**POSTCODE\_GROUP** The main postcode, to which **POSTCODE** is grouped under;

### ****StateCitySoundex****

The records should be sorted first by the **STATE** field, with the **CITY\_SOUNDEX** field as the second level of sorting.

Here is the list of fields required for this file:

1. CITY\_STATE\_POSTCODE\_ID
2. STATE
3. CITY\_SOUNDEX

#### Field Descriptions:

**CITY\_STATE\_POSTCODE\_ID** The ID of a CityStatePostcode record from the **CityStatePostcode** file;

**STATE** The number code for the given state or province;

**CITY\_SOUNDEX** The Soundex for the city name from the corresponding CityStatePostcode record as indicated by the **CITY\_STATE\_POSTCODE\_ID** field;

### ****StreetIntersectionSoundex****

The following algorithm is suggested for determining intersections from original input data:

1. First, the latitude and longitude of each point that make up a street segment, should be compared with every other point in the original data. Any that are matches, should be set aside. (If using Alteryx, this can be done with a Join tool, feeding all records into both inputs, and matching on **both** Latitude and Longitude.)
2. Unless your algorithm has already taken this next step into account, at this time, from the set just created, disregard any matches where the segments are the same. (A street does not intersect with itself.) You may have to create some type of key, based on the street name, pre-directional, post-directional, and type to do this check.
3. Lastly, make sure only the pair that is in one direction in the record is used. For example, if street "A" intersects with street "B", depending upon your matching algorithm, you may have two records in the matched set, one as "A" then "B"; and a second as "B" then "A". They obviously both correspond to the same intersection, so only one is necessary.

Before assigning the each record its ID (**StreetIntersectionSoundexID**), the file should be sorted, as follow:  
**STATE** —in ascending order;  
**SOUNDEX1** —in ascending order;  
**SOUNDEX2** —in ascending order;  
Here is the list of fields required for this file:

1. StreetIntersectionSoundexID
2. STATE
3. STREET\_NAME\_ID1
4. SOUNDEX1
5. STREET\_SEGMENT\_OFFSET1
6. STREET\_NAME\_ID2
7. SOUNDEX2
8. STREET\_SEGMENT\_OFFSET2

#### Field Descriptions:

**StreetIntersectionSoundexID** A zero based ID for the given record;

**STATE** The number code for the given state or province;

**STREET\_NAME\_ID1** The ID of a StreetName record from the **StreetName** file;

**SOUNDEX1** The Soundex for the street name, as indicated by the **STREET\_NAME\_ID1** field;

**STREET\_SEGMENT\_OFFSET1** The sequence number of the coordinate record from the **Coordinates** file applying to the street segment in question;

**STREET\_NAME\_ID2** The ID of a StreetName record from the **StreetName** file;

**SOUNDEX2** The Soundex for the street name, as indicated by the **STREET\_NAME\_ID2** field;

**STREET\_SEGMENT\_OFFSET2** The sequence number of the coordinate record from the **Coordinates** file applying to the street segment in question;

### ****StreetName****

A StreetName record should be created for each unique combination of postcode, city, **PREDIR**, **NAME**, **SUFFIX**, and **POSTDIR**. Before this is done, all possible input records that will be used to create these record, should be sorted as follow:   
postcode —in ascending order;  
province/state —in ascending order;  
city —in ascending order;  
**NAME** —in ascending order;  
**SUFFIX** —in ascending order;  
**PREDIR** —in ascending order;  
**POSTDIR** —in ascending order;  
LOW\_ADDR —in ascending order;  
(Note that postcode, province, city, and LOW\_ADDR are not part of these records. This sorting will generally take place before either the **CityStatePostcode** and **StreetSegment** files are created, while the input data is still more or less in one set of records. postcode, province, and city will be substituted by the **CITY\_STATE\_POSTCODE\_ID** field, and LOW\_ADDR will be placed in the **StreetSegment** file.) After all the records are created, but just prior to output, the file should then be finally sorted by the **STREET\_NAME\_ID** field.

