

# Weather Underground Data Integration

December 4, 2014

## Introduction

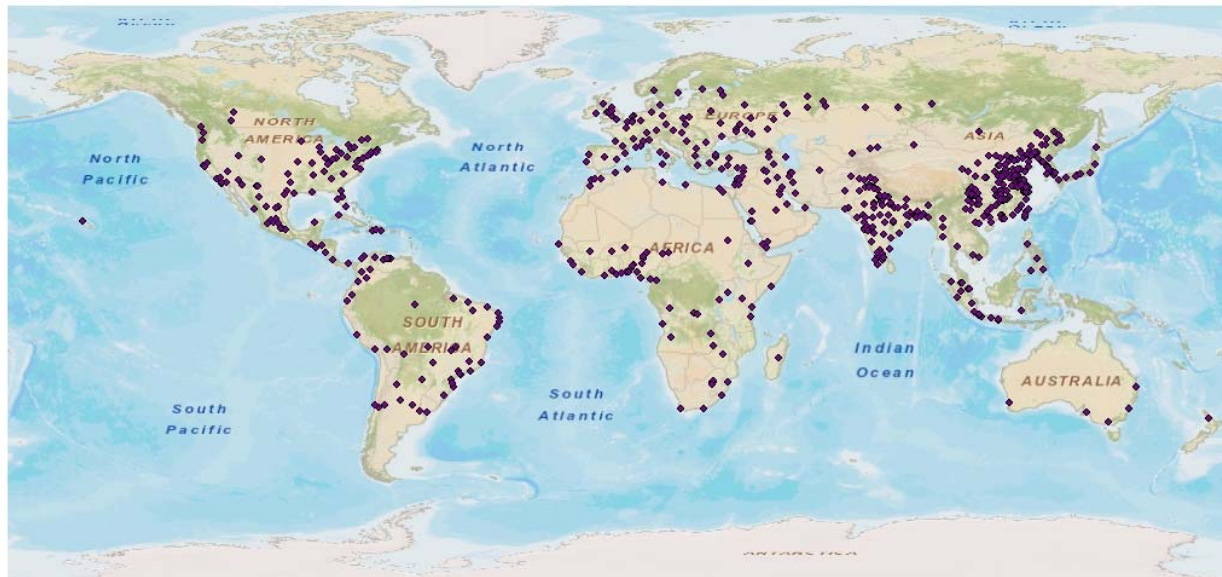
This project was born from a conversation with a fellow traveler who lamented the lack of a tool to choose destination based on climate preferences. A quick internet search yielded some GIS data sources which I promptly shared. I was immediately rebuffed with the remark “I have no idea what to do with that.” I immediately recognized the opportunity to use my developing GIS skillset to address this problem. The concept was simple, obtain climate data and incorporate it into a GIS. The execution; however, revealed that there were many details that would require attention. That said, this project serves as a proof of concept for a larger system that would add value to the travelling community. Moreover, I discovered a new start up, [whatsitlike.com](http://whatsitlike.com), that is doing something very similar. I have already made contact with the founder in order to gain insight into how they are executing the project. The driving principal of their project is that it is easy to choose *where* to go, and much more difficult to decide *when* to go. A project like mine could serve as a foundation or template for a larger project similar to the aforementioned.

I expect that the results of my tool would be consumed by travelers who want to determine the best times and destinations for their adventures. Consequently, this project should ultimately be delivered as an online portal that is easily used and understood by a non-specialist. Furthermore, the code should be expanded to include additional data such as real time weather, in order to maximize its usefulness.

Due to the nature of my project, I did not have a pilot study area, but instead used the World Database of Large Urban Areas, 1950 – 2050<sup>1</sup> (Figure 1) as the foundational map layer. The shapefile contains both city and country names, and provided a suitable driver for the project. Moreover, the shapefile required no additional formatting or processing in order to be used, thus freeing me to focus on development. I reviewed several sources of climate data and settled on Weather Underground ([wunderground.com](http://wunderground.com)) as a data source because of their robust data, free REST API, and familiarity with their website.

