Python Scripting in the ArcGIS Environment:

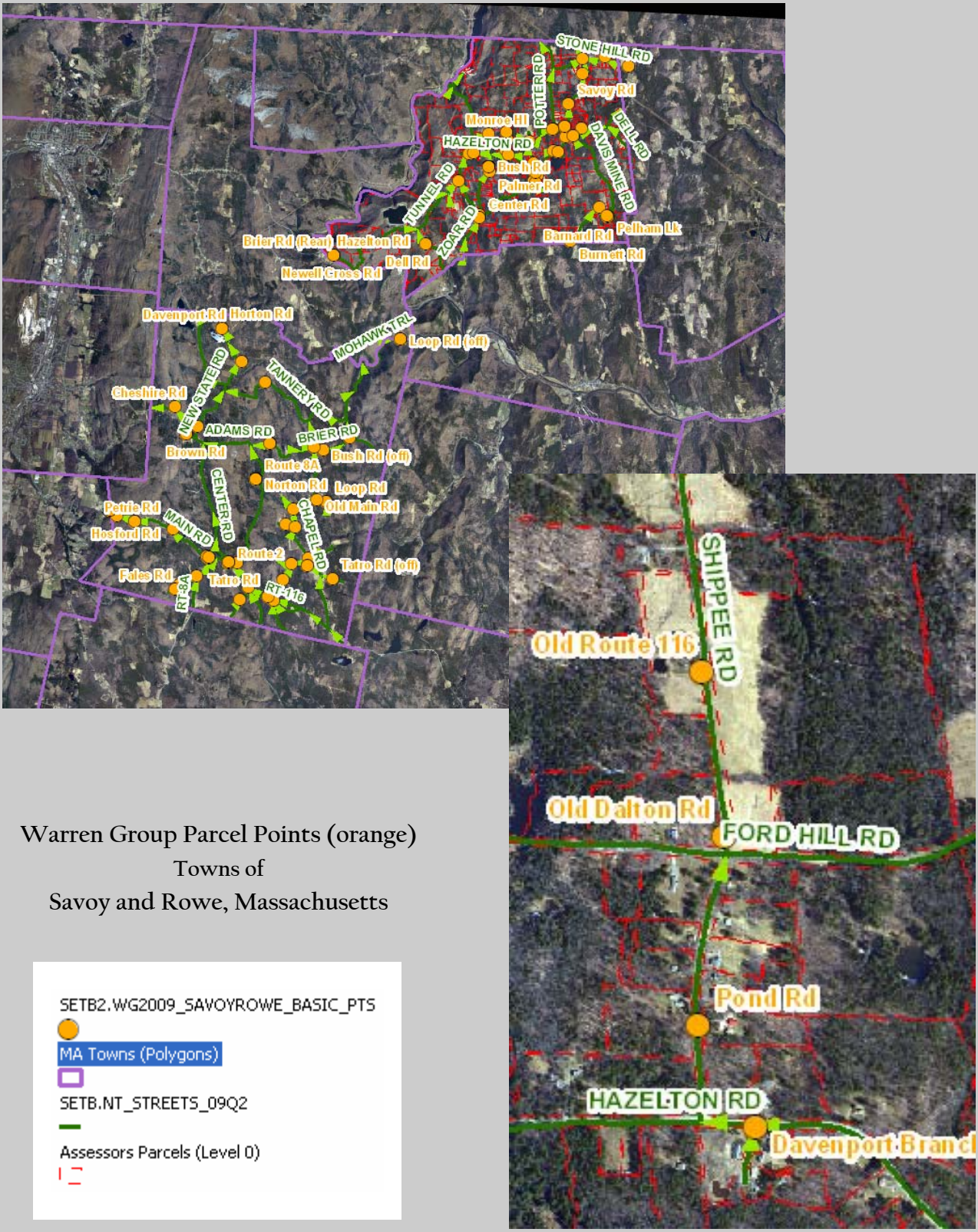
Creating a Point Feature Class on SDE

ITC 7976 Directed Study Fall 2009
Final Project Report
December 2009

Graduate Student: Claire Palmer
Email: clairempalmer@gmail.com

Supervising Instructor: Dr. Glenn Hazelton
Email: g.hazelton@neu.edu

Python for ArcGIS and Open Source GIS
MPS-GIT ITC 7976 Directed Study Fall 2009
Northeastern University
College of Professional Studies



Warren Group Parcel Points (orange)
Towns of
Savoy and Rowe, Massachusetts

SETB2.WG2009_SAVOYROWE_BASIC_PTS

MA Towns (Polygons)

SETB.NT_STREETS_09Q2

Assessors Parcels (Level 0)

Figure 1. WG2009_SAVOYROWE_BASIC_PTS Overlaid With NAVTEQ streets linework

Table of Contents

Python Scripting in the ArcGIS Environment: Creating a Point Feature Class on SDE

Introduction

This report describes the ArcGIS tools and methods used in a Python script to create a point feature class from a table of LAT/LON values inside an SDE geodatabase. This feature class could serve as an additional resource dataset for the 911 Project at MassGIS to perform QA/QC on streets listed in the Master Street Address Guide table (MSAG) that are missing from the NAVTEQ streets dataset. Typically, GIS professionals generate points from a table of LAT/LON values by selecting the "Display XY data" option from inside ArcMap. Although this method is quick and simple, it offers no flexibility for selecting subsets of rows or columns for the output. This becomes problematic when dealing with large datasets with many fields; for example, the Warren Group 2009 table of parcel names and transactions for Massachusetts used for this project contains 2,752,102 records and 113 fields. Utilizing a point dataset of 2 million records for QA/QC is possible, but hardly preferable to a dataset comprised of only the pertinent columns and rows (no nulls or duplicates) for the task at hand. A Python script, by comparison, is fully customizable to end-user requirements and can be crafted to perform the same geoprocessing routines executed by ArcGIS.

Data Tables and Pre-processing

In addition to creating a working Python script that produces a functional feature class, an objective for this final project was to craft the script to work in an SDE environment. Nearly all of the ArcGIS python script examples provided by ESRI reference shapefiles for source and output datasets. However, because MassGIS stores many of its official datasets on SDE, it made sense to continue to work in the SDE environment and to explore any scripting differences (syntactical or otherwise) that may arise between SDE data and stand-alone shapefiles.

