University of Redlands

Using Global OpenStreetMap Data to Solve Local Problems

A Major Individual Project submitted in partial satisfaction of the requirements
for the degree of Master of Science in Geographic Information Systems

by
David Crawford

Douglas Flewelling, Ph.D., Committee Chair
Mark Kumler, Ph.D.

February 2021

Using Global OpenStreetMap Data to Solve Local Problems

Copyright © 2021

by
David Crawford

The report of David Crawford is approved.

Mark Kumler, Ph.D.

Douglas Flewelling, Ph.D., Committee Chair

February 2021

Acknowledgements

I would like to extend thanks to my client, Christopher Price, for excellent guidance and an interesting topic on which to author a report, and for patience in waiting for the results. Thanks to my friends and family that have encouraged and helped me throughout the process of completing this project. Thanks to Esri, my employer, for granting me the opportunity to pursue an MS-GIS. Thanks to my parents for instilling in me a work ethic and drive to always pursue higher education. And a final, but most important of all, thanks to my wonderful wife, Julianna, without whom completing a masters would have remained not more than a 'pipe dream'. Julianna, for the encouragement, support and love that you have given me during this entire process, there are no words to describe how important you have been, and will always be, to me.

Abstract

Using Global OpenStreetMap Data to Solve Local Problems

by
David Crawford

Geographic Information Systems (GIS) consulting companies are looking for ways to cut costs. Data acquisition is often the largest cost of any GIS project. OpenStreetMap (OSM) provides Volunteered Geographic Information (VGI) which is freely available for all people to use. OSM data is global in scale, and very difficult to work with. OSM data needs to be organized and loaded into a Geodatabase. This enables Geoprocessing and analysis, using Esri's ArcGIS Platform. This is essential to every consultant, as almost all their products are a result of analysis. The purpose of this project was to answer the question: Can global OpenStreetMap data be used to answer local questions?

This project suggests a consistent and repeatable workflow to extract, translate, and load (ETL) OSM data into an Esri Geodatabase. The project determines reliability of using VGI, with a thorough look at current research on the quality and accuracy of VGI and, specifically, OSM. It is essential that we understand the potential issues with the quality and accuracy of VGI so that we may associate the appropriate level of trust to the results of any analysis. Current research and analysis conclude that OSM and VGI provide data of a consistent quality and accuracy which is appropriate to use in analysis at the local level. This project enables analysis of OSM's global dataset at scales to address local problems.

Table of Contents

Chapter 1 – Introduction	1
1.1 Client.....	1
1.2 Problem Statement	1
1.3 Proposed Solution	1
1.3.1 Goals and Objectives	1
1.3.2 Scope.....	2
1.3.3 Methods.....	2
1.4 Audience	2
1.5 Overview of the Rest of this Report	2
Chapter 2 – Background and Literature Review	3
2.1 Does OpenStreetMap Provide Accurate, Quality Data?.....	3
2.1.1 OpenStreetMap Data Quality and Accuracy.....	3
2.1.2 What was Discovered?.....	4
2.2 Can Volunteered Geographic Information be Trusted?.....	4
2.2.1 Volunteered Geographic Information	5
2.2.2 What was Discovered?.....	5
2.3 Summary	5
Chapter 3 – Systems Analysis and Design.....	7
3.1 Problem Statement	7
3.2 Requirements Analysis	7
3.2.1 Details of Functional Requirements.....	7
3.2.2 Details of Nonfunctional Requirements.....	9
3.3 System Design	11
3.3.1 System Interface Requirements	12
3.3.2 Hardware and Software Requirements	12
3.4 Project Plan	12
3.4.1 Planning Phase	13
3.4.2 Design Phase.....	15
3.4.3 Implementation/Testing Phase.....	15
3.4.4 Deployment Phase	15
3.5 Summary	15
Chapter 4 – Database Design.....	17
4.1 Conceptual Data Model	17
4.2 Logical Data Model	17
4.2.1 OpenStreetMap Data Model	18
4.2.2 Local Government Information Model	18
4.2.3 OpenStreetMap Attribution	18
4.2.4 Local Government Information Model Attribution	19
4.3 Data Sources	20
4.4 Data Collection Methods	20
4.5 Data Scrubbing and Loading	20
4.6 Summary	21

Chapter 5 – Implementation.....	23
5.1 Introduction.....	23
5.2 Extracting OpenStreetMap data.....	23
5.3 Building an OpenStreetMap Taxonomy	25
5.4 Translating OpenStreetMap Data to Local Government Information Model.....	26
5.4.1 Loading Data into the Correct Target Feature Class	28
5.5 Creating a .Net Application to Extract a Subset of Data	30
5.6 Summary	32
Chapter 6 – Results and Analysis.....	35
6.1 Test Cases	35
6.2 Performance Numbers	41
Chapter 7 – Conclusions and Future Work.....	43
7.1 Project Summary.....	43
7.2 Future Work	43
Works Cited	45
Appendix A. Python Script Mapping Extracted OSM data to LGIM Feature Classes	47
Appendix B. Python Script to Download and Extract OSM File.....	81

Table of Figures

Figure 3-1: Complete system interaction graphic	11
Figure 4-1: Complete field list on Park class that the LGIM provides with the OSM fields for this project added.....	20
Figure 5-1: Raw .osm data converted to simple point, line, and polygon features in the area of Naperville, Illinois.	24
Figure 5-2: Viewing the dual data representation of a street in Naperville, Illinois..	25
Figure 5-3: A sample of mapping the OSM highway attribute to the appropriate RoadCenterline feature class using an insert cursor in Python.....	28
Figure 5-4: View of Naperville, Illinois with the organized full OSM dataset in the LGIM.	29
Figure 5-5: View of the complete RoadCenterline feature class following data extraction.....	30
Figure 5-6: The User experience that allows an analyst to select the type of data they would like to use for analysis.....	31
Figure 5-7: The User is shown a graphic such as this for the area that will be extracted for their analysis.	32
Figure 5-8: Roads in Naperville, Illinois, extracted from OSM source data, with a route between three locations displayed.	33
Figure 6-1 : Map showing the ability to extract specific data around New Orleans to answer a question posed.....	36
Figure 6-2: Map showing the ability to extract data and leverage a Network Dataset built on the extracted data.	38
Figure 6-3: Map showing the ability to extract polygons to answer questions.	40

List of Tables

Table 1.	Functional Requirements	7
Table 2.	Nonfunctional Requirements	9
Table 3.	Preliminary Project Plan	12
Table 4.	Adjusted Project Plan.....	14
Table 5.	Sample of the field names in OSM (left most column) and the values that appear in those columns indicating different feature types.	25
Table 6.	A sample of the data mapping from OSM ‘highway’ to RoadCenterline feature class and appropriate road class attribute in the LGIM.	27
Table 7.	Performance numbers captured for extraction a collection of themes at 3 set scales (city [Redlands, CA], state [Louisiana], Country [USA]).....	41

List of Acronyms and Definitions

DM	Data Model
ESRI	Environmental Systems Research Institute Inc.
ETL	Extract, Translate and Load
GIS	Geographic Information Systems
LGIM	Local Government Information Model
OSD	Open Sourced Data
OSM	OpenStreetMap
SDK	Software Developer Kit
VGI	Volunteered Geographic Information

Chapter 1 – Introduction

The Geographic Information Systems (GIS) consulting industry is highly competitive. Consultants are always trying to outperform each other with lower costs. One of the largest costs of consulting is data acquisition. (Price, personal communications, 2016). GIS data are expensive, and at times difficult to find. When working on consulting projects acquiring cheap, useful, detailed data creates a competitive edge. GIS consultants are investigating the use of Volunteered Geographic Information (VGI), also known as Open Sourced Data (OSD), to minimize project costs. While VGI is not appropriate for all projects, for many, it provides a level of detail appropriate to answer key questions and guide decision making.

1.1 Client

This report was prepared for Viking Geographic Inc., a GIS consulting company. Viking Geographic is owned by Christopher Price. Mr. Price has engaged in several consulting projects and has an interest in engaging in many more. Data costs were always a problem for Mr. Price in his consulting work. Mr. Price is interested in using OpenStreetMap (OSM) as the data source for future consulting projects.

1.2 Problem Statement

GIS consultants struggle to obtain authoritative, affordable or free data. OSM offers global data coverage, with highly detailed data in heavily populated regions, and less detailed data in sparsely populated regions. The data format is not natively supported by the ArcGIS Platform. GIS consultants use the geoprocessing and analytic capabilities of the ArcGIS Platform to deliver client-focused solutions. Converting OSM data into an appropriate data format is time consuming. Users performing analysis are forced to sift through an abundance of data to identify data relevant to their task.

1.3 Proposed Solution

This project proposes extracting, translating and loading (ETL) the OSM global dataset into the Local Government Information Model (LGIM), which is a data model provided by Esri ArcGIS Solutions. This project would create a series of scripts to periodically update those data from the OSM central repositories. The project provides a data extraction workflow allowing a user to identify an area and topic, or topics, of interest and to access relevant data. The user is then able to perform complex analytics and geoprocessing on a subset of data. The elimination of data acquisition costs while maximizing the consultants' ability to deliver the required analysis at a low cost is a high value result to the problem of high cost data acquisition.

1.3.1 Goals and Objectives

The goal of this project was to provide a methodology for converting OSM data into Esri format data and keeping those data up-to-date. The specific objectives for this project

were to provide extraction tools to enable analysis on a subset of the global dataset. The user needed to be able to define the subset required with an extent and topics of interest.

1.3.2 Scope

OSM data covers the entire globe. For this project, scope was limited to North America. The project was designed to have an ETL which moved OSM data into the LGIM. However, there are almost an infinite amount of data types represented in the OSM dataset. Therefore, this project is limited to keeping the data types which are relevant to the example analysis that the client performs. This project designed a process which could be followed in the future to include additional continents, and could be expanded to map even more of the OSM data types into the LGIM.

1.3.3 Methods

There were three primary deliverables for this project. Several methods were utilized to generate the deliverables. First, OSM data was downloaded from a central repository. This was accomplished using the Google Chrome web browser. OSM provides several trustworthy servers from which data are downloaded. Second, Python scripts were written to update an Esri Geodatabase from the OSM database server. An Esri extension toolbox for working with OSM data was utilized for the initial data extraction. The extension toolbox contains Python commands that can be executed to identify areas where data updates are available from OSM. These data are downloaded using core Python functions and applied using the 'Update OSM' Python script. Lastly, tools to extract data for an area and topic of interest were created. These tools were developed using ArcObjects and the .Net Software Developer Kit (SDK). Windows Forms provide functionality to support user interaction with the tool. A form allows a user to pick the topics of interest. A tool allows creation of a polygon on the map to identify an area of interest.

1.4 Audience

The primary target audience for this project are GIS consultants and executives at GIS consulting companies. The reader is expected to have a detailed understanding of GIS concepts, such as: area of interest, thematic mapping, and geoprocessing. The reader is expected to understand what http and REST is and how data are transferred over the internet.

1.5 Overview of the Rest of this Report

Chapter 1 of this report provided an introduction and overview of the project. Chapter 2 is an examining of existing research on the quality and accuracy of VGI and OSM. Chapter 3 is a discussion of the project requirements and the delivery plan. Chapter 4 is a discussion of how the database was designed. Chapter 5 discusses the implementation process, what scripts and tools were developed, and why. Chapter 6 summarizes results and analysis. Chapter 7 offers project conclusions and ideas for future work.

Chapter 2 – Background and Literature Review

This chapter is a review of literature pertinent to the areas of Volunteered Geographic Information (VGI) and OpenStreetMap (OSM). Mooney and Corcoran (2012) identifies that “OSM is a very large (and growing) spatial dataset and probably the most well known example of VGI.” (p.563). Alternatively, Girres and Touya (2010) point out that “OSM is a collaborative project, similar to Wikipedia, which was started in England in 2004 by Steve Coast.” (p.436). Collaborative projects, where non-experts generate content, present risks to data quality and accuracy. There is a concern that the casual user does not have the expertise to accurately and correctly capture data. To understand the usefulness of this project for real-world consulting purposes, we need to answer two questions:

1. Does OpenStreetMap provide accurate, quality data?
2. Can VGI be trusted?

Answering both preceding questions in the affirmative is to provide grounds that the results of this project are useful.

2.1 Does OpenStreetMap Provide Accurate, Quality Data?

OpenStreetMap (OSM) has become an authoritative source of global data. “There are over 400,000 contributors registered in the OSM project. Volunteers in the OSM community collect geographic information using GPS devices and submit this to the global OSM database” (Mooney and Corcoran, 2012, p.562). Girres and Touya (2010) explain that OSM provides a type and attribute system for describing data, and investigate the quality and accuracy of the data from OpenStreetMap versus the commercially produced dataset BD Topo Large Scale Referential (RGE) from the company IGN. Hochmair, Zielstra and Neis (2014) look specifically at one type of feature in the OSM dataset, bicycle trails and lanes to evaluate data quality and completeness. They use Google Maps, Google Street View, and other commercial software to determine the accuracy of the data.

2.1.1 OpenStreetMap Data Quality and Accuracy

Girres and Touya (2010) used the commercial dataset RGE as ground truth. OSM comparison was completed for geometric and attribute accuracy. The questions being investigated are: are the data where they should be, and have they been identified as the correct type of data. The article identifies several areas in the French dataset where data are incorrectly located by up to 15 meters. However, it also identifies areas where the data is markedly accurate, within 1 meter Girres and Touya (2010). Data incorrectly located by very large distances, up to 15 meters, were in locations that were remote, and difficult to accurately data capture. These potential inaccuracies can severely limit the type of questions that the dataset can reliably be used to answer. For example, 15 meter inaccuracies in a shoreline can be managed, but 15 meter inaccuracies in property lines would cause analysis problems. Hochmair, Zielstra and Neis (2014) describe a more accurate OSM dataset. The data in certain areas are incredibly detailed, even more accurate than commercially available data. However, in rural, sparsely populated areas

the data can be inconsistent. Inconsistencies found in the bicycle trails data were found to be minor in most cases. Most importantly, by comparing these two articles one can see a change over time. The article published in 2010 pointed out that topological integrity is not present in the OSM dataset outside of major cities, and a lot of post-process correcting is required to enable standard GIS processes. Whereas the 2014 article identified that topological integrity is present in more of the data. Streets touch other streets at intersection points, and it was rare to find streets and bike trails that were not topologically coincident, even outside of population centers (Hochmair, Zielstra and Neis, 2014). This suggests OSM data is getting more accurate over time. Major cities and heavily populated regions had very good and detailed data. Rural areas have data accuracy issues, but those issues are being corrected and the dataset is becoming more accurate over time.

2.1.2 What was Discovered?

Girres and Touya (2010) point out that there is an “...advantage of responsiveness and flexibility of OSM, but also the problematic aspect of heterogeneity in OSM data, highly limiting the possible applications”(p.458). In other words, we should not blindly trust OSM data to be accurate. Two data collectors might look at the exact same feature type in two different locations and store them with different attributes. This is a system without standards, so understanding the entire taxonomy of data types and attributes stored in OSM is very important.