---

<sup>1</sup> <https://nordpil.com/resources/world-database-of-large-cities/>



Sources: Esri, HERE, DeLorme, USGS, Intermap, increment P Corp., NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, MapmyIndia, © OpenStreetMap contributors, and the GIS User Community

Figure 1

## Data Sources

This project uses two data sources: The World Database of Large Urban Areas (TWD), and the Weather Underground REST API. The key data source is the Weather Underground API which provides all of the climate data. The World Database of Large Urban Areas serves as a suitable test data set for the program given the inclusion of both country and city names in the shape file. I had to apply for and obtain a key in order to use the Weather Underground API. The key is free and available via an automated online form, and was generally easy to obtain. However, the free key does have a limitation on the total number of requests per day, and the requests per minute.

I did not alter the data in any way prior to incorporation into the GIS, the data was formatted by my program instead. I opted for this method because I believe that it provides a

better overall experience, as opposed to forcing a user to conform to very specific data format standards. However, I was not able to account for every eventuality and did have to include some specific logic to address peculiarities of TWD. Specifically, TWD has both a city name field and a city name alternate field. The alternate field contains a more common spelling of a city name if one exists. For example, the city Yangon in Myanmar has the alternate name Rangoon. The choice to place the more common spelling in the alternate column required me to include logic that made use of the alternate spelling if it existed. It should be noted that not every city has an alternate spelling. I could have edited the attribute table to populate every alternate name field, however, this ran counter to the spirit of the project, namely doing the work in the code rather than manually.

The data was formatted in several ways within the Python script: data was URL encoded, spaces were replaced with underscores, commas were removed, and compound cities were broken down to their constituent components. Data sent to the Weather Underground REST API had to be formatted, spaces in city names needed to be replaced with underscores. Spaces in country names were URL encoded by replacing spaces with %20. Several city names had commas in them, which were also removed. Each of these formatting tasks was performed with a simple substitution via Python's built in replace function. (Figure 2)

```
city = city.replace(" ", "_")
```

Figure 2

The authors of TWD chose to group some cities together with their names separated by a dash. Los Angeles, for example, was grouped with other cities. (Figure 3)

Los Angeles-Long Beach-Santa Ana

Figure 3

I worked around this choice by splitting the data prior to retrieving results via the API call. The Python function split was used for this. The split command breaks a string into multiple strings using a delimiting character, in this case the dash (Figure 4)

```
cities = city.split('-')
```

Figure 4

## Methodology

I did not build any model in model builder because my model was specifically meant to retrieve data from an online API and insert it into the attribute table supplied as one of the parameters. Consequently, the entire project is written in Python. I now wonder, in retrospect, if I should have had a more analytical and visual component to my project. I was focused entirely on the programming tasks and failed to consider those aspects.

I did some additional programming for my project that can be broken into three main categories: field validation, data formatting, iteration and data retrieval. The first thing that my script does is look for the existence of lowTemp and highTemp fields that will contain the end result of the script. The fields are created if they do not exist. The arcpy ListFields() function is used with a selector to determine the existence of each field. A zero length result represents a missing field. The script branches on the length and creates the field if necessary. This was the last function that I added to the project. (Figure 5)

```
fields = arcpy.ListFields(fc, "lowTemp")
if len(fields) > 0:
    print "lowTemp exists"
else:
    arcpy.AddField_management(fc, "lowTemp", "TEXT", "", "", 50)
    print "No low temp Field"
    arcpy.AddMessage("Low Temp Field Added")
```

Figure 5

The data formatting was discussed in the data section, so I will only provide a brief review in this section. The city and country names were formatted with the Python replace function, and composite city names were divided using the split function. There are additional formatting tasks that should be included in the next iteration of this application, they were omitted in the interest of time.

The main function of the program is the API request, so I began development with that component. I chose to use HTTPLIB to execute the HTTP GET request. My first step was to write a simple script that executed the GET request and printed the result and HTTP status. (Figure 6)

```
conn.request("GET", "/api/613393ec1cc60128/almanac/q/CA/Los_Angeles.xml ")
r1 = conn.getresponse()
```

Figure 6

When I received a 200 OK response from the REST service I then began to parse the data in order to locate the components that were necessary for my program. This proved to be more difficult than I expected because accessing the elements within the XML response was not a clean process. Consequently, I opted to switch to JSON for the response format. Python has a native JSON library that handles all of the parsing and returns a key/value array. The key value array made locating and accessing the data much easier.