The script processes a single data table – a statewide listing of parcel attributes produced by Warren Group current through 2009 (*SETB2.WG2009_STATE_ATTRIB*). The table lists every parcel address in Massachusetts, including LAT/LON values in decimal degrees to

locate the parcel geographically. There are a significant number of parcel records with "0.0" LAT/LON values, which typically indicates that the parcel is located on a newly constructed street. To serve the needs of the 911 Project, I wanted to generate only unique parcel ("street") names for each town and to create a point for each street based on its LAT/LON value. The remaining streets with "0.0" LAT/LON values would be exported into a separate table as a QA/QC resource for "new" streets.

For testing purposes, I created a subset of the *SETB2.WG2009_STATE_ATTRIB* table that included only the towns of Savoy and Rowe in Western Massachusetts. This table, *SETB2.WG2009_SAVOYROWE*, contained 1772 records. [See Figure 2.]

Figure 2. SETB2.WG2009_SAVOYROWE with Values

OBJECTID *	PROPID	MAPREF	SOURCE	INACTVFL	CITY	STREET	STNUM	LAT	LON	STATE	OWNER1	FLAG
1400	4281401	M:009 B:004 L:20	P		Savoy	Brier Rd	78	42.60501	-72.98501	MA	Irene=Papp	
1376	2736537	M:009 B:001 L:44	P		Savoy	Brier Rd	13	42.60553	-72.99226	MA	Allen=Carlow	
1587	2736559	M:009 B:001 L:14	P		Savoy	Brier Rd	127	42.60657	-72.98229	MA	Peter=Frank	
910	3971193	M:009 B:001 L:89	P		Savoy	Brier Rd	67	42.60505	-72.9885	MA	Harold=Kemp	
1540	2736157	M:009 B:001 L:35	P		Savoy	Brier Rd	53	42.60504	-72.98785	MA	Aaron=Gazaille	
1542	2736167	M:009 B:004 L:22	P		Savoy	Brier Rd	142	42.6068	-72.98159	MA	Steven=Serre	
1607	4281369	M:009 B:001 L:25	P		Savoy	Brier Rd	0	0	0	MA	Craig=Ruebesam	
1546	2736207	M:009 B:004 L:21	P		Savoy	Brier Rd	130	42.6068	-72.98159	MA	Brian K=Carlow	
300	4281365	M:009 B:001 L:17	P		Savoy	Brier Rd (Rear)	0	0	0	MA	Aaron=Gazaille	
297	4281337	M:007 B:002 L:59	P		Savoy	Brown Rd	0	0	0	MA	Edward C=Marko	
295	4281325	M:006 B:002 L:12	P		Savoy	Brown Rd	0	0	0	MA	Steven=Blazejewski	
58	4281327	M:006 B:002 L:66	P		Savoy	Brown Rd	0	0	0	MA	Frances=Blazejewski	
708	3248101		S	T	Savoy	Brown Rd	0	0	0	MA	Allen D=Haskins	
517	4281338	M:007 B:002 L:6	P		Savoy	Brown Rd	0	0	0	MA	Ralph=Lambert	
1392	4281339	M:007 B:002 L:60	P		Savoy	Brown Rd	0	0	0	MA	Allen D=Haskins	
1156	4281342	M:007 B:002 L:79	P		Savoy	Brown Rd (rear)	0	0	0	MA	Joseph J=Allbozek=Jr	
518	4281341	M:007 B:002 L:73	P		Savoy	Brown Rd (rear)	0	0	0	MA	Carl=Lambert	

As soon as I was able to get the script to produce a point feature class from the above table, I discovered that the feature class failed to draw within the spatial extent for Massachusetts in ArcMap. Troubleshooting this projection issue is further explained in Debugging section of this report. As a workaround to the problem, I was pointed to a nearly identical dataset on SDE that contains X/Y coordinate values for NAD83 Massachusetts State Plane Meters, *SETB2.WG2009_STATE_BASIC*. A subset of this table for Savoy and Rowe was created for testing purposes. [See Figure 3.]

Figure 3. SETB2.WG2009_SAVOYROWE_BASIC with Values

OBJECTID *	PROPID	MAPREF09	CITY	STREET	STNUM	X	Y	SOURCE	INACTVFL	OWNER	ZIPCODE	FLAG
579	4002395	M:0406 B:0000 L:0003	Rowe	Bear Swamp Rd	1	83706.2761094	939513.197242	P		Bear Swamp Power Co LLC	01367	1
184	4733868	M:0407 B:0000 L:0045	Rowe	Brittingham Hill Rd	0	83340.364889	937645.559287	P		George F Veber	01367	
1578	4002235	M:0407 B:0000 L:0043	Rowe	Brittingham Hill Rd	37	83127.649536	938175.81346	P		Judith Pierce	01367	
1610	4002387	M:0407 B:0000 L:0009	Rowe	Brittingham Hill Rd	72	82815.540874	938617.693578	P		John F Rossi	01367	
1503	4002230	M:0407 B:0000 L:0040	Rowe	Brittingham Hill Rd	93	82708.819326	938919.126855	P		Kenneth Fensky	01367	
794	4002366	M:0407 B:0000 L:0008	Rowe	Brittingham Hill Rd	54	82942.482927	938363.219335	P		John F Rossi	01367	
1753	4002241	M:0407 B:0000 L:0044	Rowe	Brittingham Hill Rd	0	83340.364889	937645.559287	P		M Arlene Andognini	01367	
1416	4002231	M:0407 B:0000 L:0041	Rowe	Brittingham Hill Rd	81	82775.692472	938744.341383	P		Barbara F Davisonson RET	01367	
104	4002229	M:0407 B:0000 L:0039	Rowe	Brittingham Hill Rd	99	82638.050001	938981.238803	P		James E Sousa	01367	1
688	4002390	M:0407 B:0000 L:0006	Rowe	Brittingham Hill Rd	14	83211.819899	937824.886086	P		Warren Kalaus	01367	
1154	4002234	M:0407 B:0000 L:0046	Rowe	Brittingham Hill Rd	0	83340.364889	937645.559287	P		Barbara F Davisonson RET	01367	
1032	4002367	M:0407 B:0000 L:0007	Rowe	Brittingham Hill Rd	24	83175.404325	937977.667608	P		Donald A Rice	01367	
1247	4002058	M:0404 B:0000 L:0029	Rowe	Brown Rd	36	83723.719232	940751.489804	P		Highlands RT	01367	
461	4002078	M:0202 B:0000 L:0052	Rowe	Brown Rd	20	83733.405842	940674.525791	P		Highlands RT 2	01367	1
809	3117460	M:0404 B:0000 L:0030	Rowe	Brown Rd	0	83743.30825	940587.716375	P		Highlands RT	01367	
981	3117427	M:0201 B:0000 L:0037	Rowe	Corner Pond Rd	0	83706.2761094	939513.197242	P		Rowe Town Of	01367	1
1531	3117403		Rowe	County Branch Rd	12	83749.681775	939335.380405	S	T	Robert J Clancy	01367	1
566	4001880	M:0202 B:0000 L:0035	Rowe	County Rd	143	83748.089823	939603.353344	P		Jean M Bernhardt	01367	
575	4002232	M:0407 B:0000 L:0042	Rowe	County Rd	0	83390.328019	937779.331905	P		Barbara F Davisonson RET	01367	
1755	4002254	M:0407 B:0000 L:0051	Rowe	County Rd	0	83390.328019	937779.331905	P		Jack Hayden	01367	