Girres and Touya (2010) identify that the problem of heterogeneity persists in OSM today. Hochmair, Zielstra and Neis (2014) provide a simple example to illustrate the problem. OSM data have tags that are used to delineate what street and bike lane configurations exist. There are complex interdependencies between multiple tags, ex. Highways = * AND bicycle = pathway, means a bike lane which is separate and running adjacent to a road, whereas Highways = * AND bicycle = yes, means a bike lane which is painted on the asphalt. These subtleties can be easy to misinterpret and allow users to incorrectly record information into the OSM dataset (p.65). These complexities make consistency in the data difficult. However, both groups of authors identified that even in cases of feature disagreement, it was often in the attributes describing specifics about the features, and not differences in the actual type of the feature. These differences and issues are found to be improving over time. The OSM dataset is moving in the direction of an authoritative data source. OSM data is, at this point, authoritative in major cities. It is dedicating, via crowd data creation events, and targeted dataset updating, to becoming authoritative across the entire globe.

2.2 Can Volunteered Geographic Information be Trusted?

To what level do we, and should we, trust Volunteered Geographic Information (VGI). Barron, Neis and Zipf (2013) and Mooney and Corcoran (2013) were used to evaluate the questions posed. Each of these articles investigates, albeit in a slightly different manner, the trustworthiness of VGI. The VGI discussed in these articles is specifically OSM. This is most relevant to this project because it is only OSM’s dataset that this project is interested in using.

2.2.1 Volunteered Geographic Information

Barron, Neis and Zipf (2013) posits a methodology for determining the quality of VGI from the data itself. The questions this article investigates are: Can this data be quality checked using only itself? How accurate is the data, when compared to other sources? Mooney and Corcoran (2013) investigates multiple editors contributing to a single database, and whether this results in good or bad data. One might suggest that a large collection of disparate editors, especially when there are no enforced editing standards, as is the case with OSM, could result in many different ways to identify and store information about similar features, (Mooney & Corcoran, 2013) investigates this potential.

2.2.2 What was Discovered?

Barron, Neis and Zipf (2013) discusses that there remain swaths of data which are incomplete or inaccurate. Such as buildings which have no address associated. "..., only four buildings are annotated with a house number/name indicating very low attribute completeness. In contrast, 1,146 (10.2%) of all buildings within San Francisco are tagged with a house number/name, whereas Madrid takes up a middle position with 1,024 (4.0%) tagged buildings" (Barron, Neis and Zipf, 2013, p.890). Mooney and Corcoran (2013) concurs that a problem of multiple editors is attribute incompleteness. Geographic completeness, on the other hand, is agreed by Barron, Neis and Zipf (2013) and Mooney and Corcoran (2013) to be consistent in the dataset. Mooney and Corcoran (2013) further identified that geography between multiple editors is consistent, but when changed, is improving over time. Barron, Neis and Zipf (2013) discuss in detail an analysis of different feature types and how they are changing over time. Barron, Neis and Zipf (2013) notes that "..., the majority of the polygons' geometry in all three cities has not been changed (Madrid: 62.7%; San Francisco: 67.4%; Yaoundé: 62.9%)" (p.891). Looking at other data types: streets, landuse, water, and coastlines. An analysis process was put in place to determine how the features in OSM (nodes and ways) had moved over time to determine if: 1. The streets were done being captured 2. The data was being adjusted over time to account for small inaccuracies or 3. The data was drastically moved over time indicating complete mis-entry (Barron, Neis and Zipf, 2013).

2.3 Summary

Each of the articles describes valid concerns of the public, and academic communities, as it pertains to the accuracy of OSM. People want to understand the level of faith they can place in OSM. This is a crucial portion of understanding as it pertains to this specific project because a contractor would be looking for authoritative data to ensure their results can be trusted.

The articles agree, in rural areas the data are less detailed and less accurate and in population centers it is much more detailed and remarkably accurate. The articles also concur that over time the data is becoming more accurate. Girres and Touya (2010) identified that in most areas the data, when compared to the BD Topo dataset, was very accurate – geographically. Attribution got less accurate the further one gets from major cities. Hochmair, Zielstra and Neis (2014) suggest that bike lanes are more accurately

modeled in certain locations in OSM than in Google or other commercial software. The bike lanes, at times, are attributed incorrectly, but the geography is almost always accurate. Barron, Neis and Zipf (2013) found that OSM data are being edited less frequently over time. This is to suggest that the data are being made more accurate and therefore further updates are not necessary. Barron, Neis and Zipf (2013) also notes that attribute updates are the primary update at this point in time to OSM. Finally, Mooney and Corcoran (2013) found that many editors, editing in a disparate and detached setting without any defined editing standards are often entering attribute information inaccurately, however, the geographies and broad type of data are entered correctly.

In conclusion, all articles come to a consensus in their results. OSM data are of a geographic accuracy necessary to facilitate the analysis for this project. There are, however, issues with some of the detailed attribution necessary to facilitate a complete GIS. Third-party companies are producing new editing tools to facilitate OSM contributors following consistent patterns and standards. These new tools enable detailed, accurate attribute collection. OSM data, as of now, are useful for geographic analysis which does not depend on a deep and thorough attribute record.

Based on the research conducted during this chapter OSM and VGI can be trusted to provide authoritative and accurate geographic location data. This meets the requirements for this project, because the analysis completed with this data will only require accurate geographies and not a complete attribute record.

Chapter 3 – Systems Analysis and Design

A successful project is driven by the needs of the client. This project encapsulates a specific system designed to solve Viking Geographic's specific analysis goals. Section 3.1 outlines the problem that Viking Geographic is trying to solve. Section 3.2 will then analyze the functional and nonfunctional requirements for a system which can respond adequately to the problem. Section 3.3 details the systems design and section 3.4 describes the project plan put in place to accomplish a functional system which matches the design.

3.1 Problem Statement

GIS consultants struggle to obtain authoritative, affordable, or free data. OpenStreetMap (OSM) offers data at a global scale. OSM data is not native to the Esri ArcGIS Platform. There are tools and extensions available to work with OSM data in ArcGIS. However, the quantity of data is too great for an analyst to work with effectively. GIS consultants require OSM data, in a format that is native to the Esri ArcGIS Platform, with tools that allow them to work with a desired subset of the overall dataset.

3.2 Requirements Analysis

Requirements can be best broken into three categories: functional, nonfunctional, and data. After meetings with the client, the overall requirements of the project were determined. The following sections will detail each of the requirements for this project and will divide those requirements into their appropriate categories.

3.2.1 Details of Functional Requirements

Table 1 lists the project's functional requirements with a brief description. Further in section 3.2.1 is a detailed description of each functional requirement and what the project will provide to satisfy the requirements.

Table 1. Functional Requirements

Functional Requirement	Description
Download the latest OSM data	The user shall execute a Python script, without interaction, which downloads and extracts the latest OSM data.
Load the project Geodatabase with the latest data	The user shall execute a Python script, without interaction, to load the latest OSM data into a project Geodatabase

Select a topic of interest	The user shall be presented with a list of the available data themes which they can choose from to select data appropriate to their analyses.
Define area of interest with map interaction	The user shall be presented with an interactive drawing tool to indicate an area of interest on a map.
Perform analysis on extracted data	The user shall be provided the subset of data they are interested in, in a data format suitable for geoprocessing.

The requirement of downloading the latest OSM data allows the user to have confidence that they will be able to use up-to-date data for their analyses. It is important that a consulting company is using the latest data. It is inappropriate while consulting on a project to posit solutions to problems if the data used for the analysis is out of date. How can one be certain that their results will be accurate if the data they are working with is three or four months or even years out of date. This requirement is dependent on a Python script. The user cannot be expected to interact with this script – it should run as a batch file or some other autonomous execution pattern. It is expected that the user will remain disconnected from the system – the authoritative project geodatabase – whenever the updates are being processed. The user will be presented during execution with a command prompt which provides messaging about the processing to the user.

Load the project geodatabase with the latest data is a straightforward requirement. The user wants to work with the latest data, and a Python script will be provided which moves that data into the project geodatabase. The most recent data is downloaded and extracted from a .bz2 file to an .osm file and then moved into an intermediate geodatabase. A data loading script then executes to move that data into the authoritative production geodatabase. This requirement is the ETL portion of the project, moving the data from an OSM format into an Esri Geodatabase format. Executing this script will temporarily prevent any users from accessing the project geodatabase. Metadata must be stored and made available to any user to determine what date the data they are working with was collected. As new data is loaded, this metadata is to be updated by the user of the scripts. The metadata in the project geodatabase has been populated as part of this project, and dates of new data loading is all that should be required during updates.

This project hinges on the user being able to select a topic of interest. If a user is interested in schools, there is no reason to force them to sift through schools, streets, rail, and waterways data from OSM. There is so much data that this becomes overwhelming. Providing a tool for the user to target the correct data subset in analysis becomes a significant challenge. This is one of the key requirements of this project – make OSM data subsets usable by a GIS analyst by offering a menu of topics of interest for consideration by the analyst.

The requirement to define an area of interest with map interaction is understood to be that an analyst looking at ArcMap and the entire OSM dataset should be able to interactively identify where they are interested in performing analysis and what data subsets they require. Careful discussion with the client found that the user needs to be presented with

several options for extracting the subset of data that interests him. The results of this execution must be a defined area of interest and single, or set of, topics of interest where the analyst is going to perform analysis.

It is essential for this project that a user can perform analysis on extracted data. After executing the process within ArcMap, to select an area and topic of interest the data must be stored in a format on disk which can be utilized by geoprocessing tools, via Python scripts and the arcpy module, or via gp tools within ArcMap. The data can be stored in any native Esri format – File Geodatabase, Enterprise Geodatabase or Shapefiles.

3.2.2 Details of Nonfunctional Requirements

Table 2 presents the nonfunctional requirements of this project. Unlike functional requirements, nonfunctional requirements are the software and hardware needs. Following the table, as with the previous section, section 3.2.2 includes a detailed description of the specifics of the requirements.

Table 2. Nonfunctional Requirements

Requirement	Description
Internet accessibility	The user shall be required to have an active internet connection for the data download process. The user's client must support the REST web standard.
Hard drive capacity	The user shall maintain a hard drive large enough to handle a full continent download of OSM updates - ~300GB for North America, as well as space to store results of analysis (~100GB additional). If the user elects to complete the entire world dataset, they should expect ~8TB of space to support all processing.
ArcMap Add-In compatibility	The tool must execute within ArcMap, as an Add-In, compatible with ArcMap 10.7. Newer versions will require a re-compile of the add-in.
ArcObjects .Net SDK	The user facing interactive tools shall all be written in standard object-oriented patterns using the ArcObjects .Net SDK.

Python Compatibility	The scripts shall execute with Python 2.6 or 2.7.
bz2 Python module	The user shall have the bz2 Python module to perform all data extraction
Simple process	The user shall not be presented with more windows or choice options than are essential.

OSM data is stored in an online repository. The data is only available via internet download, thus *Internet accessibility* is a necessary requirement. The client was very focused on having all the processing of OSM data into a Geodatabase obfuscated from the user – they don't know how it gets processed, they just know that it gets processed. To support this the client must have internet connectivity. If the scripts are unable to connect to the internet or the specific REST endpoint for download an error will be presented.

The data from OSM is global, and lots of the data is duplicated. OSM stores the same features, lakes for example, as two completely different representations: one is the lines (ways) providing the boundary of the feature; the other is the points (nodes) representing the vertices of the feature. This duplication and the massive quantity of attributes results in datasets which are massive. North America is about 40GB compressed and about 300GB uncompressed. Because decompression is part of the process it is imperative that the client is executing on a machine with significant *Hard drive capacity*. If the process encounters a hard drive with no capacity remaining – an error will be presented to the user.

The interactive tool will execute using an ArcMap add-in. Add-ins are locked to a specific release of ArcGIS. *ArcMap add-in compatibility* upon release will be with ArcGIS Desktop 10.7. The ArcMap add-in will require being re-compiled to work with any different version of ArcGIS Desktop. The source code is provided to enable such an operation. The ArcMap add-in will be written using the ArcGIS 10.7 *ArcObjects .Net SDK*. The user will be required to have ArcGIS Desktop 10.7. The .Net SDK is not a requirement on the client machines, however the SDK will be required on any machine making edits or re-compiling the add-in.

The project will have *Python compatibility* with Python 2.6 and 2.7. A single Python script can be authored to support both versions of the Python.exe, therefore this project will support both. Python 3.5 and 3.6 are not considered backward compatible, therefore re-writes may be required to support these higher Python releases. The user must have the bz2 *Python module* installed on the machine that is executing the Python script, because this is required to extract the data file (.osm extension) from the compressed file provided by OSM data provider geofabrik (.bz2 extension).

The requirement that this be a *Simple Process* is subjective. As such, for this project we assume it is simple if the user can double click no more than two files to download, extract, transform and load the latest OSM data into the production geodatabase, without causing any issues for users actively using the system during the process.

3.3 System Design

The OpenStreetMap data extractor and translator is a series of scripts and tools which enable a client to download the latest OSM data, organize that data into a production geodatabase, identify features of specific topics of interest and geographic area. The system is multi-dimensional. There is an internet component and a local client component. OSM data is stored on a shared global database, which is only accessible via the internet. OSM can be accessed directly from a web browser, and the data can also be downloaded locally from OSM ftp servers. Tools which work with the OSM data that has been downloaded, extracted, translated and loaded into a Geodatabase, are provided for the client. Figure 3-1 shows a graphic demonstrating the interaction of the parts of the system, and how data flows through it.

Figure 3-1: Complete system interaction graphic

The user executes a Python script, from a local client, which connects, via the internet, using Representational State Transfer (REST) protocol, to the OpenStreetMap ftp server (Geofabrik). This script downloads a .bz2 file directly from OSM; the script then extracts this file using the bz2 module. Another script is then executed from a client machine which applies the updated data from the extracted OSM file into the production Geodatabase. The production Geodatabase is a File Geodatabase stored on the hard drive of the client executing the scripts, with a data model that is appropriate to represent the majority of OSM data.

Python 2.7 was used to author all the scripts necessary for the complete system. The ArcGIS Desktop 10.7 software package was used for all client interactions. The system is built to have the data loading scripts executed prior to any individual consulting project, but it does not do differences and delta updates and therefore is not designed to be executed during a project.

3.3.1 System Interface Requirements

This system requires minimal external interactions. This is a stand-alone application, and extension to ArcGIS Desktop. It is necessary that Python scripts are authored to correctly use the REST protocol, to access the publicly facing endpoints which OSM publishes through Geofabrik.

3.3.2 Hardware and Software Requirements

The project utilizes standard hardware and software for developing script-based applications, as well as standard .Net C# programming protocols. A Dell laptop running Windows 10 was used to develop all scripts and tools. The same laptop was used for the initial data loading, and processing. The only additional requirement was the use of a secondary 2 Terabyte (TB) hard drive to store all data. The data from OSM is massive, and a very large hard drive is essential.