The next step was to open a feature class, select city names for a given country, and iterate over the data retrieving weather data for each city. I began this process with a select cursor in order to simplify the process, and changed to an update cursor later on. This wasn't necessary because the update cursor can be used in the same way a select cursor can; however, I did not know that at the time. I did encounter some difficulty formatting and using the where clause in the cursor. The where clause is necessary to limit the cities in the cursor to the selected country; however, several initial attempts resulted in errors. AddFieldDelimiters ultimately worked and allowed me to execute the cursor. (Figure 7)

```
fieldname = "Country"
delimfield = arcpy.AddFieldDelimiters(fc,fieldname)
cursor = arcpy.da.UpdateCursor(fc, ["CITY", "City_Alter", "Country", "lowTemp", "highTemp"], delimfield
+" = " + selectCountry + """)
```

Figure 7

I tested the cursor with hard coded values for the input feature class and country so that I could run and debug the code in Pyscripter. The script performs one HTTP GET request for each city and writes the results to the attribute table. (Figure 8)

```
conn.request("GET", "/api/613393ec1cc60128/almanac/q/" + encselectCountry + "/" + workCity + ".json")
r1 = conn.getresponse()
if r1.status == 200:
    data1 = r1.read()
    #print data1
    parsed_json = json.loads(data1)
```

Figure 8

The final step was to make the input feature class and country parameters and run the script in ArcMap. The arcpy arcpy.GetParameterAsText() function was used to obtain the input. The script was added to a toolbox in my final project folder, and the parameters were configured via the script properties dialog. I then ran and tested the script in ArcMap and noted that no useful feedback was given in the execution dialog. I was able to print and see messages

in Pyscripter; however, these messages did not come through to the ArcMap interface. I solved this problem with the `arcpy.AddMessage()` function, which wrote all of my messages to both the script dialog and the geoprocessing results window. The script was now fully functional and I wanted to incorporate it into an add-in so that the user could simply click a button to run the program.

My initial attempt at creating an add-in, which I could not make work, involved pasting all of my Python code into the click method for the button. This would execute the script when the button was pressed; however, it would not prompt the user for the input parameters. I read through the ArcGIS help and could not find a good way to make this happen.<sup>2</sup> There are four functions listed: `OpenDialog`, `SaveDialog`, `GPToolDialog`, and `MessageBox`, none of which allowed for string input.

Some research revealed that the `GPToolDialog` would allow me to execute a tool in a toolbox, and this solution worked for my purposes, though it is not ideal. I updated my add-in code with a single line to execute my tool in the toolbox and was successful in creating the application that I desired. (Figure 9)

```
pythonaddins.GPToolDialog(r'G:\586\final\Final.tbx', 'Wunderground')
```

Figure 9

The add-in now creates a button on a toolbar that launches the Wunderground script from the Final toolbox when clicked. The script functions in the same way as it would if the user double clicked the script in the toolbox in the catalog. The flow of control is shown in figure 10, and the development process is shown in figure 11.

---

<sup>2</sup> [http://resources.arcgis.com/en/help/main/10.1/index.html#/The\\_pythonaddins\\_module/014p00000021000000/](http://resources.arcgis.com/en/help/main/10.1/index.html#/The_pythonaddins_module/014p00000021000000/)