As a means for flagging records to be processed into a point feature class, the column "FLAG" was added to the test input tables. No other pre-preprocessing was necessary.

Code Structure – Arcgisscripting Tools and Methods

The bulk of this project was the process of piecing together the Arcgisscripting methods and tools that I had been exposed to in the class assignments and from various tasks at work. Eventually, I was able to increasingly absorb the scripting examples posted on the ESRI on-line forums, resource manuals and ArcGIS Desktop Help.

The order of processes that produce a functional point dataset and an error-free script are outlined as follows:

Part 1

1. Run SQL to update FLAG column & create output table of new streets.

Part 2

2. Create an empty feature class.
3. Add fields to the empty feature class.

Part 3

4. Open SearchCursor on input table.
5. Open InsertCursor on output feature class.
6. Loop through the input table:
    a. create a point object from LAT/LON values
    b. assign the values from a unique ID field to a field in the output feature class
    c. assign the point object and unique ID to a row object
    d. insert the row object into the output feature class

Part 4

7. Run SQL to copy the attributes from the input table to the output feature class, based on the unique ID inserted in the previous step.

The status messages from a successful run of the script printed to the PythonWin Interactive window are shown in Figure 4.

Figure 4. PythonWin Interactive Window Messages From a Successful Run

A list of *all* the attempted methods and tools in the effort to achieve a working script paints a different picture – one that sheds more light on the trial and error nature of the programming process:

Part 1

~~1. Create MakeQueryTable on WG.WG2009_STATE_ATTRIB containing DISTINCT parcel names (STREET)~~

~~2. Copy the query view to a new table~~

~~3. Create the final DIST and DIST_NEW Table Views from the above Table~~

~~a. First set up a fieldInfo object to exclude all the fields we don't want to copy when creating the table view~~

~~b. Next create a table view for the WG2009 FC with only the desired fields visible~~

~~c. Finally, use the table view to create the subset outputs. These will have only the INCLUDE_FIELDS fields.~~

4. Run SQL to update FLAG column & create output table of new streets.

Part 2

~~5. Create a spatial reference object~~

~~6. Create an empty feature class that includes the spatial reference variable~~

7. Create an empty feature class.

~~8. Open SearchCursor & InsertCursor to loop through and insert point object into output feature class~~

9. Add fields to the empty feature class.

Part 3

10. Open SearchCursor on input table.
11. Open InsertCursor on output feature class.
12. Loop through the input table:
    a. create a point object from LAT/LON values
    ~~b. assign the values from a unique ID field to the ID attribute of the point created in the previous step~~
    ~~c. assign all the values from the input table to fields in the output feature class~~
    d. assign the point object and unique ID to a row object
    e. insert the row object into the output feature class

Part 4

~~13. Open an UpdateCursor to update ID field in the output feature class based on point ID that was inserted with the point~~

~~14. Join the point ID in the output feature class to the ID of the input table to copy over attributes~~

15. Run SQL to copy the attributes from the input table to the output feature class, based on the unique ID inserted in the previous step.

The factors that contributed the most to my progress with the script were the error messages I received from the `except` blocks and dividing the script into sections and sub-tasks for each section. A typical scripting cycle thus included:

1) Research why the script won't work
2) Borrow examples from similar questions posted on forums
3) Learn more about the nature of my bug and my approach
4) Discover another tool/method to use that I hadn't thought of before
5) Implement the new idea
6) Research why the script won't work

Some very helpful resources found during this trial-and-error process were:

- The "Differences Between Geoprocessor Versions" ESRI article at: http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Differences_between_geoprocessor_versions
- Pages 4-7 of the "Python Scripting – Advanced Techniques" PDF where I learned about the ArcSDESQLExecute object: http://www.tucsonaz.gov/gis/downloads/tw_166.pdf
- "Golden Rules for Debugging Python" at: http://www.ollivier.co.nz/support/python.shtm
- "Tips and tricks - Error handling in Python script tools": http://blogs.esri.com/Dev/blogs/geoprocessing/archive/2008/12/01/Tips-and-tricks-_2D00_-Error-handling-in-Python-script-tools.aspx

The "ArcSDESQLExecute" object was particularly useful for getting me over the hurdles I was struggling with in parts 1 & 4 of the code by allowing me to send an SQL statement directly to Oracle on SDE to process the data. This method executed much faster than instantiating cursors and rows, bypassing the need for the "MakeQueryTable" and "UpdateCursor" tools that didn't seem to accomplish what I needed to do. Having the ability to utilize the ArcSDESQLExecute object was a big advantage to working in an SDE environment.