The system is designed in a linear and repeatable manner:

1. Execute a Python script to:
 - a. Download .bz2 file
 - b. Extract to .osm – using the bz2 Python library
 - c. Load the .osm to an intermediate file geodatabase – using the OpenStreetMap extension toolset
2. Executed a second Python script to:
 - a. Organize, categorize and load data from the intermediate file geodatabase to the production geodatabase
3. Use a tool in ArcGIS Desktop to extract an area and theme of interest
4. Analyze and answer critical questions.

The production geodatabase will be built and designed to accurately represent all the types of data available from OSM. The target data model can be either custom built for this project, or an existing data model from a well-known source.

3.4 Project Plan

The project plan was broken into four separate phases: planning, design, implementation/testing, and deployment. During the planning phase the client and project team discussed the goals and scope of the project. Based on these conversations an initial plan was devised. Based on the original design a preliminary plan was put in place. The further sections will outline how this initial plan had to be adjusted, and how the timeline adjusted because of this. To begin, Table 3 outlines the preliminary plan based on initial conversations and design strategy.

Table 3. Preliminary Project Plan

Task Title	Hours
Plan	
Analyze Requirements	16
Gather Sources - Literature Review	40
Gather Sources - Commercial Product Reviews	4

Gather Info - Study/Learn Open Street Map REST API	8
Design	
Geodatabase - Data Model	12
Map Source Data to Data Model	10
User Experience	12
Buttons/Options	4
Other Potential Design Activity	40
Develop/Test	
Data Loading Scripts	40
Data Update Scripts	20
Desktop Application	40
Develop a User Experience	20
Communicate with PostgreSQL Geodatabase	10
Begin with North America (project scope)	16
Deploy	
Test on North America data	40
Pilot onto full dataset	40
Test complete application	40
Review, Regression	40
Deliver the application to the user	1
TOTAL HOURS:	453

3.4.1 Planning Phase

The client was originally looking for a tool to load OSM data into an Esri Geodatabase. The plan was to write an application to extract shapes and attributes, and store these in feature classes in a Geodatabase. During the research components of the planning stage it was discovered that an Esri business partner built a toolbox that already does this. As such, this requirement was dropped from the project plan.

It was also discovered that there are several companies that provide some variation of OSM data as Esri shapefiles. As the translation of OSM data into Esri formatted data was no longer a requirement of this project, the rest became more significant. The remainder of the preliminary plan was to analyze data models and determine which would keep the largest quantity of data and help to categorize it as correctly as possible. The project team had planned to produce a brand-new data model, something designed and built within the project. Again, in the research stage it was determined that Esri Solutions provides many data models and one of these might be appropriate for the project's purpose. The preliminary plan was adjusted to account for the additional time necessary to adequately and completely research the existing data models and determine if one of them would satisfy this project's requirements. Table 4 shows the adjusted project plan.

Table 4. Adjusted Project Plan

Task Title	Hours
Plan	
Analyze requirements	16
Gather Sources - Literature Review	40
Gather Sources - Commercial Product Reviews	4
Gather Info - Study/Learn Open Street Map REST API	8
Research Data Models	60
Design	
Geodatabase - Data Model	12
Map Source Data to Data Model	10
User Experience	12
Buttons/Options	4
Extraction Tool	5
Update Script	8
Develop/Test	
Data Loading Scripts	40
Data Update Scripts	20
Desktop Application	40
Develop a User Experience	20
Communicate with PostgreSQL Geodatabase	10
Work with North America	16
Deploy	
Test on North America Data	40
Pilot onto Full Dataset	40
Test Complete Application	40
Review, Regression	40
Deliver the Application to the User	1
TOTAL HOURS	486

Key items of the adjusted plan to take note of are that the design phase was pushed back by several months from the preliminary plan, as a lot of additional time was to be spent researching data models, which originally was not accounted for. Notice that the other parts of the plan remain unaffected – in terms of hours expected to be spent – only the starting date of each of the section has been adjusted to accommodate the data model research which proved necessary.

The Local Government Information Model (LGIM), from Esri Solutions was found to be a good data model candidate because it provides for the modelling of OSM data. The project's overall goal was to solve local problems; this is exactly what local

governments do daily. It became logical that the LGIM would include appropriate attributes and geography, to ensure success in solving problems at the local level.

3.4.2 Design Phase

During the design phase we began outlining what scripts would be required and what other tools would be necessary. We also began designing the user experience and interactions with the results of the project.

For this project, very minimal design was required. The client was looking for a basic application that just ‘got the job done’. The client was not interested in an immersive and engaging user experience, instead he was solely focused on a functional project which could be used to facilitate GIS consulting contracts for his company. We attempted to design scripts and tools which required minimal human interaction, so that the results of this project could be used as background and foundation to successful data analysis.

3.4.3 Implementation/Testing Phase

During the implementation/testing phase all the Python scripts were written and then tested incrementally. First, the script would be tested on a small subset of data by using a Python debugger (WingIDE) and walking through the script to ensure all functionality was working as expected. The ArcGIS Desktop Add-In was also authored in this stage, and it was tested through rigorous scenario-based testing. Every individual topic of interest was selected for extraction, and a multitude of locations were tested, ranging in size. There was a series of ad-hoc tests which were also executed to ensure system stability, selecting several combinations of topics of interest and diverse locations. The tool was used on several areas of the United States, ranging from extremely large areas of interest, to very local areas of interest. The client is provided performance results in Chapter 6 of this report.

3.4.4 Deployment Phase

During this phase metadata was added into all the geodatabase feature classes and comments were added to all the scripts and code. This ensures that the client is provided all information they need to setup the project on their own machine and run it successfully. Documentation was provided to the customer detailing all necessary setup and configuration steps.

3.5 Summary

As described above, the plan changed throughout the course of the project. The data extraction as a product of this project was eliminated early on, relying instead on existing tools. Despite the existing tools, data loading and extraction took 5 months of data processing. This was much longer than was initially designed. In hindsight, we should have allocated more time to that process. Scope modifications during the process also gave some breathing room that was necessary.

Originally the system was designed to provide the result of a thematic geodatabase storage for OSM data. After the project, the system looked like what was originally planned. However, the process had changed. The data was converted from OSM into Esri format via a tool provided by an Esri business partner. The massaging of the data from raw converted data into a data model was the largest focus of the project. It was essential to represent as many of the types of data as possible, that would be useful for future analyses. It was not necessary, for example, to maintain aerialway data. There are not that many analysis operations that would require knowing gondola locations, and the target data model, by default, does not contain any clear method of modelling such features.

Chapter 4 – Database Design

The database necessary to support this project had a few different requirements. The database had to contain a data model – collection of database objects (tables, feature classes) – that could represent the vast collection of features from OSM. Based on the research completed during the planning phase it was determined that the data model for the project would be the Local Government Information Model (LGIM), and therefore it was a requirement that any database platform used had to support the LGIM.

This chapter goes into detail about the database design process. We begin with a discussion of the conceptual data model, which is defining what the database must represent, features, relationships, and any other data integrity. Then we discuss the logical data model, this is the realization of the conceptual model but in practical format which can be applied to a geodatabase and used for the project. We then go into detail about the specifics of OSM data and its attributes, and what the LGIM represents and its attribution.

4.1 Conceptual Data Model

The conceptual design for this project was to model datatypes based OSM data. Conceptually the project did not want to lose any of the datatypes available in OSM. The project also wanted to eliminate bad data wherever possible. The project looked to leverage a data model that could represent most of the data from OSM, while removing unnecessary, incomplete or inaccurate data. The project looked to provide a system that could provide geographically accurate responses to the client about the types of data that exist, and where the data are. The client needs to answer basic analysis questions comparing types of data, and their geographies. For example, the client is interested in identifying Police Stations based on their proximity to parks. This analysis does not require knowing what type of police station, or how much staff it has. Just where it is located. Location questions based on datatypes are answered using a data model which stores accurate geometry but not necessarily any other attributes. The client is seeking a data model which links features by geometry. It is not necessary to build intra-database connections between these objects and features based on other attributes.

4.2 Logical Data Model

Discussions with the client established that using an existing data model would be acceptable, so long as the data model selected would allow for keeping a significant portion of the OSM dataset. There are datatypes that are dropped during the ETL process, because they would add size to the storage of the geodatabase on disc, without facilitating any analysis in which the client had an interest.

Initial thinking had the project designing its own unique ‘thematic’ data model. We began with defining all the datatypes of data represented by OSM. We then analyzed, what details about that data were necessary to answer the questions being posed by the client. While creating this data model, other data models were also considered. It was determined after analysis that the Local Government Information Model (LGIM),

because it is a very well defined and well researched model, would meet the needs of this project. The LGIM was designed by professionals that do local government management.

4.2.1 OpenStreetMap Data Model

OSM does not have what one would define as a true data model. There are no standards, and because data is stored in its most raw form the ‘data model’ is a primitive representation of geographic features as either ways (lines) or nodes (points). A data model can provide a series of datasets with some interrelations. OSM does have a concept of relations, allowing users to link multiple features together by some relationship, however, these are not persisted as one might expect with a more traditional data model and dataset. Instead these relations are used to generate polygon features from the input OSM Ways during the extraction portion of the ETL process.

4.2.2 Local Government Information Model

The LGIM contains a large collection of feature datasets, tables and feature classes. There are also complex dataset types, such as Network Datasets and Geometric Networks. This project is not interested in those advanced dataset types, allowing us to physically remove those objects from the Geodatabase. Only the data from the model which is applicable to this project is maintained. The primary feature dataset in the model that was of interest was the “ReferenceData” feature dataset. This was key because it is where a local government would store all its data for the types of analysis of this project.

The power of the LGIM is that it is a well-defined data model which enables maps and apps from ArcGIS Online, and the larger online GIS community. This project was focused on a client directly accessing only the OSM data, and not being interested in all the maps and apps that are enabled by using the Local Government Information Model. This allowed the project to only populate those datasets where OSM provided dataset types with a logical mapping into the LGIM classes (Appendix 2 – OSM Data Mapping Spreadsheet shows the taxonomy of type identifying attributes in OSM and how they were mapped to LGIM classes).

This project is not specifically focused on any individual local government, it is focused on doing the analysis at scales which a local government would traditionally use. Therefore, although the data model’s name suggests it is only useful for Local Governments, the reality is that the only difference between a local government and a large government is the quantity of data. This project is looking to support analysis that is similar to what a local government would do on a regular basis, and therefore the model appears appropriate for this type of a project.

4.2.3 OpenStreetMap Attribution

There is some key attribution from the OSM which this project looked to leverage, or at least maintain. The project was interested in making sure that features were categorized correctly during the ETL process, but also looked to make the processes flexible to be expanded upon in the future. This project truncates all tables in the project geodatabase at the start of data loading, for the purpose of ensuring the data is always the newest. In the

future if one wanted to work with deltas or differences, they would require some attributes to help with that process.

There were two attributes that this project focused on, they are: OSMID and OSMTAG. These attributes are enumerated below with their purposes relative to the overall project.

OSMID – Is a TEXT field in OSM but is translated into a Double precision number in the project geodatabase. This makes a future enhancement of comparing existing features in the project geodatabase and applying updates when they are available possible. This projects always begins with empty feature classes in the project geodatabase, but by maintaining this OSMID during the process (mapped to `osm_id` in the project geodatabase) it enables some future enhancements.

OSMTAGS – Is a BLOB (binary large object) field, and in the destination is being kept as a BLOB field. This field contains a wealth of additional information for every captured feature in OSM. At the time of this project we were focused on moving geometries and maintaining datatypes, and therefore some of the embedded information in these tags was not of immediate interest. However, a project should always aim to keep all data which is reasonable during an ETL process and moving this BLOB field from source to destination feature classes is simple and prevents this process from being overly lossy. This enables someone to write code to extract the information from the BLOB field and create a more detailed attribute record.

Not all attribution from OSM made sense to maintain in our ETL process, and there were some compromises that we had to make. We wanted to move all the BLOB information into an XML field because these enable full access within ArcGIS. During the ETL process an XML field would have enabled moving the XML attributes into their own specific attribute columns for each feature. However, the project was not scoped to enable the extraction of a BLOB datatype, conversion into a proper XML format and optionally relocating that XML data into other appropriate columns within the datasets. Therefore, it was decided that we would just maintain the exact information that OSM provided in its BLOB in our own BLOB field.

4.2.4 Local Government Information Model Attribution

The LGIM contains feature classes which have a robust attribution model. This project will leave all attributes on each of the feature classes but will only selectively insert data into the fields that are appropriate. The Shape field will contain the geometry. The ObjectID field contains a unique identifier. For this project we are expanding the LGIM by adding in 2 additional attributes, referenced in section 4.2.3 above, and visible in Figure 4-1, outlined in red, **osmtags** and **osm_id**.

Figure 4-1 shows the fields available in the ‘Park’ feature class in the ‘ReferenceData’ feature dataset in the LGIM, along with the additional fields added for the purposes of this project.

Figure 4-1: Complete field list on Park class that the LGIM provides with the OSM fields for this project added.

4.3 Data Sources

OpenStreetMap. All data is pulled directly from OpenStreetMap. Every record is tagged with an OpenStreetMap User Identifier (OSMID). This identifies the specific feature in the OSM dataset and this does not change over time. The data is provided in Geographic Coordinate System – GCS_WGS_1984. Valid projected coordinate system for this data, for the purpose of editing, and projecting onto a map is: WGS 1984 Web Mercator Auxiliary Sphere.

4.4 Data Collection Methods

No data collection was completed. Data was downloaded from an internet server, categorized and sorted in the Local Government Information Model.

4.5 Data Scrubbing and Loading

The data that was imported into the data model within the project geodatabase was only that which qualified as QA'd by OSM. There are many third-party tools built to QA data in OSM. Several commercial vendors offer professional QA'd versions of OSM data. However, to handle free data there is a basic QA process run by OSM themselves. This data is identifiable because the OSMTAGS include a special tag, which results in the extraction tool (Esri business partner provided) moving the information into appropriate

fields in the table and does not remain in the osmtags BLOB field. The BLOB field is maintained in the target datasets for complete attribution correctness. For example when a road is captured by a user in the field, they would tag it as 'road'. This value would be stored in the OSMTAGS as {attribute: 'road' [additional information of who captured it and when is also stored]}. When OSM QA's these entries, they would update the OSMTAGS for this feature to be: {Highway: 'road'}. This now tells extraction tools that this feature has been QA verified as being a road, and that information will now be written into the 'highway' field during the extract. There might still be other attributes remaining in the BLOB {attribute:'3 lanes', attribute:'divided' ...}. When those values have been QA'd they would be moved from the 'attribute' area of the BLOB into the more descriptive field that they will be assigned to. Using this process we can have faith that the data we are extracting and moving into the project geodatabase are of the correct data types.

4.6 Summary

The conceptual model and logical model in this project meshed very well. Conceptually we wanted to use a typing system already established by OSM to group like data, and identify what that data are, and therefore which feature class they should be inserted into in the project geodatabase. Logically, we used a set of well-defined feature classes from a data model provided by Esri's Solutions to represent as much information as possible. By using this implementation of a logical model, it allows a consultant to use this as a base of understanding and build more information and attribution into the data over time. The user could essentially use OSM to pull shapes and geometries, and then use other means to populate the remaining attributes, resulting in an extremely rich and detailed database, for very limited expense.