Here is the list of fields required for this file:

1. CITY\_STATE\_POSTCODE\_ID
2. STREET\_NAME\_ID
3. STREET\_SEGMENT\_ID\_FIRST
4. STREET\_SEGMENT\_COUNT
5. NAME
6. SUFFIX
7. PREFIX
8. PREDIR
9. POSTDIR

#### Field Descriptions:

**CITY\_STATE\_POSTCODE\_ID** The ID of a CityStatePostode record associated with this record;

**STREET\_NAME\_ID** The zero based ID of this record;

**STREET\_SEGMENT\_ID\_FIRST** The ID of the lowest StreetSegment record which describes address ranges for this street;

**STREET\_SEGMENT\_COUNT** The number of StreetSegment records which make up all address ranges for this street;

**NAME** The name of the street, in all caps;

**SUFFIX** The type of this street; Usually, this will be an all caps character based code, such as "ST" for street; "AVE" for avenue; "PKWY" for parkway; etc. These, of course, should be in the native languages for the country who's data is being used. Also, these should be standardized, and input variations when geocoding will be standardized to the codes used in this file, using the address\_parser\_suffix\_aliases.csv parse table which will be described later.

**PREFIX** This field is no longer referred to by PortfolioExplorer, and should therefore just be empty;

**PREDIR** The pre-directional for the given street; This is usually a one or two character all caps code, such as "N" for North, "SW" for Southwest, etc. Again, these should be in the native language and standardized. Geocoding input variations can be standardized using the address\_parser\_directional\_aliases.csv parse table to be described later.

**POSTDIR** A possible post-directional for the given street; (It should be treated similarly to **PREDIR**.)

### ****StreetSegment****

It is probably best to create the records for this file in parallel to creating the records for the **StreetName** file. First, the input records used to create this file should be the same records and sorted in the same order as with the **StreetName** file (see above), and second, the StreetName ID created for those records can then be associated with the records of this file at that time. Then, after the creation of the records of this file is complete, it should again be sorted by the **STREET\_SEGMENT\_ID** before output.

Here is the list of fields required for this file:

1. StreetNameID
2. STREET\_SEGMENT\_ID
3. ADDR\_LOW
4. ADDR\_HIGH
5. LEFT\_RIGHT
6. POSTCODE\_EXT
7. COUNTY
8. CENSUS\_TRACT
9. CENSUS\_BLOCK
10. COORDINATE\_ID
11. COORDINATE\_COUNT
12. INPUT\_STREET\_SEGMENT\_ID

#### Field Descriptions:

**StreetNameID** The ID of a StreetName record associated with this record;

**STREET\_ SEGMENT \_ID** The zero based ID of this record;

**ADDR\_LOW** The low address number for this segment;

**ADDR\_HIGH**  The high address number for this segment;

**LEFT\_RIGHT** A Boolean, which describes whether this segment is the left, or right side of the street addresses in question; It should be true ("T") for the right side of the street, or false ("F") for the left side of the street.

**POSTCODE\_EXT** A possible postcode extension for this segment; For example, with U.S. Postal data, this could be the four digit extension, that in association with a CityStatePostcode record, could produce a "ZIP 9" postal code. This field may be left blank.

**COUNTY** The county in which this segment lies; For U.S. data, a three digit county code should be used. If there is no corresponding geographical area for your data, then this field may be left blank;

**CENSUS\_TRACT** This field and the following field are intended for U.S. data, but may be used to hold corresponding census data, if available, for your data. This would be used for smaller census areas, if applicable. It may be left blank if no census data is available or used.

**CENSUS\_BLOCK** This is for a larger census area data; Again, if no census data is used, this may be left blank.

**COORDINATE\_ID** The ID of the **first** Coordinate record associated with this segment. See the discussion regarding the **Coordinates** file for more information about what constitutes the first Coordinate record.

**COORDINATE\_COUNT** The number of Coordinate records that make up the full spatial description of this street segment;

**INPUT\_STREET\_SEGMENT\_ID** This is generally used to hold an ID from the original input data. Note that after the possible copying of original input records (for such things as left versus right street segments, aliased street names, etc.), this may have a number of duplicates. It is not referred to by either the loader program or PortfolioExplorer, and therefore, may be left blank.

# **Open Source Alteryx Modules Notes**

These are only a few notes that pertain specifically the Alteryx modules that are used to parse Tele Atlas "TIGER" ASCII data. They are by no means exhaustive, and mostly identify a couple of places where certain fields are set.