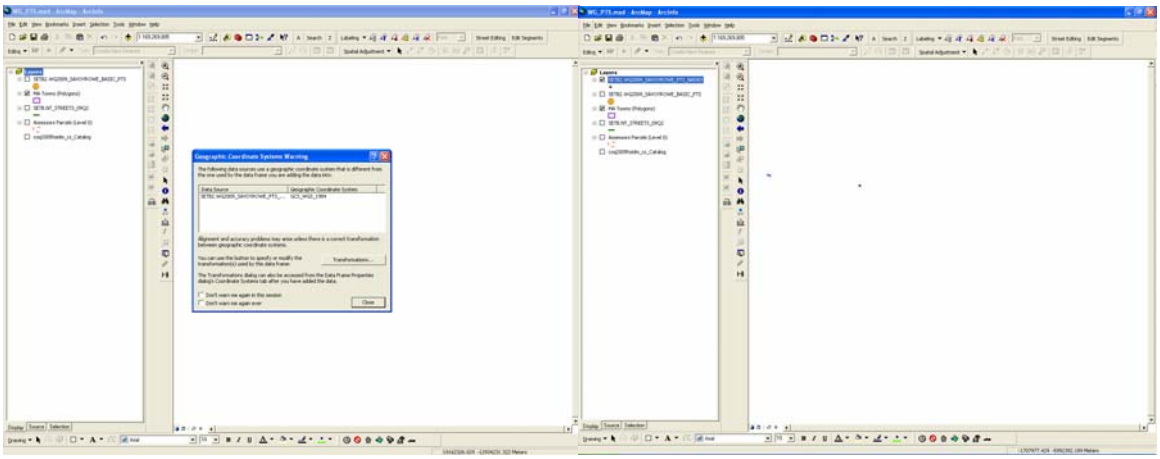
Another useful technique was the implementation of a multi-dimensional array ("matrix" in Python lingo) in part 2 of the script. This functioned as a container for the field names and settings to be added to the newly created point feature class. By wrapping each of the matrix elements into a `For` loop, I was able to write three lines of code to execute the "AddField" tool for each item in the matrix. This saved me from having to repeat the "AddField" code for each individual field.

Script Debugging

The articles on debugging mentioned in the previous section were a huge help in optimizing the error handling in my script. I was able to parse out the geoprocessing errors from the python/system-specific errors and include the ever-important line number and error message where the break occurred. Before adding the `try/except` blocks, the script occasionally crashed PythonWin without breaking out of the process or returning an error. Initializing my cursor and row objects to `None` solved the problem.

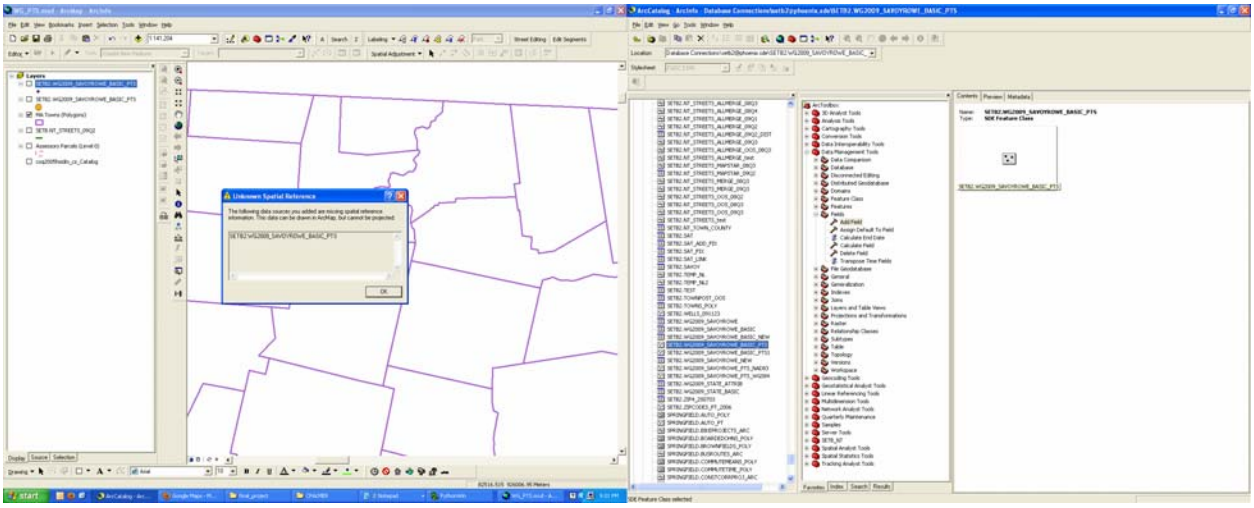
Setting the spatial reference for the point feature class was another challenge I encountered. The feature class created from the *SETB2.WG2009_SAVOYROWE* table containing LAT/LON coordinates in decimal degrees failed to draw within the spatial extent for Massachusetts. [See Figure 5.] Instead, it would project south into Rhode Island or east into Bermuda. Multiple attempts to *define the projection*, then *reproject* the feature class using the ArcToolbox projection tools failed to set the feature class's spatial reference to "NAD 1983 StatePlane Massachusetts Mainland FIPS 2001".

Figure 5. WG2009_SAVOYROWE_PTS Fails to Project Into Mass State Plane Meters



As described in the Data Tables and Pre-processing section of this report, this problem was solved by using an alternative table, *SETB2.WG2009_SAVOYROWE_BASIC*, which contains X/Y values pre-converted into NAD 1983 StatePlane Massachusetts Mainland FIPS 2001 coordinates. The resulting point feature class draws in ArcMap in the correct location, though ArcMap returns a warning that the feature class is missing spatial reference information. [See Figure 6.]

Figure 6. WG2009_SAVOYROWE_BASIC_PTS Projects Correctly Even Though It Lacks Spatial Reference Information



Although I got the results I wanted by using the alternate table, I was not satisfied that I hadn't solved the problem in the script itself. All of the ESRI examples for the CreateFeatureClass tool that I had researched specified a parameter for a *template feature class* on which to base the schema (spatial reference and field settings) for the new feature class. In my case, there were no existing feature classes on SDE upon which to model the schema for the new feature class I wanted. My attempts to code-in a separate spatial reference object and pass it in as a parameter for the new feature class were unsuccessful. Likewise, my Google search to "Create Feature Class without template" did not return any information. I sent an email to Jeff Bigos at ESRI for advice on how to approach the problem but did not receive a reply. As I am astounded that this fairly common scenario has not been previously addressed by anyone in the cyberspere, I intend to pursue a solution on the ESRI forums in the future.