Chapter 5 – Implementation

5.1 Introduction

In chapter 5 we will be describing the overall implementation of this project. We will describe the specific tools and operations necessary to complete these tasks. As a means of providing additional context and information during this chapter we will be using a specific example. We are going to track roads, in the Chicago suburbs, a city called Naperville, Illinois.

5.2 Extracting OpenStreetMap data

The operation was completed using the OpenStreetMap for ArcGIS toolset. Using a Python script the data is extracted from a generic file type (.bz2) to an OpenStreetMap file type (.osm) using the Python bz2 library. The .osm file is loaded into a file geodatabase as points, lines and polygons. The imported data represents a basic data structure with no distinct or specific type system. The data is then translated and loaded into the LGIM – using a Python script, which uses sets of attributes to categorize each feature and then appropriately moves it into a feature class and that represents that specific feature type. There are also some features for which an attribute with a coded value domain are used to further clarify the type of the feature, where appropriate, this is applied during the loading portion of the script.

Referring to our example of understanding roads in Naperville. Each of these roads would be represented in the OSM dataset. The roads will begin compressed in the downloaded .bz2 file, and then decompressed into the .osm file, and then loaded into the file geodatabase as both a point and line representation. Points will represent the vertices and lines the overall shape of the roads. During translation we use attribution on the line features to capture roads as lines only, because the points are just duplicated and unnecessary information, and to load those roads into the appropriate ‘RoadCenterline’ feature class in the target LGIM in the file geodatabase. The example will continue, but we begin by looking at the data in its raw point, line, polygon form in the file geodatabase which is a direct dump of the .osm file. Figure 5-1 below shows a view of Naperville, Illinois looking at the raw point, line, and polygons representing all of the available data.

Figure 5-1: Raw .osm data converted to simple point, line, and polygon features in the area of Naperville, Illinois.

As seen above in figure 5-1, there is so much data here to look at, with no type system applied, it is quite difficult to discern what represents what. Also, the same feature is represented in many forms – roads as lines, and points, for example. Which is why a classification of this data is so crucial to the overall project, removing data bloat and providing data in the type system that is expected. A user, in our example, expects that roads will be represented as lines. The point representation of the lines is unnecessary, and therefore not brought to the final data model. This serves the purpose of limiting the size of the final dataset and providing analysts only the data types that would be of value. Figure 5-2 below demonstrates a case of this dual data representation in the source data, and further on in chapter 5 we will highlight the single representation, removing the data redundancy.

Figure 5-2: Viewing the dual data representation of a street in Naperville, Illinois.

5.3 Building an OpenStreetMap Taxonomy

To begin the research phase, a complete taxonomy of OSM was established. The downloaded OSM data was analyzed, and an Excel spreadsheet was generated listing all the attributes and information stored in OSM to allow a user to understand each of the feature types represented. This included discriminating between conflicting information on the same feature, and ‘cracking the OSM code’ to determine what the interconnected attributes mean, in terms of the feature being represented. The complete taxonomy can be viewed in Appendix 1: OSM Data Mapping Spreadsheet. A small sample is provided in Table 5 below, look specifically at the field named highway, and the values assigned in the corresponding columns to the right.

Table 5. Sample of the field names in OSM (left most column) and the values that appear in those columns indicating different feature types.

VALUES FROM OSM DATA:					
Line FC					
Field	Value1	Value2			
highway	cycleway	footway	path	pedestrian	steps
building	N/A				

natural_	cliff	strait	coastline	coastline name	ridge
waterway	canal	river	stream		
amenity	N/A				
landuse	N/A				
place	island	ilet			
railway					
boundary	yes	administrative	territorial		
power	minor line				
leisure					
man_made	pier	pipeline	wharf	embankment	
shop					
tourism					
route					
historic	yes				
aeroway	runway	taxiway	aerodrome		
aerialway	rope tow				
barrier	fence				
military					
geological					

In the example we are using to highlight the implementation of this project, we are looking to track roads during this process. To determine which features in the original OSM dataset represents roads, we have decoded the OSM attributes. In this case, it is the highway field which stores information about roads. There are many values that can be entered into this field, and not all of them represent roads. For example, as visible in Table 5 above, cycleway in the highway field, actually represents a bike lane, or bike path. Whereas, footway and pedestrian are two attributes which represent a sidewalk or pedestrian path where vehicles do not travel. In the coming section we will dive a little deeper into what specific attribution we are using to delineate roads from all other features using the highway OSM attribute.

5.4 Translating OpenStreetMap Data to Local Government Information Model

Using the taxonomy identified in the first step of implementation we built up a mapping, based on a collection of attributes, that identifies which feature class in the LGIM is an appropriate destination for each feature. Table 6 below shows a sample of the mapping spreadsheet that was generated to map specific attributes on features in OSM to a specific feature class and subtype in the LGIM. In the case of the road centerline data, there is no subtype field, but there is a field named RoadClass, which helps to discriminate which specific type of road we might be working with. This will allow us to better categorize the data and provide more accurate analysis on the output. Each feature class being

loaded in the LGIM was analyzed to determine whether or not it had a subtype, and if it did not have a subtype, if there was a field that could be used as a more detailed discriminator.

Table 6. A sample of the data mapping from OSM ‘highway’ to RoadCenterline feature class and appropriate road class attribute in the LGIM.

OSM Conveted LineFeatureClass		TARGET in LGIM	
Attribute	Value	Feature Class	ROADCLASS
Highway			
	[]residential	ReferenceData-RoadCenterline	Major Arterial
	[]tertiary	ReferenceData-RoadCenterline	Minor Arterial
	abandoned		
	access	ReferenceData-RoadCenterline	Service
	access_ramp	ReferenceData-RoadCenterline	Ramp
	bridleway	FacilitiesStreets-Trail	
	bus_guideway	ReferenceData-RoadCenterline	Service
	bus_stop		
	byway	ReferenceData-RoadCenterline	Highway
	construction	ReferenceData-RoadCenterline	Other
	corridor	ReferenceData-RoadCenterline	Highway
	cycleway	FacilitiesStreets-StreetBikeLane	
	demolished	ReferenceData-RoadCenterline	Unknown
	dismantled	ReferenceData-RoadCenterline	Unknown
	disused	ReferenceData-RoadCenterline	Unknown
	elevator		
	emergency_access_point	ReferenceData-RoadCenterline	Service
	escalator		
	escape	ReferenceData-RoadCenterline	Service
	footway	FacilitiesStreets-Trail	
	ford		
	junction		
	landing		
	living_street	ReferenceData-RoadCenterline	Other
	motorway	ReferenceData-RoadCenterline	Freeway
	motorway[]	ReferenceData-RoadCenterline	Freeway

Table 6 above highlights a further list of the highway attribute values that are permissible within QA inspected OSM datasets. Each of these attribute values represent what the person participating in collecting the data for OSM identified the feature as, and was confirmed using a QA process. Turning back to our example of identifying roads in Naperville, Illinois, at this point of the implementation we are still looking at the features on a dataset wide scale. We are not limiting the searches or data gathering to the extent of Naperville, that is a choice the analyst can make at the time they choose. We are looking

to decide what value from the highway field in the raw OSM dataset maps to a road, and specifically, what type of road it represents. There are also values in this field such as footway, which do not map to a road at all. As seen in Table 6 above, when we discover the value footway in the highway field, we map it to the Trail feature class, a class within the LGIM used to track footpaths. The project worked to discriminate as much as possible so that data was represented in only a single location, in a single shape type, in a place that an end user would expect to locate that type of feature.

5.4.1 Loading Data into the Correct Target Feature Class

A Python script was authored which utilized the mapping created above to use search and insert cursors in Python to locate the appropriate data, and insert it into the correct feature class, and subtype, or to apply the discriminator field attribute, in the LGIM. The Python script is available in complete form in Appendix A: Data Loading Python script. Figure 5-3 below shows a small snippet of the Python code which performs the data loading operation.

```
targetClass = "RoadCenterline"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0

whereClause = "highway = 'motorway' OR highway = 'byway' OR " +
              "highway = 'primary' OR highway = 'secondary' OR " +
              "highway = 'trunk' OR highway = 'secondary' OR " +
              "highway = 'tertiary' OR highway = 'service' OR " +
              "highway = 'residential' OR highway = 'road' OR " +
              "highway = 'unclassified' OR highway = 'motorway_link' OR " +
              "highway = 'trunk_link' OR highway = 'primary_link' OR " +
              "highway = 'secondary_link' OR highway = 'living_street' OR " +
              "highway = 'bus_guideway' OR highway = 'ford' OR " +
              "highway = 'platform' OR route = 'road'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourceLineClass, ["OSMID", "osmtags",
                                                                              "SHAPE@", "highway", "route"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass, ["ROADCLASS",
                                                                              "osm_id", "osmtags",
                                                                              "Shape@"])

for row in searchRows:
    if row[3] == 'motorway' or row[3] == 'biway':
        insertRows.insertRow(["Freeway", float(row[0]), row[1], row[2]])
    if row[3] == 'primary':
        insertRows.insertRow(["Highway", float(row[0]), row[1], row[2]])
    if row[3] == 'secondary' or row[3] == 'trunk':
        insertRows.insertRow(["Major Arterial", float(row[0]), row[1], row[2]])
    if row[3] == 'secondary' or row[3] == 'tertiary':
        insertRows.insertRow(["Minor Arterial", float(row[0]), row[1], row[2]])
    if row[3] == 'service':
        insertRows.insertRow(["Service", float(row[0]), row[1], row[2]])
    if row[3] == 'residential':
        insertRows.insertRow(["Local", float(row[0]), row[1], row[2]])
    if row[3] == 'motorway_link' or row[3] == 'trunk_link' or row[3] == 'primary_link' or row[3] == 'secondary_link':
        insertRows.insertRow(["Ramp", float(row[0]), row[1], row[2]])
    if row[3] == 'road' or row[3] == 'unclassified' or row[3] == 'living_street' or row[3] == 'bus_guideway' or row[3] == 'ford':
        insertRows.insertRow(["Other", float(row[0]), row[1], row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
```

Figure 5-3: A sample of mapping the OSM highway attribute to the appropriate RoadCenterline feature class using an insert cursor in Python

Figure 5-3 above shows the part of the data loading script which specifically relates to the example case we are using during this chapter. To maps roads, we begin by

building a where clause, which includes all of the conditions that OSM uses to track a road. For example: highway = motorway or highway = residential. All of the conditions are linked together using an OR statement, because if any of those attributes are set, then the feature represents a road. A search cursor is populated first, from all the records that satisfy the where clause (query) built up. And an insert cursor is then opened on the target 'RoadCenterline' feature class in the LGIM, targeting specific fields. The RoadClass field is included, because this is where we will populate the discriminating information to help delineate what type of road is being represented. The osm_id and osmtags fields are added because, as described early in this report in Chapter 4, keeping this information allows us to mitigate data loss. Lastly, we insert the shape because in a GIS the shape/geometry of a feature is essential, and this is how we support the rest of the system.

Figure 5-4 below shows the result of executing the data loading script in Naperville, Illinois. As can be seen in the Table of Contents, on the left side of the image we have the data which was originally just a point, line, and polygon feature class segregated into many feature classes, each representing a specific data type or set of data types.

Figure 5-4: View of Naperville, Illinois with the organized full OSM dataset in the LGIM.

Figure 5-4 above highlights that completing the data extraction, and translation process. We are in a position to begin GIS analysis. However, we are working with datasets that are huge, For example, as shown in Figure 5-5 below, the Roads class contains 19,553,321 road segments.

OBJECTID *	Shape *	Centerline ID	Left From Address	Left To Address	Right From Address
1	Polyline	<Null>	<Null>	<Null>	<Null>
2	Polyline	<Null>	<Null>	<Null>	<Null>
3	Polyline	<Null>	<Null>	<Null>	<Null>
4	Polyline	<Null>	<Null>	<Null>	<Null>
5	Polyline	<Null>	<Null>	<Null>	<Null>
6	Polyline	<Null>	<Null>	<Null>	<Null>
7	Polyline	<Null>	<Null>	<Null>	<Null>
8	Polyline	<Null>	<Null>	<Null>	<Null>
9	Polyline	<Null>	<Null>	<Null>	<Null>
10	Polyline	<Null>	<Null>	<Null>	<Null>
11	Polyline	<Null>	<Null>	<Null>	<Null>
12	Polyline	<Null>	<Null>	<Null>	<Null>
13	Polyline	<Null>	<Null>	<Null>	<Null>

Figure 5-5: View of the complete RoadCenterline feature class following data extraction.

Figure 5-5 above shows the reason that building a data extraction add-in in ArcMap is important for this project. The analyst needs to be able to work with reasonable sized data, otherwise it can be incredibly difficult to manage. Trying to build a Network Dataset, or even run a basic Geoprocessing operation/tool, on a class with 19 million records will be inherently slow. The next section addresses a tool to help deal with this.

5.5 Creating a .Net Application to Extract a Subset of Data

There were a minimal number of user requirements provided for the ArcGIS Desktop tool that would be authored. The client was primarily interested in executing the data extraction using a Python script and passing in bounding box coordinates. However, to be able to provide this tool to others, who are not as technically savvy, building a user interactive tool was appropriate. The tool was designed to allow a user to open a map, with the OSM data added but not displayed. Using a basemap, the user navigates to their area of interest. The user selects from a pick list, the data they are interested in. The user then clicks a button to generate an area of interest and begins clicking the map outlining their area of interest. When the user double clicks the drawing is complete and the area of interest is defined. Figure 5-6 below shows the user experience for selecting the topics or interest that the analyst is interested in working with.

Figure 5-6: The User experience that allows an analyst to select the type of data they would like to use for analysis.

It would be typical to expect one, two or three selected options for any given analysis. However, figure 5-6 above shows that an 'All' option was provided in case the user decides for the area of interest and project they are working on, everything matters and must be included. Warning, that selecting 'All' will significantly increase the amount of time for the data extraction to complete. Chapter 6 of this report includes a small sampling of performance metrics for the user to reference.

Figure 5-7 below shows what the user sees after clicking on the map to define their area of interest. The user is then able to click the 'Extract the Data' button, as seen above in figure 5-6, and the tool extracts all the data for the themes the user has selected for the area of interest.

Figure 5-7: The User is shown a graphic such as this for the area that will be extracted for their analysis.

The map is updated with re-sourced layers pointing to the newly extracted data. The new data is grouped together and ready for analysis. Default symbology is provided as a result of the extraction process.

Referring back to figure 5-6 above, notice there are two options for extraction ‘streets’, with or without a network. If the user selects ‘All’ they will get streets with a network. This means, that once the streets are extracted a simple Network Dataset is built on those streets, enabling driving directions, service area and drive time calculations. Because of limits in the translation, street names are not accessible, and would require additional work to pull the information from the OSMTAGS into the appropriate LGIM fields. Therefore turn-by-turn navigation is not enabled in the current network product.