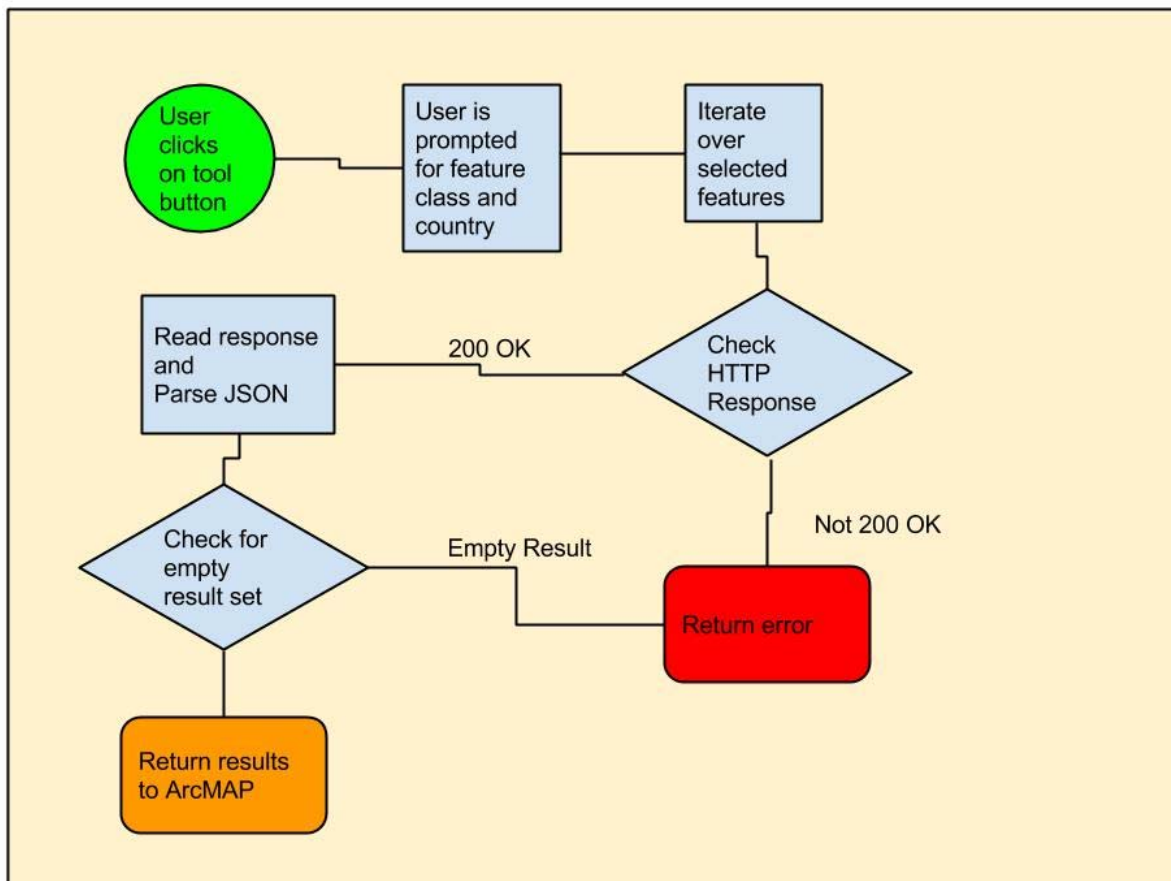


Figure 10

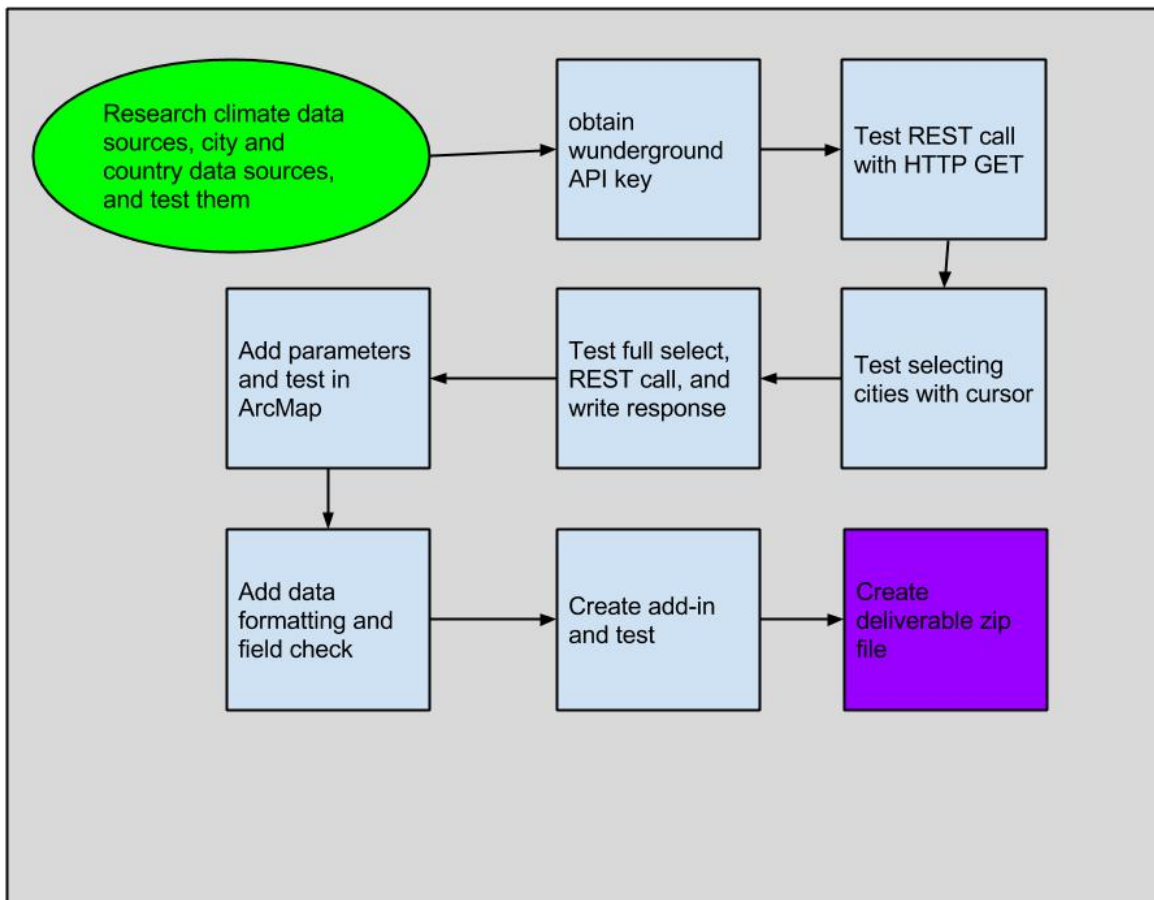


Figure 11



## Installation and Execution

1. Unzip the contents of the zip file into g:\586\final\
2. Double click the add-in file to install the ArcMap add-in (Figure 12)