Summary of Results

The point feature class, *SETB2.WG2009_SAVOYROWE_BASIC_PTS*, contains a set of 63 records – a point for each distinct street name. Although the points plotted on street centerlines, the value for STREET_NAME in the point feature class did not match that of the NAVTEQ streets linework. [See Figure 1 at the beginning of this report.] This discrepancy is

attributed to the non-reliable nature of Warren Group LAT/LON values. In relation to the scope of this final project, the incorrect point data was of no concern.

Once the script succeeded in producing a functional feature class from the test tables, it was run on the full version table containing records for the entire state (2,752,102 records). The script was run in the evening and was still in process the next morning. It was discovered that the script hadn't finished due to a time-out error to the database connection. The script was killed, and inspection of the data table showed that the FLAG column was updated to "1" where appropriate, but no new feature class on SDE had been created. Further investigation into how to optimize the script for large datasets would be helpful for future versions.

Future Code Improvements

In addition to coding a solution to set a spatial reference for the feature class and to optimize the script to handle large datasets, the script would achieve improved functionality if it was added as a tool in ArcToolbox. At minimum, the code would need to accept user parameters for the input table and output feature class name, spatial reference, and field settings. Because the current script fails if the input table does not contain a FLAG field, a future version could check for the existence of a FLAG field, and, if not, add it to the input table.

I found the ESRI Arcgisscripting forum to be very helpful for finding bits of code for enhancing the script. One example is the GetCount tool that returns the total records in a dataset: http://forums.esri.com/Thread.asp?c=93&f=1729&t=292293&mc=6 . After a handful of visits to the ESRI forum, I was extremely satisfied to find that my fluency with the language was improving. Perusing the forum was a great learning experience.

Overall, I am very pleased with the results of this scripting project. While there is still much to learn, I feel that I have broken the "newbie" barrier of programming. Having navigated the "unpredictable weather" of the full programming cycle – prototyping, implementing, debugging, and research – I feel I have gained my sea legs for future projects.