The tool was written using the ArcObjects .Net SDK because this provides seamless interaction between the user and ArcGIS desktop. This tool could be extended to work with ArcGIS Pro, and the new Pro Managed SDK, but that would be for future consideration. Using ArcObjects allowed the project to access geodatabases and data directly and create extractions in the most efficient way possible.

5.6 Summary

As described in Chapter 5, several Python scripts and a user facing desktop tool were authored to enable an analyst with access to extract data for any area of North America that they are interested in for the themes of data that interest them. This implementation enables users to begin working with OSM data to solve problems at the local level.

Chapter 6 will go into the results of the implementation, including sample map outputs and performance metrics which indicate the usability of the tool.

Completing the example we worked through this chapter, we have tracked roads in Naperville, Illinois from dual point and line representations in raw OSM, translated into line only roads in LGIM, and extracted using an area of interest polygon and an ArcMap add-in tool to enable detailed analysis. Figure 5-8 below wraps up this example, displaying the roads, as a traversable network dataset, in Naperville, Illinois.

Figure 5-8: Roads in Naperville, Illinois, extracted from OSM source data, with a route between three locations displayed.

Figure 5-8 above includes a simple route, showing that when the user chooses to extract the street, with network, they do receive a fully functional, and traversable street network.

Chapter 6 – Results and Analysis

Based on the test cases executed, the project can update from OSM into a geodatabase data model with a very detailed system for defining types of features. The output data model of the Local Government Information Model serves the use cases very well and enables analysis at most local scales. There is value brought by this project to any analyst looking for free and available data for local data analysis using ArcGIS Desktop. The project provided an ETL process which remains consistent and is flexible so that it can be expanded overtime to enable further processing. Chapter 7 will discuss the potential future expansions of this project. The results of this project allow a GIS Consultant to answer many questions, all over North America.

6.1 Test Cases

Several test cases were posed, ensuring that the application and resulting OSM data could be used to answer questions that might be of interest to a GIS analyst, consultant or contractor.

The test case questions are as follows:

1. Identify all residential buildings within 100 meters of a railroad in the city of New Orleans, Louisiana
2. Determine areas that are not within a 15-mile drive of any police station in Birmingham, Alabama, helping identify areas that might be underserved by police.
3. Identify all forests larger than 20 km² in and around Portland, Oregon, helping fire fighters identify wildfire risk locations.

To answer all test questions we ensure that we are working with updated data. We execute the two Python scripts, so that we have a copy of the project's local government information model with the latest OSM dataset loaded.

1. In order to identify all residential buildings within 100 meters of a railroad in the city of New Orleans Louisiana, we begin by opening the project's ArcMap 'Data Processing' map. We zoom to the location on the map where we are interested in working, in this case, New Orleans, Louisiana – using the basemap as a guide. We then click the button on the customized toolbar to launch the data extraction tool. We select buildings and railroads, and draw a polygon around New Orleans, Louisiana. We wait a few seconds for the operation to complete, and re-source the layers in the map. We now have the specific data in the specific area of interest. This will enable us to complete the actual analysis operations. Begin by creating a buffer around all railroads which is 100 meters wide. Use the Buffer geoprocessing tool to accomplish this task, input is Railroads, output: '100m railroad buffer'. After the tool runs, the buffer is added to the map automatically. We then use the Select by Locations tool, to locate all buildings which intersect the 100m railroad buffer layer. Once the appropriate buildings are selected, we use the Make Layer from Selection tool, to generate a new layer of those buildings that are within 100 meters of the railroads. We then symbolize the new layer using a red

polygon symbol. We chose to then turn off the buildings layer (which includes those within and outside of the 100 meter buffer), to clean up the map display, and highlight the specific buildings we are interested in. The result of this operation, and the answer to the test question posed is shown on the following map:

Buildings within 100 meters of Railroads in New Orleans

Figure 6-1 : Map showing the ability to extract specific data around New Orleans to answer a question posed.

2. In order to determine areas that are not within a 15-mile drive of any police station in Birmingham, Alabama, allowing people to see areas that might be underserved by police, we begin by opening the project's ArcMap 'Data Processing' map. We zoom to the location on the map where we are interested in working, in this case, Birmingham, Alabama – using the basemap as a guide. We then click the button on the customized toolbar to launch the data extraction tool. We select Public Safety and Streets (with network), and draw a polygon around Birmingham, Alabama. We wait a few seconds for the operation to complete, and re-source the layers in the map. We now have the specific data in the specific area of interest. This will enable us to complete the actual analysis operations. Begin by opening the Network Analyst toolbar, and creating a new Service Area. We then right click the new Service Area group layer, and go to its properties. On the 'Analysis Settings' tab, we set the Impedance to Length (meters) and set the Default Breaks to: 8045, 16090, 24135. This translates to 5 miles, 10 miles, 15 miles. Next we launch the 'Network Analyst Window' using the button on the toolbar, and right click the 'Facilities' layer, and select 'Load Locations'. Load the police station in the study area as facilities. Using the Network Analyst toolbar click 'Solve'. This will display, using the road network, service area polygons. Any road outside of the largest polygon is outside of 15 miles from a police station. use the Select by Locations tool, to select all roads that are not within the 15 mile service area. Use the Make Layer from Selection tool, to generate a new layer of those road segments that are more than 15 miles from the nearest police station. We then symbolize the new layer using a red line symbol. The result of this operation, and the answer to the test question posed is shown on the following map:

Areas of Birmingham more than 15 Miles from Police

Figure 6-2: Map showing the ability to extract data and leverage a Network Dataset built on the extracted data.

3. To Identify all forests larger than 20 km² in and around Portland, Oregon, helping fire fighters identify wildfire risk locations, we begin by opening the project's ArcMap 'Data Processing' map. We zoom to the location on the map where we are interested in working, in this case, Portland, Oregon – using the basemap as a guide. We then click the button on the customized toolbar to launch the data extraction tool. We select Landuse and draw a polygon around Portland. Forests are large areas, so we extended this area of interest north to Olympus and south to Eugene and east to Bend to include as many forests in the area as possible. We wait a few seconds for the operation to complete, and re-source the layers in the map. We now have the specific data in the specific area of interest. This will enable us to complete the actual analysis operations. Begin by using the Select by Locations tool, to locate all forests which are 20km² or larger. Once the appropriate forests are selected, we use the Make Layer from Selection tool, to generate a new layer of those forests that are large. We then symbolize the new layer using a green polygon with a thick red outline, so that it stands out against other forests and the basemap. The result of this operation, and the answer to the test question posed is shown on the following map:

Forests at Risk of Wildfires Near Portland, OR

Figure 6-3: Map showing the ability to extract polygons to answer questions.

6.2 Performance Numbers

The user should be aware of two unique performance aspects of this project. The first is the amount of time it takes to download the .bz2 file, extract it to an .osm file, and then the time taken for the initial load of this .osm file into a geodatabase. This process is time and resource intensive. And can have wildly different processing times depending on the location of the geodatabase. It was discovered that loading the .osm file into a geodatabase stored on an external hard drive took orders of magnitude longer than performing that same operation using a hard drive internal to the machine doing the work. Executing to an external hard drive took 2604 hours to complete, approximately 3 months of processing time. However, the same execution to a local hard drive took 72 hours to complete, approximately 3 days of processing time. It will be important to work with the hard drive that is internal to the machine to avoid this performance problem.

The second consideration for performance of this project is how long it can take to generate an extraction based on the amount of data being requested, and the number of themes of data being requested. Table 7 presents data that indicates the scales where this type of a tool is successful. Using this tool to look at all of the data from OSM at any scale is not really reasonable. There is so much data that the entire country takes over 1 hour and 30 minutes. This demonstrates that this project should be leveraged to target specific data for specific analyses, and different data for different analyses. The tool will continue to work at all provided scales, but the value diminishes if too much data is requested. It also defeats some of the project's purpose, to classify the data, and segregate it for use in specific analytic operations that are answering local questions.

Table 7. Performance numbers captured for extraction a collection of themes at 3 set scales (city [Redlands, CA], state [Louisiana], Country [USA])

Themes Selected	Size of Region	Wall Clock Time Taken
All	City (Redlands, CA)	2 minutes 20 seconds
All	State (Louisiana, USA)	5 minutes 17 seconds
All	Country (USA)	1 hour and 30 minutes
Rail, Streets	City (Redlands, CA)	1 minute 23 seconds
Rail, Streets	State (Louisiana, USA)	2 minutes 20 seconds
Rail, Streets	Country (USA)	44 minutes, 16 seconds
Public works, education	City (Redlands, CA)	4-5 seconds
Public works, education	State (Louisiana, USA)	21 seconds

Public works, education	Country (USA)	1 minute 15 seconds
Landuse, water	City (Redlands, CA)	3 seconds
Landuse, water	State (Louisiana, USA)	12 seconds
Landuse, water	Country (USA)	9 minutes 5 seconds

To achieve the above performance numbers, all timing was done with a standard stopwatch. Hardware used: Dell Precision 5520 Laptop (32 GB of RAM, Intel core i7 duo core @ 2.90 GHz), using standard ArcMap 32-bit processing. In the analysis there are no specific benchmarks for acceptable time, however, for a city scale to extract all of the data available, and it is a significant amount of data, 2 minutes and 30 seconds felt workable. The intention is that the user can accomplish several extractions, and detailed analysis in one day. Waiting a few minutes for a city worth of data to extract for use in analysis seems reasonable. To extract the whole of the USA taking approximately one and a half hours seems long, but it is also not the described intention of this project. The project is designed to take a huge amount of data and massage it to a manageable quantity for a user to accomplish a locally scaled analysis. For these purposes a few minutes of extraction time is a small price to pay, to have a project built on top of free data.

Chapter 7 – Conclusions and Future Work

7.1 Project Summary

This project set out to provide an analysis and understanding of volunteered geographic information (VGI), OpenStreetMap (OSM), and intended to make a determination of whether this type of data could be used for GIS consultations. The project intended to provide a consistent, repeatable, operation that would take data from an open sourced dataset, translate it into a widely consumable format.

Using academic articles and background research, it was determined that there was an appropriate amount of trust that could be assigned to VGI, and specifically the quality of data in the OSM dataset. While deep attribution continues to improve, it is not at a level that we could call complete, the location represented for most features is quite accurate. As additional time passes and more people around the world continue to contribute to the OSM dataset the attribution record will become more complete, and the location and shape of features will be more accurate.

This project was expected to offer a translation of data from OSM format (not native to any common GIS system without paying for a commercial version of the dataset), into a format recognized and usable by Esri's ArcGIS Platform. The project designed two Python scripts to accomplish this task, and was successful. The first Python script connects to a known OSM ftp site and downloads the raw data in a compressed format (bz2). The script then extracts the file into the native format (.osm), and then uses an Esri extension toolbox to extract this file into a raw point, line and polygon feature class in an Esri File Geodatabase. The second script then executes, which leverages 64-bit background Geoprocessing to load the data from the raw point, line and polygon, into an appropriate data model for use in consulting projects.

This project analyzed several data models available for representing the OSM dataset in a way that would best facilitate analysis and use in the Esri ArcGIS Platform. It was concluded that the Local Government Information Model (LGIM) was the most appropriate data model to use, and therefore was the target dataset implemented by this project.

Overall the project determined that moving data from OSM into an Esri Geodatabase was a manageable operation, and the resulting data was of sufficient quality for use in consulting contracts.

7.2 Future Work

This project enables OSM data for use by anyone. There are still long processing times, because of the sheer quantity of data. Future work could be dedicated to examining the OSM data prior to extraction, to determine if there is a more efficient way to load it into an Esri accepted format. Within the general context of this report, future project could apply this projects methodology and tools to the rest of the globe and extract more information from OSMTAGS.

There are innumerable uses for cheap, or free, high quality data. Consultants are always looking to underbid each other and provide the best quality output for the

cheapest possible cost. This project focused on the OpenStreetMap dataset, but there are many other open source data providers. Future projects could evaluate and compare all of the open source data in the GIS landscape and determine which sets are of a sufficient quality and consistency to satisfy high quality project deliveries.

A future project, in the same general area as this project, could evaluate Esri's community basemap project. This is a series of high precision basemaps populated by the Esri user community and hosted on ArcGIS Online for all users to consume. Other future projects could consider more directly integrating with the more modern GIS solution, ArcGIS Pro.

A future project could move entirely away from Esri's platform and software altogether, and instead evaluate the effectiveness of Q-GIS (an open source GIS application) or MapInfo, for working with OpenStreetMap data, and providing analytic capability.

There are a plethora of future projects that could center around the acquisition, data cleansing, data transformation and presentation of open sourced data and volunteered geographic information for satisfying contractors, or just general GIS needs.

Works Cited

- Barron, C., Neis, P., & Zipf, A. (2013). A Comprehensive Framework for Intrinsic OpenStreetMap Quality Analysis. *Transactions in GIS*, 18(6), 877-895.
- Ciepluch, B., Jacob, R., Mooney, P., & Winstanley, A. C. (2009). Using OpenStreetMap to Deliver Location-based Environmental Information in Ireland. *Seventeenth ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, (pp. 17-22). Seattle.
- Footy, G. M., See, L., Fritz, S., Van der Velde, M., Perger, C., Schill, C., & Boyd, D. S. (2013). Assessing the Accuracy of Volunteered Geographic Information arising from Multiple Contributors to an Internet Based Collaborative Project. *Transactions in GIS*, 847-860.
- Girres, J.-F., & Touya, G. (2010). Quality Assessment of the French OpenStreetMap Dataset. *Transactions in GIS*, 14(4), 435-459.
- Hochmair, H. H., Zielstra, D., & Neis, P. (2014). Assessing the Completeness of Bicycle Trail and Lane Features in OpenStreetMap for the United States. *Transactions in GIS*, 19(1), 63-81.
- Mooney, P., & Corcoran, P. (2012). The Annotation Process in OpenStreetMap. *Transactions in GIS*, 16(4), 561-579.
- Mooney, P., & Corcoran, P. (2013). Analysis of Interaction and Co-editing Patterns amongst OpenStreetMap Contributors. *Transactions in GIS*, 18(5), 633-659.
- Open Street Map API v0.6*. (2016). Retrieved from Wikipedia:
http://wiki.openstreetmap.org/wiki/API_v0.6#Capabilities:_GET_.2Fapi.2Fcapabilities

