# C# Assignment #2

One of the most common tasks for developers on this project is writing and updating Converters. A Converter on this project is a block of C# code that takes a set of data files from a supplier, extracts the address information, and stores it in the standard format used in our build processes.

In this exercise, you will create a simple Converter in C#.

## Inputs:

There are four input files for this exercise that you will use as input for a C# program that will assemble them into a file of addresses. The four input files are:

1. Cities.txt – this a file of city and state information
2. Streets.txt – this is file of street and postal code information
3. Buildings.txt – this is a file of building numbers that relate to various streets
4. Apartments.txt – this is a file of apartment numbers that relate to various buildings

All of the files are in pipe-delimited text using a UTF-8 encoding.

Here are the input formats:

Cities:

1. CityId – unique identifier for the row, referenced from the Streets.txt file
2. StateName – the name of the state
3. CityName – the name of the city

Streets:

1. StreetId – unique identifier for the row
2. CityId – the identifier for the row in the Cities file in which this street resides
3. StreetName – the name of the street
4. PostalCode – the postal code for the street, always 5 digits

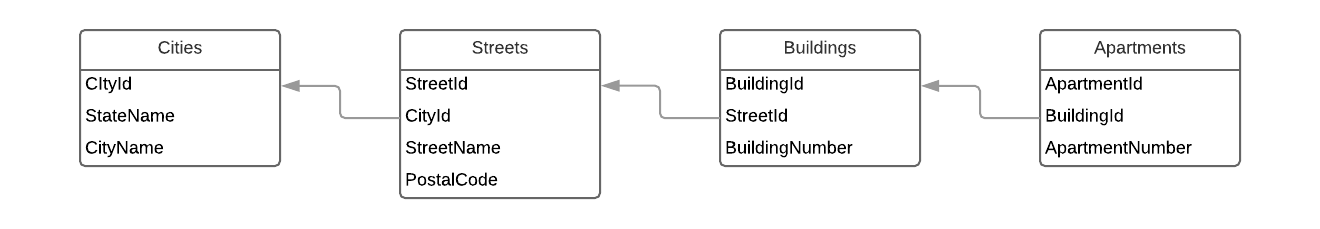
Buildings:

1. BuildingId – unique identifier for the row, referenced from the Apartments file
2. StreetId – the identifier for the row in the Streets file to which this building number belongs
3. BuildNumber – the number for the building, may include numbers, text or both

Apartments:

1. ApartmentId – unique identifier for the row
2. BuildingId – the identifier for the row in the Buildings file to which this apartment number belongs
3. ApartmentNumber – the number for the apartment, may include numbers, text or both

Diagram of file relationships between the files:



## Output:

The output file you produce will be in pipe-delimited text using a UTF-8 encoding. It will have the following columns:

1. StateName – the value from the Cities file that goes with a given StreetId
2. CityName – the value from the Cities file that goes with a given StreetId
3. PostalCode – the value from the Streets file that goes with a given StreetId
4. StreetName – the value from the Streets file that goes with a given StreetId
5. DeliveryPoints – a compound field that has a semicolon-delimited list of all of the BuildingNumber values that relate to a given StreetId. For each BuildingNumber in the list, if there are entries in the Apartments file that reference the BuildingId, they should be in a comma-delimited list surrounded by parentheses.

Here is an example output line extracted from the actual sample files:

WA|Seattle|98155|Main St|11;22;45(A,B,C);68;104A;120(1,2,3)

In the example above, the “A,B,C” values come from rows in the Apartments file that have the same BuildingId as “45” and “1,2,3” values come from rows in the Apartments file that have the same BuildingId as “120”.

## Assignment Constraints

One of the natural ways to assemble this sort of table data would be to import it into a database and use SQL queries to assemble the output rows. However, for the purposes of this exercise, you are asked to produce a C# program that will assemble the output rows. Please limit yourself to classes that come with the standard with Visual Studio. You may assume that this will run on a Windows 10 system that has 16G of RAM.