Appendix A.
Generate Warren Group Points
gen_WG_PTS.py

```
##--------------------------------------------------------------------------------
## Tool Name:  Generate WG points
## Source Name: gen_WG_PTS.py
## ArcGIS Version: ArcGIS 9.3
## Author: Claire Palmer (MassGIS)
## Last Updated: 12/08/2009
## By: Claire Palmer (MassGIS)
##
## Required Argumuments:
## - SDE.SETB Warren Group data table:  WG.WG2009_STATE_ATTRIB
## - for testing:  SETB2.WG2009_SAVOYROWE
##
## Optional Arguments:
## - None
##
## Description:
## Creates Point Feature Class of unique parcel names from the
## WG.WG2009_STATE_ATTRIB feature class on SDE
##
## Purpose:
## For use as a reference dataset for locating street names listed in the MSAG
## that are not included in the NAVTEQ streets dataset
##
##--------------------------------------------------------------------------------
# Import system modules
import sys, string, os, arcgisscripting
import ArcGIS_AddMsgAndPrint
AddMsgAndPrint = ArcGIS_AddMsgAndPrint.AddMsgAndPrint

# Create the Geoprocessor object
gp = arcgisscripting.create(9.3)

# traceback variables for exception handling
def trace():
   import traceback
   tb = sys.exc_info()[2]                    # get the traceback object
   tbinfo = traceback.format_tb(tb)[0]       # tbinfo contains the error's line number and the code
   line = tbinfo.split(", ")[1]
   filename = sys.path[0] + os.sep + "gen_WG_PTS.py"
   synerror = traceback.format_exc().splitlines()[-1]
   return line, filename, synerror

# Load required toolboxes...
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")

# Set the Geoprocessing environment...
gp.configKeyword = "USERS"
gp.OverWriteOutput = True
```

```
#=============
# Connections
#=============
# Set workspace
gp.workspace = ("Database Connections\\setb2@phoenix.sde")
sdeConnExecute = gp.CreateObject("ARCSDESQLEXECUTE","Database Connections\\setb2@phoenix.sde")

#=============
# Variables
#=============
### Set variables for input Warren Group data table, "new streets" output data table
### real: WG.WG2009_STATE_ATTRIB (2,752,102 records), test: SETB2.WG2009_SAVOYROWE (1772 records)
### NOTE: this script assumes that input table has a FLAG field of [NUMBER(1)]

#input_tbl = "SETB2.WG2009_STATE_ATTRIB"
#input_tbl = "SETB2.WG2009_STATE_BASIC"
#input_tbl = "SETB2.WG2009_SAVOYROWE"   # LAT/LON in decimal degrees (geographic)
input_tbl = "SETB2.WG2009_SAVOYROWE_BASIC"   # LAT/LON in MA state plane meters (projected)

output_tbl = input_tbl + "_NEW"     # output table of "new" street names (no lat/lon)

output_fc = input_tbl + "_PTS"      # output feature class

#-----------------------------------------------------------------------------------------------------------
# PART 1
# Prep the Warren Group data table (SETB2.WG2009_SAVOYROWE) before processing :
#   0) Clean Up
#      a) Delete output table if Exists
#      b) Reset FLAG = NULL
#   1) Tag only one occurance of a distinct town_id, street that has positive lat/lon values: "existing" streets
#      Set FLAG = 1
#   2) For the rest ("new streets"):
#      Create table for distinct town_id, street records where lat/lon = 0
#   3) Group SQL into a list
#   4) Run SQL using the sdeConnExecute object created in the connections section
#
#-----------------------------------------------------------------------------------------------------------
print "++++++++++++++++++++++++++++++++++++++++++++++\n"
print "Pre-Processing Warren Group Table....\n"
print "++++++++++++++++++++++++++++++++++++++++++++++\n"

# Delete output table if already exists
if gp.Exists(output_tbl):
   gp.Delete_Management(output_tbl)
   print "Deleted data table: " + output_tbl

# SQL to Reset the FLAG column
sql_reset = "UPDATE " + input_tbl + " SET FLAG = NULL"

# SQL to Update FLAG = 1 for "existing" streets
#sql_flag1 = "UPDATE " + input_tbl + " SET FLAG = 1 WHERE (street, objectid) IN  (select street, min(objectid) FROM
" + input_tbl + " WHERE lat <> 0 GROUP BY town_id, street)"
# SQL for SAVOYROWE_BASIC
```

```
sql_flag1 = "UPDATE " + input_tbl + " SET FLAG = 1 WHERE (street, objectid) IN  (select street, min(objectid) FROM
" + input_tbl + " WHERE x <> 0 GROUP BY town_id, street)"

# SQL to create table for "new" streets
#sql_flag2 = "CREATE TABLE   " + output_tbl + "  as (SELECT DISTINCT town_id, street, lat, lon FROM " + input_tbl +
" WHERE street NOT IN (SELECT street FROM " + input_tbl + " WHERE flag = 1 GROUP BY town_id, street)) ORDER
BY town_id, street"
# SQL for SAVOYROWE_BASIC
sql_flag2 = "CREATE TABLE   " + output_tbl + "  as (SELECT DISTINCT town_id, street, x, y FROM " + input_tbl + "
WHERE street NOT IN (SELECT street FROM " + input_tbl + " WHERE flag = 1 GROUP BY town_id, street)) ORDER BY
town_id, street"

# Group all SQL statements into a list
SQLList = [sql_reset, sql_flag1, sql_flag2]


print "++++++++++++++++++++++++++++++++++++++++++++\n"

try:
# For each SQL statement passed in, execute it.
   for sql in SQLList:
      print "Execute SQL Statement: " + sql
      try:
         # Pass the SQL statement to the database.
         sdeReturn = sdeConnExecute.Execute(sql)
      except Exception, ErrorDesc:
         print ErrorDesc
         sdeReturn = False

      # If the return value is a list (a list of lists), display each list as a row from the
      #   table being queried.
      if isinstance(sdeReturn, list):
         print "Number of rows returned by query: " + len(sdeReturn), "rows"
         for row in sdeReturn:
            print row
         print "++++++++++++++++++++++++++++++++++++++++++++\n"
      else:
         # If the return value was not a list, the statement was most likely a DDL statment.
         #   Check its status.
         if sdeReturn == True:
            print "ran sucessfully."
            print "++++++++++++++++++++++++++++++++++++++++++++\n"
         else:
            print "FAILED."
            print "++++++++++++++++++++++++++++++++++++++++++++\n"

   # Commit the changes
   sdeConnExecute.CommitTransaction()
   print "Commited Transaction\n"
   print "Done prepping Warren Group table."
   print "++++++++++++++++++++++++++++++++++++++++++++\n"

except Exception, ErrorDesc:
      print Exception, ErrorDesc
```

```
except:
   print "Problem executing SQL."

#-------------------------------------------------------------------------------------------------------------