Appendix A. Python Script Mapping Extracted OSM data to LGIM Feature Classes

```
##-----
## MIP Data Loading script - From OSM format into the new data model
## Author: David Crawford
##
## Created: 26/10/2016
## Description: This is an application which takes OSM feature data and moves it into a known data model.
##             The data model is consistent with the type of data necessary to represent. Each source data feature
##             has either a subtype or a full class to be entered into.
##
##
##             The script is divided by datatype. First we load Polylines, then Points, then Polygons.
##             At the top of each section is a listing of all of the classes from the
##             LGIM, and then a mapping of the OSM tags/fields which
##             will be loaded into that class, in cases where there is a further
##             discriminator (subtype, or type field) that info is included
##             in this mapping table.
##
##
##             Future Note: Over time there may be additional attributes mapped in OSM, at
##             which point this script could be augmented to populate
##             additional classes, or to add conditions onto the whereClause that
##             would allow additional features to be loaded
##             into the LGIM for this project. (The companion excel document lays
##             out the entire taxonomy of the OSM attributes that were mapped - there
##             are additional information in the OSMTAGS field (streetname, for example) that could be harvested and used
##             to make the LGIM for this project even more robust and detailed.)
##-----
import arcpy
sourceWorkspace = r"D:\MIP\MIP\OutputData.gdb"
sourcePointClass = r"NorthAmerica\NorthAmerica_osm_pt"
sourceLineClass = r"NorthAmerica\NorthAmerica_osm_ln"
sourcePolygonClass = r"NorthAmerica\NorthAmerica_osm_ply"

#Target workspace
targetWorkspace =
r"C:\Users\dave6199\Pictures\Personal\MIP\Final_Scripts_and_Data\NorthAmerica_VikingGeographicDatabase.gdb"
```

```
#####POLYLINE
FEATURES!#####
```

```
## ----- Target Classes to be populated (in the LGIM model)-----##
```

```
#          PLSSBoundary
#          CIPPolylines
#          wMainsWithTraceSummary
#          FiveFootContour
#          TenFootContour
#          TwentyFootContour
#          TwoFootContour
#          BuildingFloorplanLine
#          BuildingFloorplanPublish
#          Guardrail
#          PavementMarkingLine
#          Sidewalk                                | highway = 'pedestrian' OR 'footway'
OR 'steps'
#          SiteAmenityLine
#          Street
#          StreetBikeLane                          | highway = 'cycleway'; route = 'bicycle'
#          StreetCurbType
#          StreetFunctionalClass
#          StreetLaneWidth
#          StreetNumberLane
#          StreetOwnership
#          StreetPavementType
#          StreetSnowRoute
#          StreetSpeedLimit
#          StreetTruckRoute
#          Trail                                    | highway = 'path' OR
'bridleway' OR 'track' OR 'raceway'; leisure = 'track'; route = 'hiking'

#          BridgeMaintAgreement
#          LandscapeMaintAgreement
#          PaveMarkMaintAgreement
#          RoadBlock
#          RoadDetour
#          SanitaryMaintAgreement
#          SignalMaintAgreement
#          SignMainAgreement
```



```

#           StormwaterMaintAgreement
#           StreetlightMaintAgreement
#           StreetMaintAgreement
#           WaterMaintAgreement

#           Levee
#           PreIncidentPlanLine

#           Railroad |           railway = 'rail' OR 'tram' OR
'light_rail' OR 'subway' OR 'monorail'; route = 'railway' OR 'train' OR 'tram'
#           RoadCenterline | RoadClass (Freeway, Highway,
Ramp, Major Arterial, Minor Arterial, Collector, Local, Service, 4WD, Recreation, Resource, Ferry, Other, Unknown)
#           highway
= 'motorway' OR 'byway' (Freeway); highway = 'primary' (Highway); highway = 'secondary' OR 'trunk' (Major
Arterial);
#           highway
= 'secondary' OR 'tertiary' (Minor Arterial); highway = 'service' (service); highway = 'residential' (Local);
#           highway
= 'road' OR 'unclassified' OR (Other); highway = 'motorway_link' OR 'trunk_link' OR 'primary_link' OR
'secondary_link' (Ramp);
#           highway
= 'living_street' OR 'bus_guideway' OR 'ford' OR 'platform' (Other); route = 'road' (Other)
#           Waterline |           waterway = 'stream' OR
'river' OR 'canal' OR 'ditch' OR 'drain' OR 'weir' OR 'dam'; natural = 'coastline'

#           ssCasing
#           ssGravityMain
#           ssLateralLine
#           ssOpenDrain
#           ssPressurizedMain
#           ssVirtualDrainline

#           swCasing
#           swCulvert
#           swGravityMain
#           swOpenDrain
#           swPressurePipe
#           swVirtualDrainline

#           wAbandonedLine
#           wCasing
#           wConstructionLine
#           wLateralLine

```

```

#                               wMain                               |                               man_made = 'pipeline'
## -----##

targetFD = "FacilitiesStreets"
targetClass = "Sidewalk"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
i = 0
whereClause = "highway = 'pedestrian' OR highway = 'footway' OR highway = 'steps'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\" + sourceLineClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\" + targetFD + "\\" + targetClass, ["osm_id", "osmtags",
"Shape@"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], row[2]])
    print("row: " + str(i) + "osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

targetClass = "StreetBikeLane"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
i = 0
whereClause = "highway = 'cycleway' OR route = 'bicycle'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\" + sourceLineClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\" + targetFD + "\\" + targetClass, ["osm_id", "osmtags",
"Shape@"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], row[2]])
    print("row: " + str(i) + "osmid: " + row[0])
    i += 1
    del row

del searchRows

```

```

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetClass = "Trail"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

i = 0

whereClause = "highway = 'path' OR highway = 'bridleway' OR highway = 'track' OR highway = 'raceway' OR leisure
= 'track' OR route = 'hiking'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourceLineClass, ["OSMID", "osmtags",
"SHAPE@", "highway"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["HIKING", "EQUESTRIAN", "osm_id", "osmtags", "Shape@"])

for row in searchRows:
    if row[3] == 'bridleway':
        insertRows.insertRow(["", "YES", float(row[0]), row[1], row[2]])
    else:
        insertRows.insertRow(["YES", "", float(row[0]), row[1], row[2]])

    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetFD = "ReferenceData"
targetClass = "Railroad"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

i = 0

whereClause = "railway = 'rail' OR railway = 'tram' OR railway = 'light_rail' OR railway = 'subway' OR railway =
'monorail' OR route = 'railway' OR route = 'train' OR route = 'tram'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourceLineClass, ["OSMID", "osmtags",
"SHAPE@", "railway", "route"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["RAILTYP", "osm_id", "osmtags", "Shape@"])

for row in searchRows:
    if row[4] == 'tram' or row[3] == 'tram':

```

```

        insertRows.insertRow(["Tram", float(row[0]),row[1],row[2]])
    if row[4] == 'train' or row[4] == 'railway' or row[3] == 'rail':
        insertRows.insertRow(["Rail", float(row[0]),row[1],row[2]])
    if row[3] == 'monorail':
        insertRows.insertRow(["Monorail", float(row[0]),row[1],row[2]])
    if row[3] == 'light_rail':
        insertRows.insertRow(["Lightrail", float(row[0]),row[1],row[2]])
    if row[3] == 'subway':
        insertRows.insertRow(["Subway", float(row[0]),row[1],row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetClass = "RoadCenterline"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0

whereClause = "highway = 'motorway' OR highway = 'byway' OR highway = 'primary' OR highway = 'secondary' OR
highway = 'trunk' OR highway = 'secondary' OR highway = 'tertiary' OR highway = 'service' OR highway = 'residential'
OR highway = 'road' OR highway = 'unclassified' OR highway = 'motorway_link' OR highway = 'trunk_link' OR
highway = 'primary_link' OR highway = 'secondary_link' OR highway = 'living_street' OR highway = 'bus_guideway'
OR highway = 'ford' OR highway = 'platform' OR route = 'road'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourceLineClass, ["OSMID", "osmtags", "SHAPE@",
"highway", "route"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["ROADCLASS","osm_id","osmtags", "Shape@"])

for row in searchRows:
    if row[3] == 'motorway' or row[3] == 'biway':
        insertRows.insertRow(["Freeway", float(row[0]), row[1], row[2]])
    if row[3] == 'primary':
        insertRows.insertRow(["Highway", float(row[0]), row[1], row[2]])
    if row[3] == 'secondary' or row[3] == 'trunk':
        insertRows.insertRow(["Major Arterial", float(row[0]), row[1], row[2]])
    if row[3] == 'secondary' or row[3] == 'tertiary':
        insertRows.insertRow(["Minor Arterial", float(row[0]), row[1], row[2]])
    if row[3] == 'service':

```

```

        insertRows.insertRow(["Service", float(row[0]), row[1], row[2]])
    if row[3] == 'residential':
        insertRows.insertRow(["Local", float(row[0]), row[1], row[2]])
    if row[3] == 'motorway_link' or row[3] == 'trunk_link' or row[3] == 'primary_link' or row[3] ==
'secondary_link':
        insertRows.insertRow(["Ramp", float(row[0]), row[1], row[2]])
    if row[3] == 'road' or row[3] == 'unclassified' or row[3] == 'living_street' or row[3] == 'bus_guideway' or
row[3] == 'ford' or row[3] == 'platform' or row[4] == 'road':
        insertRows.insertRow(["Other", float(row[0]), row[1], row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetClass = "waterline"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0

whereClause = "waterway = 'stream' OR waterway = 'river' OR waterway = 'canal' OR waterway = 'ditch' OR waterway
= 'drain' OR waterway = 'weir' OR waterway = 'dam' OR natural = 'coastline'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourceLineClass, ["OSMID", "osmtags", "SHAPE@",
"waterway", "natural"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["WTRTYPE", "osm_id", "osmtags", "Shape@"])

for row in searchRows:
    if row[3] == 'stream' or row[3] == 'river':
        insertRows.insertRow(["Stream or River", float(row[0]), row[1], row[2]])
    if row[3] == 'canal' or row[3] == 'ditch':
        insertRows.insertRow(["Canal or Ditch", float(row[0]), row[1], row[2]])
    if row[3] == 'drain':
        insertRows.insertRow(["Estuary", float(row[0]), row[1], row[2]])
    if row[3] == 'weir' or row[3] == 'dam':
        insertRows.insertRow(["Dam or Weir", float(row[0]), row[1], row[2]])
    if row[4] == 'coastline':
        insertRows.insertRow(["Coastline", float(row[0]), row[1], row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1

```

```

del row

del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetFD = "WaterDistribution"
targetClass = "wMain"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0
whereClause = "man_made = 'pipeline'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourceLineClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass, ["osm_id", "osmtags",
"Shape@"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
del row

del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

#####*****##POINT FEATURES!*****#####
## ----- Target Classes to be populated (in the LGIM model)-----##
#           LGIM Class Name                               |           OSM
# (keywords/fields that will map here)
#           AddressEntrancePoint
#           AddressPoint
#           SiteAddressPoint
#
#           AssessAppealApplicants
#           TaxParcelAppeal
#           TaxParcelForeclosure
#           TaxParcelSale
#           TaxRevertedProperty
#

```

```

#           ControlCorner
#
#           CIPPoints
#           CIPProjectsLocations
#
#           ServiceRequests
#
#           CensusLandmarkPoint
#
#           EarlyVotingLocation
#           PollingPlace
#
#           SpotElevation
#
#           Accessoint
#           AssignmentBreak
#           AssignmentBreakLabel
#           DamageComBuilding
#           DamagePublicFacility
#           DamageResBuilding
#           OpsIncidentPoint
#           ResourceAssignment
#
#           BridgePoint
#           BuildingPhotoLoc
#           CurbRamp
#           ParkingPayStation
#           ParkingSpace | amenity
= 'parking'
#           PavementMarkingPoint
#           Pole
| power = 'pole'
#           RRCrossing
| highway = 'crossing'; railway = 'halt' OR 'crossing' OR 'level_crossing' OR 'turntable'
#           Sign
| highway = 'stop' OR 'motorway_junction'
#           Signal
| highway = 'traffic_signals'
#           SignalCabinet
#           StreetFurniture | highway
= 'bus_stop'

```

```

#           StreetIntersection | highway =
'mini_roundabout' OR 'turning_circle OR 'emergency_access_point'
#           Streetlight
#           TrafficCamera | highway
= 'speed_camera'
#           Tree
#
#           AED
#           FireIncident
#           FireResponseRestrictPoint
#           FireSafetySurvey
#
#           RoadBlockPoint
#           wHydrantInspection
#
#           CommunityEvent
#           DevelopmentSite
#           LuCodeViolation
#           LuPermit
#           LuWorkorder
#
#           Arrest
#           CallService
#           FieldInterview
#           Offense
#           Parolee
#           Probationer
#           SexOffender
#           Warrant
#
#           ConveyanceDivisionPoint
#           TaxParcelPoint
#
#           CommodityDistributionPoint
#           CommodityStagingPoint
#           EmergencyFacility
#           EmergencyShelter
#           MissionReadyPackage
#           NFPA704
#           PreIncidentPlanNote

```


PreIncidentPlantPhoto

PreIncidentPlanPoint

PublicSafetyResource

SecurityImprovement

SpecialNeedPopulation

VideoFeed

#

CommercialBusiness |
amenity = 'restaurant' OR 'food_court' OR 'fast_food' OR 'bbq' OR 'pub' OR 'bar' OR 'cafe' OR 'biergarten'
OR 'nightclub' OR 'stripclub' OR 'brothel' OR 'sauna' OR 'cinema' OR 'studio' OR 'theatre'

|
shop = 'alcohol' OR 'bakery' OR 'beverages' OR 'bicycle' OR 'books' OR 'butcher' OR 'car'
OR 'car_repair' OR 'clothes' OR 'computer' OR 'convenience' OR 'confectionery' OR 'department_store' OR
'dry_cleaning' OR 'electronics' OR 'fabrics' OR 'farm' OR 'florist' OR 'funeral_directors' OR 'furniture' OR
'garden_centre' OR 'general' OR 'gift' OR 'glazier' OR 'hairstylist' OR 'hardware' OR 'jewelry' OR 'laundry' OR 'mall'
OR 'massage' OR 'motorcycle' OR 'optician' OR 'second_hand' OR 'sports' OR 'stationery' OR 'supermarket' OR 'shoes'
OR 'toys' OR 'travel_agency' OR 'video'; tourism = 'camp_site' OR 'caravan_site' OR 'hostel' OR 'hotel' OR 'motel'

CommunityCenter

EducationFacility | amenity = 'kindergarten' OR 'school' OR 'college' OR 'university'

FacilitySitePoint (701 - Agriculture,
| amenity = 'ferry_terminal' OR 'bicycle_parking' OR 'bicycle_rental' OR 'bus_station' OR 'car_rental' OR
'car_sharing' OR 'car_wash' OR 'fuel' OR 'taxi'(810)

710 - Industry,
| amenity = 'atm' OR 'bank' (760)

720 - Commerical and Retail,
| amenity = 'architect_office' (720)

730 - Education,
| amenity = 'arts_centre'; historic = 'archaeological_site' OR 'battlefield' OR
'boundary_stone' OR 'castle' OR 'memorial' OR 'monument' OR 'ruins' OR 'wayside_cross' OR 'wayside_shrine' OR
'wreck'; leisure = 'nature_reserve'; tourism = 'alpine_hut' OR 'attraction' OR 'artwork' OR 'chalet' OR 'guest_house' OR
'picnic_site' OR 'theme_park' OR 'zoo' (820)

740 - Emergency Response and Law
Enforcement, | amenity = 'public_building' OR 'vending_machine' (790)