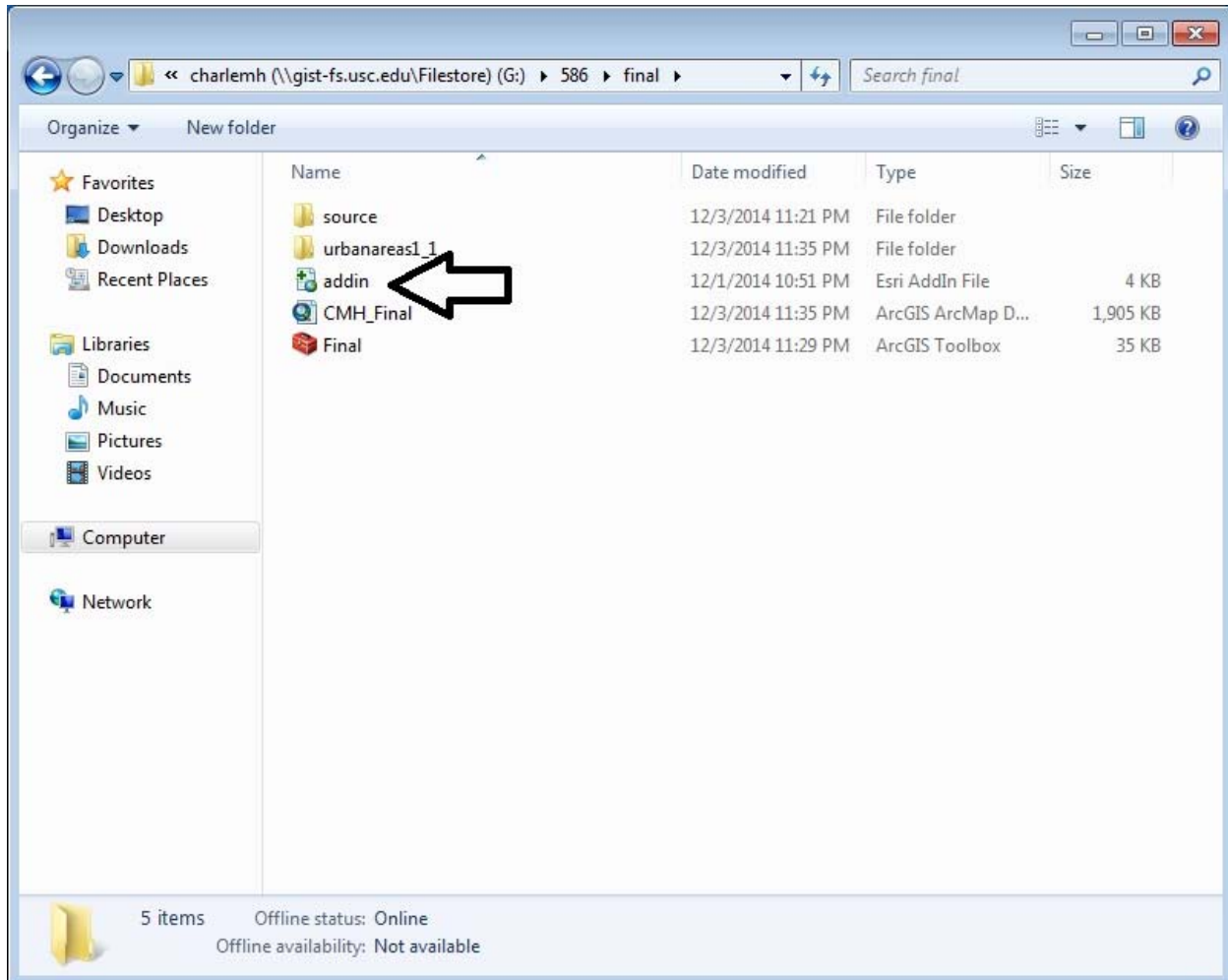
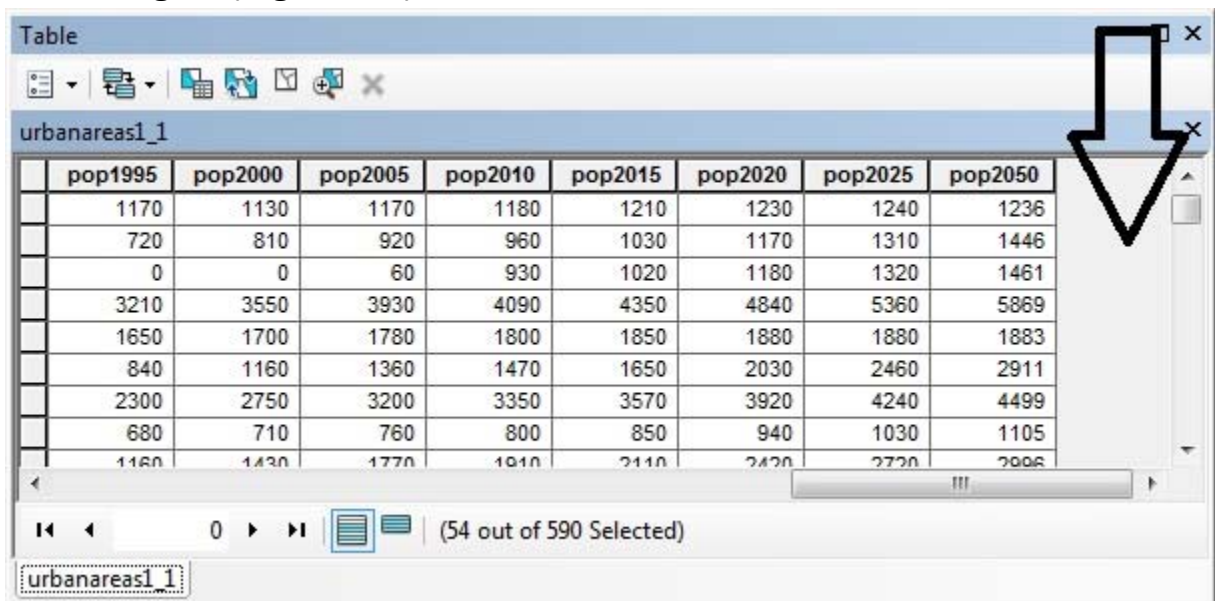


Figure 12

3. Launch ArcMap
4. Click on Customize -> Add-In Manager to confirm the installation
5. If the tool bar with the AlmanacData button does not show, right click on the toolbar and click the check box next to the Wunderground add-in in order to show the button.
6. Open the CMH\_Final.mxd map document
7. Open the attribute table and confirm that the low and high temp fields do not exist and do not contain data, they are the last fields on the right. (Figure 13)



	pop1995	pop2000	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050		
	1170	1130	1170	1180	1210	1230	1240	1236		
	720	810	920	960	1030	1170	1310	1446		
	0	0	60	930	1020	1180	1320	1461		
	3210	3550	3930	4090	4350	4840	5360	5869		
	1650	1700	1780	1800	1850	1880	1880	1883		
	840	1160	1360	1470	1650	2030	2460	2911		
	2300	2750	3200	3350	3570	3920	4240	4499		
	680	710	760	800	850	940	1030	1105		
	1160	1430	1770	1910	2110	2420	2720	2996		