# PART 2
# Create a new Feature Class :
#   0) Clean Up
#      a) Delete output feature class if Exists
#   1) Create a new 'blank' feature class
#      a) create a spatial reference object
#      b) Use a projection file to define the spatial reference's properties
#      c) Run CreateFeatureClass using the spatial reference object
#   2) Set up a NEW_FIELDS array
#   3) Add fields and values to output_fc
#
#-------------------------------------------------------------------------------------------------------------
print "++++++++++++++++++++++++++++++++++++++++++++++\n"
print "Creating a new feature class... \n"
print "++++++++++++++++++++++++++++++++++++++++++++++\n"

try:
   # Delete output feature class if already exists
   if gp.Exists(output_fc):
      gp.Delete_Management(output_fc)
      print "Deleted feature class: " + output_fc + "\n"
      print "++++++++++++++++++++++++++++++++++++++++++++++\n"

   # Create a new point feature class

   # Create a spatial reference object
   # didn't work: "Geoprocessing error on line 187 of E:\Python\final_project\gen_WG_PTS_7a.py"
   # spatialRef = gp.CreateObject("spatialreference")

   # Use a projection file to define the spatial reference's properties
   # spatialRef.CreateFromFile(r"C:\\Program Files\\ArcGIS\\Coordinate Systems\\Projected Coordinate
Systems\\State Plane\\NAD 1983\\NAD 1983 StatePlane Massachusetts Mainland FIPS 2001.prj")

   # CreateFeatureClass_management (out_path, out_name, geometry_type, template, has_m, has_z,
spatial_reference, config_keyword, spatial_grid_1, spatial_grid_2, spatial_grid_3)
   # gp.CreateFeatureclass(gp.workspace, output_fc, "POINT", "", "", "", spatialRef)
   gp.CreateFeatureclass(gp.workspace, output_fc, "POINT")

   print "feature class created "

   print "++++++++++++++++++++++++++++++++++++++++++++++\n"
   print "adding fields to " + output_fc + " ...\n"
   print "++++++++++++++++++++++++++++++++++++++++++++++\n"

##   # Create new fields array   # [FIELD_NAME, FIELD_TYPE, FIELD_PRECISION, FIELD_SCALE, FIELD_LENGTH]
##   NEW_FIELDS = [
##      ["PROPID", "LONG", "", "", "10"],   # 0
##      ["MAPREF", "TEXT", "", "", "25"],   # 1
##      ["SOURCE", "TEXT", "", "", "1"],    # 2
##      ["INACTVFL", "TEXT", "", "", "1"],  # 3
```

```
##       ["COUNTY", "TEXT", "", "", "15"],   # 4
##       ["CITY", "TEXT", "", "", "20"],     # 5
##       ["STATE", "TEXT", "", "", "2"],     # 6
##       ["STNUM", "LONG", "", "", "10"],    # 7
##       ["STNUMEXT", "TEXT", "", "", "5"],  # 8
##       ["STREET", "TEXT", "", "", "25"],   # 9
##       ["ZIPCODE", "TEXT", "", "", "5"],   # 10
##       ["LAT", "DOUBLE", "38", "8", "38"], # 11
##       ["LON", "DOUBLE", "38", "8", "38"], # 12
##       ["OWNER1", "TEXT", "", "", "25"],   # 13
##       ["OWNER1LN", "TEXT", "", "", "25"], # 14
##       ["OWNER2", "TEXT", "", "", "25"],   # 15
##       ["OWNER2LN", "TEXT", "", "", "25"], # 16
##       ["STATEUSE", "TEXT", "", "", "3"],  # 17
##       ["YEARBUILT", "TEXT", "", "", "4"], # 18
##       ["REALTOWN", "TEXT", "", "", "35"], # 19
##       ["TOWN_ID", "SHORT", "", "", "5"],  # 20
##       ["FLAG", "SHORT", "", "", "1"]      # 21
##       ]

    # Create new fields array   # for processing WG2009_SAVOYROWE_BASIC table
   NEW_FIELDS = [
      ["PROPID", "LONG", "", "", "10"],   # 0
      ["MAPREF09", "TEXT", "", "", "25"], # 1
      ["SOURCE", "TEXT", "", "", "1"],    # 2
      ["INACTVFL", "TEXT", "", "", "1"],  # 3
      ["COUNTY", "TEXT", "", "", "15"],   # 4
      ["CITY", "TEXT", "", "", "20"],     # 5
      ["STATE", "TEXT", "", "", "2"],     # 6
      ["STNUM", "LONG", "", "", "10"],    # 7
      ["STNUMEXT", "TEXT", "", "", "5"],  # 8
      ["STREET", "TEXT", "", "", "25"],   # 9
      ["ZIPCODE", "TEXT", "", "", "5"],   # 10
      ["LAT", "DOUBLE", "19", "10", "8"], # 11
      ["LON", "DOUBLE", "19", "10", "8"], # 12
      ["OWNER", "TEXT", "", "", "25"],    # 13
      ["GC_SOURCE", "TEXT", "", "", "15"],# 14
      ["SQ_FT", "DOUBLE", "", "", "25"],  # 15
      ["STATEUSE", "TEXT", "", "", "3"],  # 16
      ["REALTOWN", "TEXT", "", "", "35"], # 17
      ["TOWN_ID", "SHORT", "", "", "5"],  # 18
      ["FLAG", "SHORT", "", "", "1"]      # 19
      ]


   # Add Fields to output_fc
   # AddField_management (in_table, field_name, field_type, field_precision, field_scale, field_length,
   #                      field_alias, field_is_nullable, field_is_required, field_domain)

   for (FIELD_NAME, FIELD_TYPE, FIELD_PRECISION, FIELD_SCALE, FIELD_LENGTH) in NEW_FIELDS:
      print "Adding: " + FIELD_NAME + ": " + FIELD_TYPE + " (" + FIELD_LENGTH + ")"
      gp.AddField_management(output_fc, FIELD_NAME, FIELD_TYPE, FIELD_PRECISION, FIELD_SCALE,
FIELD_LENGTH, "", "NULLABLE", "NON_REQUIRED", "")
```

```
   print "done adding fields to: " + output_fc

#---------------------------------------
# Return GEOPROCESSING specific errors
except arcgisscripting.ExecuteError:
   line, filename, err = trace()
   gp.AddError("Geoprocessing error on " + line + " of " + filename + " :")
   print gp.AddError("Geoprocessing error on " + line + " of " + filename + " :")
   for msg in range(0, gp.MessageCount):
      if gp.GetSeverity(msg) == 2:
         gp.AddReturnMessage(msg)
      print gp.AddReturnMessage(msg)
   print "Geoprocessing error on " + line + " of " + filename + " :\n" + str(err)

# Return any PYTHON or system specific errors
except:
   line, filename, err = trace()
   gp.AddError("Python error on " + line + " of " + filename)
   gp.AddError(err)
   print "Python error on " + line + " of " + filename + " :\n" + str(err)

#---------------------------------------

#-----------------------------------------------------------------------------------------------------------
# PART 3
# Process the Warren Group data table (SETB2.WG2009_SAVOYROWE) into a point feature class:
#   0) Clean Up: delete rows in output_fc
#   1) Create point and cursor objects
#   2) Open the input table
#   3) Loop through input table where FLAG = 1
#      assign the  x, & y properties of the pnt object,
#       propid value to the PROPID column,
#       create new row ssign pnt object to the shapefield,
#       save the insert.
#
#-----------------------------------------------------------------------------------------------------------
print "+++++++++++++++++++++++++++++++++++++++++++++\n"
print "Processing LAT LON values to points....\n"
print "+++++++++++++++++++++++++++++++++++++++++++++\n"

# initialize cursor and row objects to ensure that they exist when del statement executes in the finally clause
inRows, inRow = None, None
outRows = None

try:
##    # Clean up: delete rows in output_fc
##    delRows = gp.deletefeatures(output_fc)
##    print "deleted features in: " + output_fc
##    print "+++++++++++++++++++++++++++++++++++++++++++++\n"

   # Second list of include fields -- only the ones we need
   #INCLUDE_FIELDS = ["PROPID", "LAT", "LON", "FLAG"]
   INCLUDE_FIELDS = ["PROPID", "X", "Y", "FLAG"]
```

```
   # Open searchcursor
   inRows = gp.SearchCursor(input_tbl, "FLAG = 1", INCLUDE_FIELDS)
   inRows.Reset
   inRow = inRows.Next()

   # Open insertcursor
   outRows = gp.InsertCursor(output_fc)
   pnt = gp.CreateObject("Point") #creates the point object for the output_fc
   feat = outRows.NewRow()

   print "saving points to: " + output_fc + " ..."
   print "++++++++++++++++++++++++++++++++++++++++++++\n"

   while inRow:
      if inRow.GetValue(INCLUDE_FIELDS[3]) == 1:  # "FLAG = 1" in the SearchCursor is not working, so I'm setting
my criteria here
         # Create new row for output feature
         #feat = outRows.NewRow()

         # Assign the pnt object properties to the geometry field
         feat.Shape = pnt
         pnt.x = inRow.GetValue(INCLUDE_FIELDS[1])
         pnt.y = inRow.GetValue(INCLUDE_FIELDS[2])
         #print "LAT (x): " + str(pnt.x)
         #print "LON (y): " + str(pnt.y)

         #Assign attribute values
         feat.PROPID = inRow.GetValue(INCLUDE_FIELDS[0])
         #print "PROPID: " + str(feat[0])

         # Insert the feature
         outRows.InsertRow(feat)
         # Get next feature in searchcursor
         inRow = inRows.Next()
         #print "getting next row"

   print "done saving points to: " + output_fc

#---------------------------------------
# Return GEOPROCESSING specific errors
except arcgisscripting.ExecuteError:
   line, filename, err = trace()
   gp.AddError("Geoprocessing error on " + line + " of " + filename + " :")
   print gp.AddError("Geoprocessing error on " + line + " of " + filename + " :")
   for msg in range(0, gp.MessageCount):
      if gp.GetSeverity(msg) == 2:
         gp.AddReturnMessage(msg)
      print gp.AddReturnMessage(msg)
   print "Geoprocessing error on " + line + " of " + filename + " :\n" + str(err)

# Return any PYTHON or system specific errors
except:
   line, filename, err = trace()
   gp.AddError("Python error on " + line + " of " + filename)
```

```
    gp.AddError(err)
    print "Python error on " + line + " of " + filename + " :\n" + str(err)

# These actions will always execute
finally:
    # Delete row(s) and cursor(s)
    del inRow, inRows, outRows
    gp.AddMessage("\n   ** Cursor and row have been deleted ** \n")
    print "\n   ** Cursor and row have been deleted ** \n"

#-------------------------------------

#-----------------------------------------------------------------------------------------------------------
# PART 4
# Copy the attributes from input_tbl to output_fc
#   0)
#   1) Set up SQL statement
#   2) Run SQL using the sdeConnExecute object
#
#-----------------------------------------------------------------------------------------------------------
print "++++++++++++++++++++++++++++++++++++++++++++++\n"
print "copying attributes ..."
print "++++++++++++++++++++++++++++++++++++++++++++++\n"

# SQL to Update fields in output_fc with input_tbl values
# for SAVOYROWE
#sql_updatepts = "UPDATE " + output_fc + " pts SET (pts.MAPREF, pts.SOURCE, pts.INACTVFL, pts.COUNTY,
pts.CITY, pts.STATE, pts.STNUM, pts.STNUMEXT, pts.STREET, pts.ZIPCODE, pts.LAT, pts.LON, pts.OWNER1,
pts.OWNER1LN, pts.OWNER2, pts.OWNER2LN, pts.STATEUSE, pts.YEARBUILT, pts.REALTOWN, pts.TOWN_ID,
pts.FLAG) = (select tbl.MAPREF, tbl.SOURCE, tbl.INACTVFL, tbl.COUNTY, tbl.CITY, tbl.STATE, tbl.STNUM,
tbl.STNUMEXT, tbl.STREET, tbl.ZIPCODE, tbl.LAT, tbl.LON, tbl.OWNER1, tbl.OWNER1LN, tbl.OWNER2,
tbl.OWNER2LN, tbl.STATEUSE, tbl.YEARBUILT, tbl.REALTOWN, tbl.TOWN_ID, tbl.FLAG FROM  " + input_tbl + " tbl
WHERE pts.propid = tbl.propid)"
# for SAVOYROWE_BASIC
sql_updatepts = "UPDATE " + output_fc + " pts SET (pts.MAPREF09, pts.SOURCE, pts.INACTVFL, pts.COUNTY,
pts.CITY, pts.STATE, pts.STNUM, pts.STNUMEXT, pts.STREET, pts.ZIPCODE, pts.LAT, pts.LON, pts.OWNER,
pts.GC_SOURCE, pts.SQ_FT, pts.STATEUSE, pts.REALTOWN, pts.TOWN_ID, pts.FLAG) = (select tbl.MAPREF09,
tbl.SOURCE, tbl.INACTVFL, tbl.COUNTY, tbl.CITY, tbl.STATE, tbl.STNUM, tbl.STNUMEXT, tbl.STREET, tbl.ZIPCODE,
tbl.X, tbl.Y, tbl.OWNER, tbl.GC_SOURCE, tbl.SQ_FT, tbl.STATEUSE, tbl.REALTOWN, tbl.TOWN_ID, tbl.FLAG FROM  "
+ input_tbl + " tbl WHERE pts.propid = tbl.propid)"

try:
    # Pass the SQL statement to the database.
    sdeReturn = sdeConnExecute.Execute(sql_updatepts)

    # If the return value is a list (a list of lists), display each list as a row from the table being queried.
    if isinstance(sdeReturn, list):
        print "Number of rows returned by query: " + len(sdeReturn), "rows"
        for row in sdeReturn:
            print row
        print "++++++++++++++++++++++++++++++++++++++++++++++\n"
    else:
        # If the return value was not a list, the statement was most likely a DDL statment.
        #   Check its status.
```

```
        if sdeReturn == True:
            print "ran sucessfully."
            print "++++++++++++++++++++++++++++++++++++++++++++++++\n"
        else:
            print "FAILED."
            print "++++++++++++++++++++++++++++++++++++++++++++++++\n"
    # Commit the changes
    sdeConnExecute.CommitTransaction()
    print "Commited Transaction\n"
    print "++++++++++++++++++++++++++++++++++++++++++++++++\n"

    print "done updating attributes "

#---------------------------------------
# Return GEOPROCESSING specific errors
except arcgisscripting.ExecuteError:
    line, filename, err = trace()
    gp.AddError("Geoprocessing error on " + line + " of " + filename + " :")
    print gp.AddError("Geoprocessing error on " + line + " of " + filename + " :")
    for msg in range(0, gp.MessageCount):
        if gp.GetSeverity(msg) == 2:
            gp.AddReturnMessage(msg)
        print gp.AddReturnMessage(msg)
    print "Geoprocessing error on " + line + " of " + filename + " :\n" + str(err)

except Exception, ErrorDesc:
        print Exception, ErrorDesc

# Return any PYTHON or system specific errors
except:
    line, filename, err = trace()
    gp.AddError("Python error on " + line + " of " + filename)
    gp.AddError(err)
    print "Python error on " + line + " of " + filename + " :\n" + str(err)

#---------------------------------------

# free memory
del gp
print "done processing script"
```