750 - Energy,
| man_made = 'lighthouse' OR 'beacon' (810)

760 - Banking and Finance,
| man_made = 'gasometer' OR 'mineshaft' OR 'petroleum_well' OR 'tower' OR 'windmill';
power = 'tower' OR 'station' OR 'sub_station' OR 'generator'(750)

780 - Mail and Shipping,
| man_made = 'watermill' OR 'watewater_plant' OR 'water_tower' OR 'water_works'
(850)

790 - Building General,
| tourism = 'information' (880)

800 - Health and Medical,
| military = 'airfield' OR 'bunker' OR 'barracks' OR 'danger_area' OR 'range' OR
'naval_base' (830)

810 - Transportation Facilities,

#		820 - Public Attractions and Landmark
Buildings,		
#		830 - Government and Military,
#		840 - Weather,
#		850 - Water Supply and Treatment,
#		880 - Information and Communication)
#	FireStation	amenity
= 'fire_station'		
#	GolfCourse	leisure =
'golf_course' OR 'miniature_golf'		
#	HazardousFacility	
#	HouseWorship	amenity =
'place_of_worship'		
#	Library	amenity
= 'library'		
#	MedicalFacility	amenity =
'pharmacy' (Feature Code = pharmacy); amenity = 'hospital' (Feature Code = hospital); amenity = 'dentist' (Feature Code = dentist); amenity = 'doctors' (Feature Code = doctor); amenity = 'veterinary' (Feature Code = veterinary)		
#	Museum	tourism
= 'museum'		
#	Park	leisure =
'dog_park' OR 'sports_centre' OR 'stadium' OR 'track' OR 'pitch' OR 'water_park' OR 'marina' OR 'fishing' OR 'park' OR 'playground' OR 'garden' OR 'ice_rink'		
#	PoliceStation	amenity = 'police'
#	PostOffice	amenity
= 'post_office' OR 'post_box'		
#	PublicWork	amenity
= 'courthouse' OR 'crematorium' OR 'embassy' OR 'emergency_phone' OR 'public_building' OR 'prison' OR 'grave_yard' OR 'fire_hydrant' OR 'shelter' OR 'telephone' OR 'toilets' OR 'waste_basket' OR 'townhall'		
#	RailroadStation	highway =
'platform'; railway = 'station' OR 'subway_entrance' OR 'tram_stop'		
#	RecyclingFacility	amenity = 'recycling' OR
'waste_disposal'		
#		
#	SewerStormwater_Net_Junctions	
#	ssBend	
#	ssCleanOut	
#	ssControlValve	
#	ssDischargePoint	
#	ssFitting	
#	ssInlet	
#	ssManhole	
#	ssNetworkStructure	
#	ssPump	

```

#           ssServiceConnection
#           ssSystemValve
#           ssTap
#           ssTestStation
#           ssValveOperator
#           ssVault
#
#           Stormwater_Net_Junctions
#           swCleanOut
#           swControlValve
#           swDischargePoint
#           swFitting
#           swInlet
#           swManhold
#           swNetworkStructure
#           swSystemValve
#           swWeirStructure
#
#           CurrentSnowPlowLocation
#           HistoricSnowPlowLocation
#
#           wAbandonedPoint
#           Water_Net_Junctions
#           wControlValve
#           wCurbStopValve
#           wElevationPt
#           wFitting
#           wHydrant
#           wNetworkStructure
#           wPump
#           wSamplingStation
#           wServiceConnection
#           wSystemValve
#           wTestStation
#
## -----##

```

```

targetFD = "FacilitiesStreets"
targetClass = "ParkingSpace"
print(targetClass)

```

```

arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "amenity = 'parking'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@"])
for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row
del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "Pole"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "power = 'pole'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@"])
for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row
del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "RRCrossing"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

```

```

i = 0

whereClause = "highway = 'crossing' OR railway = 'halt' OR railway = 'crossing' OR railway = 'level_crossing' OR
railway = 'turntable'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@", "TYPEXING"])

for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], row[2], "Public Vehicle"])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "Sign"
print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

i = 0

whereClause = "highway = 'stop' OR highway = 'motorway_junction'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@", "SIGNTYPE"])

for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], row[2], "Regulatory"])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "Signal"
print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

i = 0

whereClause = "highway = 'traffic_signals'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@"])

for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "StreetFurniture"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "highway = 'bus_stop'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@"])

for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "StreetIntersection"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "highway = 'mini_roundabout' OR highway = 'turning_circle OR highway = 'emergency_access_point'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@"])

```

```

for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

targetClass = "TrafficCamera"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
i = 0
whereClause = "highway = 'speed_camera'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\" + targetFD + "\\" + targetClass,
["osm_id", "osmtags", "SHAPE@", "CAMERATYP"])
for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2],"Speed Camera"])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

targetFD = "ReferenceData"

targetClass = "CommercialBusiness"
i=0
arcpy.TruncateTable_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
#arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

whereClause = "amenity = 'restaurant' OR amenity = 'food_court' OR amenity = 'fast_food' OR amenity = 'bbq' OR
amenity = 'pub' OR amenity = 'bar' OR amenity = 'cafe' OR amenity = 'biertgarten' OR amenity = 'nightclub' OR
amenity = 'stripclub' OR amenity = 'brothel' OR amenity = 'sauna' OR amenity = 'cinema' OR amenity = 'studio' OR
amenity = 'theatre' OR shop = 'alcohol' OR shop = 'bakery' OR shop = 'beverages' OR shop = 'bicycle' OR shop =
'books' OR shop = 'butcher' OR shop = 'car' OR shop = 'car_repair' OR shop = 'clothes' OR shop = 'computer' OR shop =
'convenience' OR shop = 'confectionery' OR shop = 'department_store' OR shop = 'dry_cleaning' OR shop =
'electronics' OR shop = 'fabrics' OR shop = 'farm' OR shop = 'florist' OR shop = 'funeral_directors' OR shop =
'furniture' OR shop = 'garden_centre' OR shop = 'general' OR shop = 'gift' OR shop = 'glazier' OR shop = 'hairstylist'
OR shop = 'hardware' OR shop = 'jewelry' OR shop = 'laundry' OR shop = 'mall' OR shop = 'massage' OR shop =
'motorcycle' OR shop = 'optician' OR shop = 'second_hand' OR shop = 'sports' OR shop = 'stationery' OR shop =

```

```

'supermarket' OR shop = 'shoes' OR shop = 'toys' OR shop = 'travel_agency' OR shop = 'video' OR tourism =
'camp_site' OR tourism = 'caravan_site' OR tourism = 'hostel' OR tourism = 'hotel' OR tourism = 'motel'

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "amenity", "shop", "tourism"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@", "LOCATIONTYPE"])

for row in searchRows:
## THE Target LOCATIONTYPE field is only 30 characters long, so trim out, if it is too long.
    locationType = str(row[3])+str(row[4])+str(row[5])
    locationType = locationType.split(';')[0]
    locationType = locationType[:30]
    print(locationType)
    insertRows.insertRow([float(row[0]),row[1],row[2],locationType])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "EducationFacility"
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
whereClause = "amenity = 'kindergarten' OR amenity = 'school' OR amenity = 'college' OR amenity = 'university'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "amenity"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "SHAPE@", "LOCATIONTYPE"])

for row in searchRows:
    insertRows.insertRow([float(row[0]),row[1],row[2],row[3]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows
arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "FacilitySitePoint"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

```


i = 0

```
whereClause = "amenity = 'ferry_terminal' OR amenity = 'bicycle_parking' OR amenity = 'bicycle_rental' OR amenity = 'bus_station' OR amenity = 'car_rental' OR amenity = 'car_sharing' OR amenity = 'car_wash' OR amenity = 'fuel' OR amenity = 'taxi' OR man_made = 'lighthouse' OR man_made = 'beacon' OR amenity = 'atm' OR amenity = 'bank' OR amenity = 'architect_office' OR amenity = 'arts_centre' OR historic = 'archaeological_site' OR historic = 'battlefield' OR historic = 'boundary_stone' OR historic = 'castle' OR historic = 'memorial' OR historic = 'monument' OR historic = 'ruins' OR historic = 'wayside_cross' OR historic = 'wayside_shrine' OR historic = 'wreck' OR leisure = 'nature_reserve' OR tourism = 'alpine_hut' OR tourism = 'attraction' OR tourism = 'artwork' OR tourism = 'chalet' OR tourism = 'guest_house' OR tourism = 'picnic_site' OR tourism = 'theme_park' OR tourism = 'zoo' OR amenity = 'public_building' OR amenity = 'vending_machine' OR man_made = 'gasometer' OR man_made = 'mineshaft' OR man_made = 'petroleum_well' OR man_made = 'tower' OR man_made = 'windmill' OR power = 'tower' OR power = 'station' OR power = 'sub_station' OR power = 'generator' OR man_made = 'watermill' OR man_made = 'wastewater_plant' OR man_made = 'water_tower' OR man_made = 'water_works' OR tourism = 'information' OR military = 'airfield' OR military = 'bunker' OR military = 'barracks' OR military = 'danger_area' OR military = 'range' OR military = 'naval_base'"
```

subtype810 =

```
['ferry_terminal', 'bicycle_parking', 'bicycle_rental', 'bus_station', 'car_rental', 'car_sharing', 'car_wash', 'fuel', 'taxi', 'lighthouse', 'beacon']
```

subtype760 = ['atm', 'bank']

subtype720 = ['architect_office']

subtype820 =

```
['arts_centre', 'archaeological_site', 'battlefield', 'boundary_stone', 'castle', 'memorial', 'monument', 'ruins', 'wayside_cross', 'wayside_shrine', 'wreck', 'nature_reserve', 'alpine_hut', 'attraction', 'artwork', 'chalet', 'guest_house', 'picnic_site', 'theme_park', 'zoo']
```

subtype790 = ['public_building', 'vending_machine']

subtype750 = ['gasometer', 'mineshaft', 'petroleum_well', 'tower', 'windmill', 'tower', 'station', 'sub_station', 'generator']

subtype850 = ['watermill', 'wastewater_plant', 'water_tower', 'water_works']

subtype880 = ['information']

subtype830 = ['airfield', 'bunker', 'barracks', 'danger_area', 'range', 'naval_base']

```
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags", "SHAPE@", "amenity", "man_made", "historic", "leisure", "tourism", "power", "military"], whereClause)
```

```
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["SUBTYPEFIELD", "osm_id", "osmtags", "SHAPE@", "LOCATIONTYPE"])
```

for row in searchRows:

if row[3] in subtype810 or row[4] in subtype810:

insertRows.insertRow([810, float(row[0]), row[1], row[2], str(row[3]) + str(row[4])])

if row[3] in subtype760:

insertRows.insertRow([760, float(row[0]), row[1], row[2], row[3]])

if row[3] in subtype720:

insertRows.insertRow([720, float(row[0]), row[1], row[2], row[3]])

if row[3] in subtype820 or row[5] in subtype820 or row[6] in subtype820 or row[7] in subtype820:

insertRows.insertRow([820, float(row[0]), row[1], row[2], str(row[3]) + str(row[5]) + str(row[6]) + str(row[7])])

if row[3] in subtype790:

insertRows.insertRow([790, float(row[0]), row[1], row[2], row[3]])

if row[4] in subtype750 or row[8] in subtype750:

```

        insertRows.insertRow([750, float(row[0]),row[1],row[2],str(row[4]) + str(row[8])])
    if row[4] in subtype850:
        insertRows.insertRow([850, float(row[0]),row[1],row[2],row[4]])
    if row[7] in subtype880:
        insertRows.insertRow([880, float(row[0]),row[1],row[2],row[7]])
    if row[9] in subtype830:
        insertRows.insertRow([830, float(row[0]),row[1],row[2],row[9]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetClass = "FireStation"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0
whereClause = "amenity = 'fire_station'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass, ["osm_id", "osmtags",
"Shape@"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], row[2]])
    print("row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetClass = "GolfCourse"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0
whereClause = "leisure = 'golf_course' OR leisure = 'miniature_golf'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "leisure"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@", "LOCATIONTYPE"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2], row[3]])

    print("row: " + str(i) + "osmid: " + row[0])

    i += 1

    del row


del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "HouseWorship"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "amenity = 'place_of_worship'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@"])
for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2]])

    print("row: " + str(i) + "osmid: " + row[0])

    i += 1

    del row


del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "Library"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "amenity = 'library'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2]])

    print("row: " + str(i) + "osmid: " + row[0])

    i += 1

    del row


del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "MedicalFacility"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0

whereClause = "amenity = 'pharmacy' OR amenity = 'hospital' OR amenity = 'dentist' OR amenity = 'doctors' OR
amenity = 'veterinary'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "amenity"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@", "FEATURECODE"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2], row[3]])

    print("row: " + str(i) + "osmid: " + row[0])

    i += 1

    del row


del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "Museum"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0

whereClause = "tourism = 'museum'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2]])

    print("row: " + str(i) + "osmid: " + row[0])

    i += 1

    del row


del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "Park"

print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

i = 0

whereClause = "leisure = 'dog_park' OR leisure = 'sports_centre' OR leisure = 'stadium' OR leisure = 'track' OR leisure
= 'pitch' OR leisure = 'water_park' OR leisure = 'marina' OR leisure = 'fishing' OR leisure = 'park' OR leisure =
'playground' OR leisure = 'garden' OR leisure = 'ice_rink'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "leisure"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["osm_id", "osmtags", "Shape@", "FEATURECODE"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2], row[3]])

    print("row: " + str(i) + "osmid: " + row[0])

    i += 1

    del row


del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "PoliceStation"

print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

i = 0

whereClause = "amenity = 'police'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@"], )

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2]])

    print("row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row


del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "PostOffice"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "amenity = 'post_office' OR amenity = 'post_box'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "amenity"], "amenity = 'post_office'")
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@", "FEATURECODE"])
for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2], row[3]])

    print("row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row


del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "PublicWork"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0
whereClause = "amenity = 'courthouse' OR amenity = 'crematorium' OR amenity = 'embassy' OR amenity =
'emergency_phone' OR amenity = 'public_building' OR amenity = 'prison' OR amenity = 'grave_yard' OR amenity =