* **OLD\_ID** In module 7, is a renamed value of the original **CityStatePostcodeID** after the adding in of new records containing city aliases; The new **CityStatePostcodeID** is created just prior to output of CityStatePostcode2 database;
* StreetName2 data gets new **CityStatePostcodeID** from module 7, based on **OLD\_ID** after the adding in of city alias records to create the CityStatePostcode2 data;
* **INPUT\_STREET\_SEGMENT\_ID** is **RECNUM** from original TIGER data; Its use in the StreetSegment2 output file is really unnecessary, as the Loader does not reference it;
* **FINANCE** and **FINANCE\_NUMBER** are both references to the same data, and come from the USPS provided CTYSTATE.CSV file, **D** Detail records;
* **POSTCODE\_GROUP** comes from USPS CTYSTATE.CSV file, **S** records;

# Using the Loader Program

Once all of the above files have been created, and you are fairly confident in their accuracy and completeness, you will then need to create the required database files needed by PortfolioExplorer to actually geocode. These database files consist of a set of .dat files, and .txt files. To create these files, you must run the Loader program, once for each of the above files you just created. The program normally builds as a command line program, so some type of batch file can be used to run it the required number of times, with the correct parameters to generate the output.

The program takes the following parameters:

1. The particular Loader name; It must be one of the following (case insensitive):
   1. CitySoundex
   2. CityStatePostcode
   3. CityStatePostcodeFaIndex
   4. Coordinate
   5. PostcodeAlias
   6. PostcodeCentroid
   7. StreetIntersectionSoundex
   8. StreetName
   9. StreetNameSoundex
   10. StreetSegment
2. The file path and name; A relative path is acceptable.
3. The path to the output folder; Again, a relative path is acceptable.

Lastly, you can redirect the output of the program to a Log file, to which you can refer later to verify the success of the operation.

So, a batch file line (on the Windows operating system) may look like this:  
Portfolioexplorerloaders.exe StreetName ..\INTERMEDIARY\StreetName.csv..\OUTPUT >>..\LOG\BuildLog.TXT

## Database Files Created by the Loader

When the program is run with all 10 input files, 55 output files should be built. Here is a list of those files:

CitySoundex.dat StreetNameSoundex.dat  
CityStatePostcode.dat StreetNameStreetSegmentCountHuff.txt  
CityStatePostcodeFaIndex.dat StreetNameStreetSegmentIDFirstHuff.txt  
CoordinateLatitudeHuff1.txt StreetNameSuffixHuff.txt  
CoordinateLatitudeHuff2.txt StreetSegment.dat  
CoordinateLongitudeHuff1.txt StreetSegmentAddrHighHuff1.txt  
CoordinateLongitudeHuff2.txt StreetSegmentAddrHighHuff2.txt  
CoordinatePoint.dat StreetSegmentAddrLowKeyHuff1.txt  
CoordinatePointPositionIndex.dat StreetSegmentAddrLowKeyHuff2.txt  
PostcodeAliasByGroup.dat StreetSegmentAddrLowNonkeyHuff1.txt  
PostcodeAliasByPostcode.dat StreetSegmentAddrLowNonkeyHuff2.txt  
PostcodeCentroid.dat StreetSegmentCensusBlockKeyHuff1.txt  
StreetIntersectionSoundex.dat StreetSegmentCensusBlockKeyHuff2.txt  
StreetIntersectionSoundex1Huff.txt StreetSegmentCensusBlockNonkeyHuff1.txt  
StreetIntersectionSoundex2Huff.txt StreetSegmentCensusBlockNonkeyHuff2.txt  
StreetIntersectionSoundexPostitionIndex.dat StreetSegmentCensusTractKeyHuff1.txt  
StreetIntersectionStateHuff.txt StreetSegmentCensusTractKeyHuff2.txt  
StreetIntersectionStreetNameID1Huff.txt StreetSegmentCensusTractNonkeyHuff1.txt  
StreetIntersectionStreetNameID2Huff.txt StreetSegmentCensusTractNonkeyHuff2.txt  
StreetIntersectionStreetSegmentOffset1Huff.txt StreetSegmentCoordinateCountHuff.txt  
StreetIntersectionStreetSegmentOffset2Huff.txt StreetSegmentCoordinateIDHuff1.txt  
StreetName.dat StreetSegmentCoordinateIDHuff2.txt  
StreetNameCityStatePostcodeIDHuff.txt StreetSegmentCountyKeyHuff.txt  
StreetNameNameHuff.txt StreetSegmentCountyNonkeyHuff.txt  
StreetNamePositionIndex.dat StreetSegmentPositionIndex.dat  
StreetNamePostdirHuff.txt StreetSegmentPostcodeExtKeyHuff.txt  
StreetNamePredirHuff.txt StreetSegmentPostcodeExtNonKeyHuff.txt  
StreetNamePrefixHuff.txt

Some of the above .dat files may be very large, dependent upon the amount of original input data. Also, if you do not have census information, the text files corresponding to those may be of size zero. This is okay, but you still need to have the files for PortfolioExplorer to load.

Lastly, one additional file needs to be included in the same folder with these database files. That file should be named **Version.txt**. It should be a standard ASCII text file, and only contains the number "12" (ASCII codes 0x31 and 0x32) as its content.

# Parse Tables

Lastly, PortfolioExplorer will need to load several tables and other auxiliary files to help it parse parts of an address. With the exception of one of these files, most of these will need to be created manually, and will not need to be updated with each possible new build from more current original data. You may need to occasionally add to, or subtract from these files, as certain types of input from your users is encountered, but other than that, after they are first built, they will not need much maintenance.

If you are creating a data set based on U.S. data, then the versions of these file that are in the open source repository (in the path "/install/tables") may be used. If you are creating your own (for a different country), then you should refer to these files to see how they are organized, and what their content is.

Note that all strings in the following .csv files should be **all** capitals, and the field names are **not** within the files. While many of the files are not required to be in any sorted order, it is often better for them to be sorted. Please look at those in the repository for hints about possible sorting.

Here are a list of the tables files needed by PortfolioExplorer:

* **address\_parser\_address\_token\_table.csv** Denotes typical parts of an address; Used for First Line parsing;
* **address\_parser\_city\_component\_aliases.csv** Denotes typical parts of a city name; Used for Last Line parsing;
* **address\_parser\_directional\_aliases.csv** Standardizes directional strings to a one or two character format; Used for First Line parsing;
* **address\_parser\_directionals.csv** Lists typical directional strings; Used for First Line parsing;
* **address\_parser\_reverse\_full\_directionals.csv** Lists typical directional strings, but backwards; Used for First Line parsing;
* **address\_parser\_reverse\_suffixes.csv** Lists typical street suffixes (street types), but backwards; Used for First Line parsing;
* **address\_parser\_state\_aliases.csv** Standardizes variations of state or province names or abbreviations to a two character format; Used for Last Line parsing;
* **address\_parser\_streetname\_aliases.csv** Denotes typical parts of a street name, including numbered streets; Used for First Line parsing;
* **address\_parser\_streetname\_multiword\_aliases.csv** Standardizes typical multiword parts of street names; Used for First Line parsing;
* **address\_parser\_streetname\_multiword\_search\_aliases.csv** Listing of individual words of typical multiword parts of a street name; Used for First Line parsing;
* **address\_parser\_streetname\_prefix\_dash\_aliases.csv** Lists typical parts of a street name, which may be followed by a dash or hyphen; Used for First Line parsing;
* **address\_parser\_suffix\_aliases.csv** Standardizes variations of street suffixes (street types) to a format generally recognized by a postal service; Used for First Line parsing;
* **address\_parser\_unit\_designator\_aliases.csv** Standardizes variations of names of housing units (such as "apartment" or "office") to a format generally recognized by a postal service; Used for First Line parsing;
* **state\_abbr\_to\_fips.csv** Converts state/province abbreviations (which may be pre-pended with a two character country abbreviation) to a postal service recognized numeric code; Used throughout geocoding process to perform conversion;
* **state\_fips\_to\_abbr.csv** Converts state/province postal service recognized numeric code (which may be pre-pended with a two character country abbreviation) to an abbreviation; Used throughout geocoding process to perform conversion;
* **states.csv** List of naming variations of states/provinces; Used for finding state abbreviations as any part of an address;
* **city\_replacement.csv** Maps possible alternate city names to recognized or standardized city names; Used throughout geocoding process to find addresses by a possible alias;  
  The file should have two fields as follows:  
  The first is a look up key, in the format of  
  "two character state abbreviation" "|" "City Alias" "|" "postal code"  
  The second field being the **standard** city name;   
  Generally, this file should be created automatically, when creating the input files for the Loader program, as it contains information which may change as postal office city names and aliases may change.
* **address\_parser\_patterns.dlp** This file is an XML file which contains some of the patterns used by the parser for matching input data; Generally, the one available can be used for most countries, with only some editing of language specific matching string constants nested within the XML.