```

```
'fire_hydrant' OR amenity = 'shelter' OR amenity = 'telephone' OR amenity = 'toilets' OR amenity = 'waste_basket' OR amenity = 'townhall'
```

```
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags", "SHAPE@", "amenity"], whereClause)
```

```
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags", "Shape@", "LOCATIONTYPE"])
```

```
for row in searchRows:
```

```
    insertRows.insertRow([float(row[0]), row[1], row[2], row[3]])
```

```
    print("row: " + str(i) + " osmid: " + row[0])
```

```
    i += 1
```

```
del row
```

```
del searchRows
```

```
del insertRows
```

```
arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
```

```
targetClass = "RailroadStation"
```

```
print(targetClass)
```

```
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
```

```
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
```

```
i = 0
```

```
whereClause = "highway = 'platform' OR railway = 'station' OR railway = 'subway_entrance' OR railway = 'tram_stop'"
```

```
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags", "SHAPE@", "highway", "railway"], whereClause)
```

```
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags", "Shape@", "LOCATIONTYPE"])
```

```
for row in searchRows:
```

```
    insertRows.insertRow([float(row[0]), row[1], row[2], str(row[3])+str(row[4])])
```

```
    print("row: " + str(i) + " osmid: " + row[0])
```

```
    i += 1
```

```
del row
```

```
del searchRows
```

```
del insertRows
```

```
arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
```

```
targetClass = "RecyclingFacility"
```

```
print(targetClass)
```

```
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
```

```
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
```

```
i = 0
```

```

whereClause = "amenity = 'recycling' OR amenity = 'waste_disposal'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePointClass, ["OSMID", "osmtags",
"SHAPE@", "amenity"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@", "LOCATIONTYPE"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2], row[3]])

    print("row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

#####POLYGON
FEATURES!#####

## ----- Target Classes to be populated (in the LGIM model)-----##

#           LGIM Class Name           |           OSM
(keywords/fields that will map here)

#           GeopoliticalArea
#           SchoolDistrict
#
#           LocalTaxDistrict
#           SchoolTaxDistrict
#           SpecialTaxDistrict
#           TaxParcelNeighborhood
#
#           PLSSQuarterSection
#           PLSSSection
#           PLSSTownship
#
#           CIPPolygons
#           CIPProjects
#           CIPProjectOverview
#
#           ParcelMarkup
#
#           CensusBlock
#           CensusBlockGroup
#           CensusTract
#

```


#	CountyDistrict
#	LocalDistrict
#	StateHouseDistrict
#	StateSenateDistrict
#	USHouseDistrict
#	USSenateDistrict
#	VotingPrecinct
#	
#	EvacuationArea
#	ExclusionHotZone
#	OpsIncidentArea
#	Plume
#	
#	PerformanceIndicator
#	
#	Building
#	BuildingFloor
#	BuildingFloorSection
#	BuildingInteriorSpace
#	LandscapeArea
#	PavementSchedule
#	StreetPavementType
#	
#	EMSDistrict
#	FireDistrict
#	FireMapIndex
#	TargetHazard
#	
#	CallVolumeDistrictReport
#	CallVolumeSnowMapReport
#	EngineeringGrid
#	piAlert
#	PlanDrawing
#	PlowActivityDistrictReport
#	PlowActivitySnowMapReport
#	PrintGrid
#	PublicWorksDistrict
#	
#	LandUseCase
#	

```

#           FEMAloodZone
#           IncentiveZone
#           LandUseCurrent
#           LandUseProposed
#           ZoningDistrict
#
#           LawDistrict
#           PSAPDistrict
#
#           Block
#           ConveyanceDivision
#           Encumbrance
#           OwnerParcel
#           SimultaneousConveyance
#           TaxMapIndex
#           TaxParcel
#
#           FloodImpactArea
#           InundationArea
#           PreIncidentComplex
#           PreIncidentPlan
#           PreIncidentPlanArea
#
#           BuildingFootprint
#                   | building = 'yes' OR 'apartments' OR 'block' OR 'brewery' OR 'detached' OR 'glasshouse'
OR 'hall' OR 'house' OR 'hut' OR 'pavilion' OR 'semi' OR 'shed'
#           FacilitySite(701 - Agriculture,
#                   | boundary = 'national_park' OR 'stadium'; geological = 'paleontological_site'; historic =
'archaeological_site' OR 'battlefield' OR 'ruins' (820)
#
#                   710 - Industry,
#                   | building = 'city_hall' (830)
#
#                   720 - Commerical and Retail,
#                   | building = 'barn'; landuse = 'farm' OR 'farmyard' (701)
#
#                   730 - Education,
#                   | building = 'factory' OR 'office' OR 'store' OR 'warehouse' OR
'golf_course'(720))
#
#                   740 - Emergency Response and Law
Enforcement,      | landuse = 'cemetery' OR 'military' (830)
#
#                   750 - Energy,
#                   | landuse = 'commercial' OR 'garages' OR 'orchard' OR 'quarry' OR
'retail' OR 'vineyard' OR 'greenhouse_horticulture' (720)
#
#                   760 - Banking and Finance,
#                   | landuse = 'industrial' OR 'landfill' (710)

```

```

#                                     780 - Mail and Shipping,
#                                     | tourism = 'camp_site' OR 'caravan_site' OR 'picnic_site' OR 'theme_park' OR 'zoo' (820)
#                                     790 - Building General,
#                                     |
#                                     800 - Health and Medical,
#                                     810 - Transportation Facilities,
#                                     820 - Public Attractions and Landmark
Buildings,
#                                     830 - Government and Military,
#                                     840 - Weather,
#                                     850 - Water Supply and Treatment,
#                                     880 - Information and Communication
#
# LandBase
#                                     | landuse = 'allotments' OR 'village_green' OR 'residential'
#
# Landform
#                                     | natural = 'beach' OR 'fell' OR 'glacier' OR 'wetland'
#
# Soil
#                                     | natural = 'heath' OR 'land' OR 'marsh' OR 'mud' OR 'scree'
OR 'scrub'
#
# Vegetation
#                                     | landuse = 'forest' OR 'grass' OR 'greenfield' OR 'meadow'
OR 'brownfield' OR 'wood'; natural= 'wood'
#
# Waterbody
#                                     | landuse = 'reservoir' OR 'basin' OR 'salt_pond'; natural =
'bay'
#
#
# ssDetention
#
#
# swDetention
#
#
# WaterPressureZone
#
# WaterUtilityDistrict
#
# wStructure
## -----##

```

```

targetClass = "BuildingFootprint"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0

whereClause = "building = 'yes' OR building = 'apartments' OR building = 'block' OR building = 'brewery' OR building = 'detached' OR building = 'glasshouse' OR building = 'hall' OR building = 'house' OR building = 'hut' OR building = 'pavilion' OR building = 'semi' OR building = 'shed'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourcePolygonClass, ["OSMID", "osmtags", "SHAPE@", "OBJECTID", "building"], whereClause)

```

```

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "FEATURECODE"])

for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], row[4]])
    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)

targetClass = "FacilitySite"
#5 subtypes to load here, so 5 separate where clauses, with 5 separate sets of cursors

#subtype 820
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass)
i = 0
whereClause = "boundary = 'national_park' OR boundary = 'stadium' OR geological = 'paleontological_site' OR
historic = 'archaeological_site' OR historic = 'battlefield' OR historic = 'ruins' OR tourism = 'camp_site' OR tourism =
'caravan_site' OR tourism = 'picnic_site' OR tourism = 'theme_park' OR tourism = 'zoo'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "SUBTYPEFIELD"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], 820])
    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

#Subtype 701
whereClause = "building = 'barn' OR landuse = 'farm' OR landuse = 'farmyard'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "SUBTYPEFIELD"])

```

```

for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], 701])
    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

#Subtype 710
whereClause = "landuse = 'industrial' OR landuse = 'landfill'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "SUBTYPEFIELD"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], 710])
    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

#Subtype 720
whereClause = "building = 'factory' OR building = 'office' OR building = 'store' OR building = 'warehouse' OR building
= 'golf_course' OR landuse = 'commercial' OR landuse = 'garages' OR landuse = 'orchard' OR landuse = 'quarry' OR
landuse = 'retail' OR landuse = 'vineyard' OR landuse = 'greenhouse_horticulture'"
searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\ " + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID"], whereClause)
insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\ " + targetFD + "\\ " + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "SUBTYPEFIELD"])
for row in searchRows:
    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], 720])
    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])
    i += 1
    del row

del searchRows
del insertRows

#Subtype 830

```

```

whereClause = "landuse = 'cemetery' OR landuse = 'military' OR building = 'city_hall'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\" + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\" + targetFD + "\\" + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "SUBTYPEFIELD"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], 830])

    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

targetClass = "LandBase"
print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

i = 0

whereClause = "landuse = 'allotments' OR landuse = 'village_green' OR landuse = 'residential'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\" + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID", "landuse"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\" + targetFD + "\\" + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "MUNITYPE"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], row[4]])

    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

targetClass = "Landform"
print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\" + targetFD + "\\" + targetClass)

i = 0

whereClause = "natural = 'beach' OR natural = 'fell' OR natural = 'glacier' OR natural = 'wetland'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID", "natural"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "LNDFORMTYP"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], row[4]])

    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row


del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "Soil"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0

whereClause = "natural = 'heath' OR natural = 'land' OR natural = 'marsh' OR natural = 'mud' OR natural = 'scree' OR
natural = 'scrub'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID", "natural"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@", "SOILTYPE"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2], row[4]])

    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row


del searchRows
del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)


targetClass = "Vegetation"
print(targetClass)
arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)
i = 0

whereClause = "landuse = 'forest' OR landuse = 'grass' OR landuse = 'greenfield' OR landuse = 'meadow' OR landuse
= 'brownfield' OR landuse = 'wood' OR natural= 'wood'"

```

```

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID", "landuse", "natural"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass, ["osm_id", "osmtags",
"Shape@", "TYPE", "DESCRIPTION"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], row[2], row[4], row[5]])

    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

targetClass = "Waterbody"

print(targetClass)

arcpy.TruncateTable_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

arcpy.RemoveSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

i = 0

whereClause = "landuse = 'reservoir' OR landuse = 'basin' OR landuse = 'salt_pond' OR natural = 'bay'"

searchRows = arcpy.da.SearchCursor(sourceWorkspace + "\\\" + sourcePolygonClass, ["OSMID", "osmtags",
"SHAPE@", "OBJECTID"], whereClause)

insertRows = arcpy.da.InsertCursor(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass,
["BUILDINGID", "osmtags", "osm_id", "Shape@"])

for row in searchRows:

    insertRows.insertRow([float(row[0]), row[1], float(row[0]), row[2]])

    print("objectid: " + str(row[3]) + ":: row: " + str(i) + " osmid: " + row[0])

    i += 1

    del row

del searchRows

del insertRows

arcpy.AddSpatialIndex_management(targetWorkspace + "\\\" + targetFD + "\\\" + targetClass)

```


Appendix B. Python Script to Download and Extract OSM File

```
#-----  
# MIP Data Extraction script.  
# Author: David Crawford  
#  
# Created: 26/10/2016  
# Description: This is an application which connects to a known URL where we can  
# download the North America OSM dataset from. And then use some core  
# python libraries to extract the downloaded .bz2 file.  
#  
# Prerequisites:  
# 1. The user has the bz2 python library installed  
# 2. The user must have ArcGIS Desktop 10.7  
# installed on their machine  
# 3. The user must have the osmtools installed on  
# their local machine  
#  
# Recommendations:  
# 1. It is highly recommended that you take the  
# ArcGIS license offline (if using a license manager)  
# this will allow the script to process  
# for several days without losing its license  
# 2. It is strongly encouraged to use the 64-bit  
# background geoprocessing arcpy module to execute this script (it will cut runtime down  
# by a significant time)  
#  
# This script will take several days to execute.  
# The download portion is completely dependent on the  
# available bandwidth and internet connectivity  
# The extract process will be using the osm tools installed  
# with ArcGIS Desktop 10.7, and at that point performance  
# becomes dependent on whether or not you leverage  
# 64-bit Geoprocessing.  
#-----  
import arcpy  
import urllib2  
import sys  
import os  
import bz2  
from bz2 import decompress
```

```

northAmericaUrl = "https://download.geofabrik.de/north-america-latest.osm.bz2"
#EuropeUrl = "https://download.geofabrik.de/europe-latest.osm.bz2"
#AfricaUrl = "https://download.geofabrik.de/africa-latest.osm.bz2"
#AsiaUrl = "https://download.geofabrik.de/asia-latest.osm.bz2"
#AustraliaUrl = "https://download.geofabrik.de/australia-oceania-latest.osm.bz2"
#CentralAmericaUrl = "https://download.geofabrik.de/central-america-latest.osm.bz2"
#SouthAmericaUrl = "https://download.geofabrik.de/south-america-latest.osm.bz2"

```

```

#ADD additional variable names in here, for the other files you are choosing to
download.

```

```

urlSet = [northAmericaUrl] #,EuropeUrl,SouthAmericaUrl, ...]

```

```

outputFiles = []

```

```

for url in urlSet:

```

```

    file_name = url.split('/')[-1]

```

```

    u = urllib2.urlopen(url)

```

```

    f = open(file_name, 'wb')

```

```

    meta = u.info()

```

```

    file_size = int(meta.getheaders("Content-Length")[0])

```

```

    print "Downloading: %s Bytes: %s" % (file_name, file_size)

```

```

    file_size_dl = 0

```

```

    block_sz = 8192

```

```

    while True:

```

```

        buffer = u.read(block_sz)

```

```

        if not buffer:

```

```

            break

```

```

        file_size_dl += len(buffer)

```

```

        f.write(buffer)

```

```

        status = r"%10d  [%3.2f%%]" % (file_size_dl, file_size_dl * 100. /

```

```

file_size)

```

```

        status = status + chr(8)*(len(status)+1)

```

```

        print status,

```

```

        dirpath = r"C:\users\dave6199\Downloads"

```

```

        filepath = os.path.join(dirpath,file_name)

```

```

        newfile = bz2.decompress(filepath)

```

```

        newfilepath = os.path.join(dirpath,newfile)

```

```

    f.close()

```

```

    outputFiles.append(newfilepath)

```

```

##-----

```

```

#                NOW Load the .osm files into point/line and polygons using the

```

```

OSM toolbox

```

```

#

```

```
#      This target Geodatabase becomes the 'source geodatabase' in the data loading
script.
#      make appropriate edits, so that the script loads data where appropriate.
#      North America      - ~350GB
#
#      The sample output provided includes these 3 test continents (North America,
Europe and Antarctica) and is 948GB (extracted from osm dated 11/01/2020)
##-----
```

for file in outputFiles:

```
    #Set the Target Geodatabase - where will the extracted raw data go to.
    targetGeodatabase =
r"C:\VikingGeographicDataLocation\OSMExtractedData.gdb"
    #The name of the Feature Dataset (hard coded for North America) - you are in a
for loop, so could rename using the file name is a clever way
    targetFeatureDatasetName = "NorthAmerica"
    #The name of the Target Feature Classes (hard coded for North America) - you
are in a for loop, so could rename using the file name is a clever way
    targetPointClassName = "NorthAmerica_osm_pt"
    targetLineClassName = "NorthAmerica_osm_ln"
    targetPolygonClassName = "NorthAmerica_osm_ply"

    arcpy.OSMGPFileLoader(newfilepath,
"DO_NOT_CONSERVE_MEMORY","ALL",targetGeodatabase + "\\\" +
targetFeatureDatasetName, targetGeodatabase + "\\\" + targetFeatureDatasetName + "\\\" +
targetPointClassName, targetGeodatabase + "\\\" + targetFeatureDatasetName + "\\\" +
targetLineClassName, targetGeodatabase + "\\\" + targetFeatureDatasetName + "\\\" +
targetPolygonClassNameClassName)

print("Download and Extraction Complete -> Ready to run the Data Loading script to
move into the LGIM")
```