(54 out of 590 Selected)

urbanareas1\_1

Figure 13

8. Run the tool

- a. Set the input feature class to  
G:\586\final\urbanareas1\_1\urbanareas1\_1.shp
- b. Enter Bulgaria for the Country and click ok. (Figure 14)
  - i. Other countries are also available such as Canada, USA, China, etc. I would ask that you limit your choices to countries with fewer entries in the attribute table.

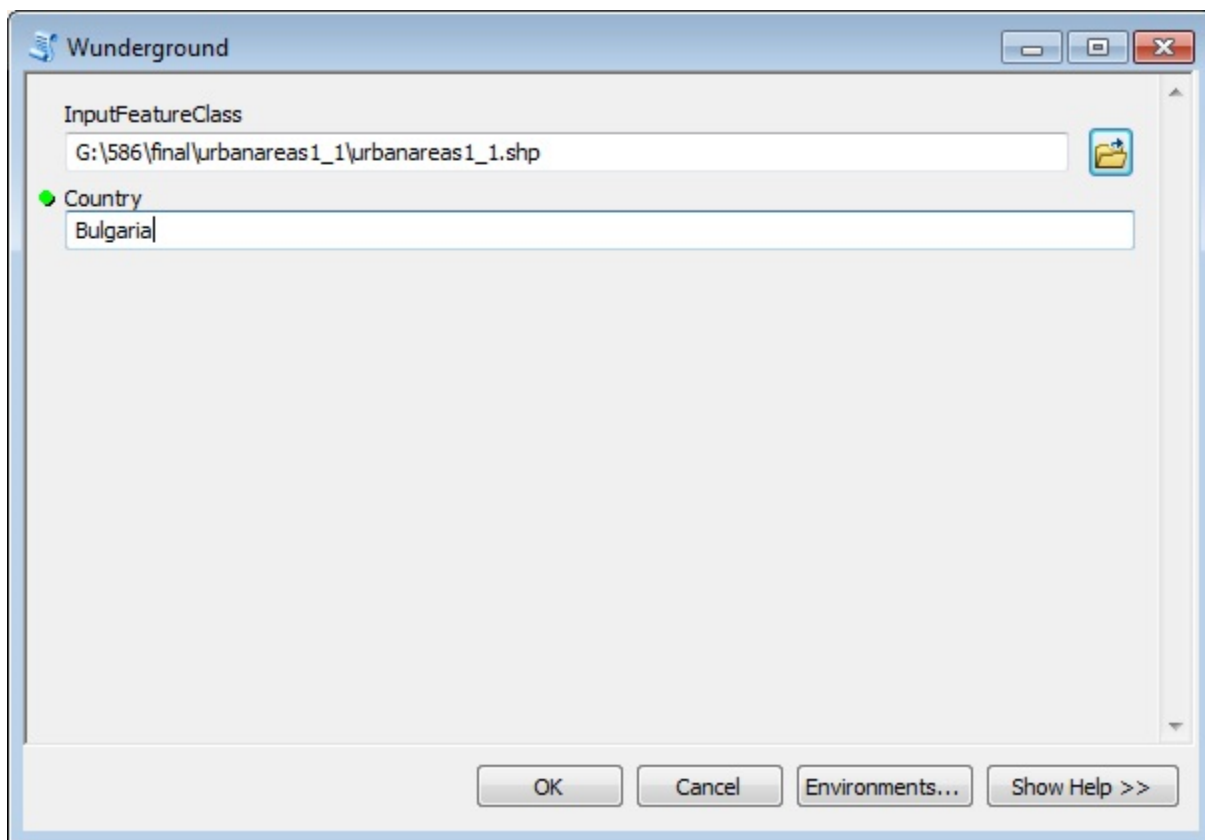


Figure 14

## 9. Review the results

- a. The script dialog should show successful execution (Figure 15)
- b. The attribute table will now have lowTemp and highTemp fields containing temperature data for the cities in the selected country (Figure 16)

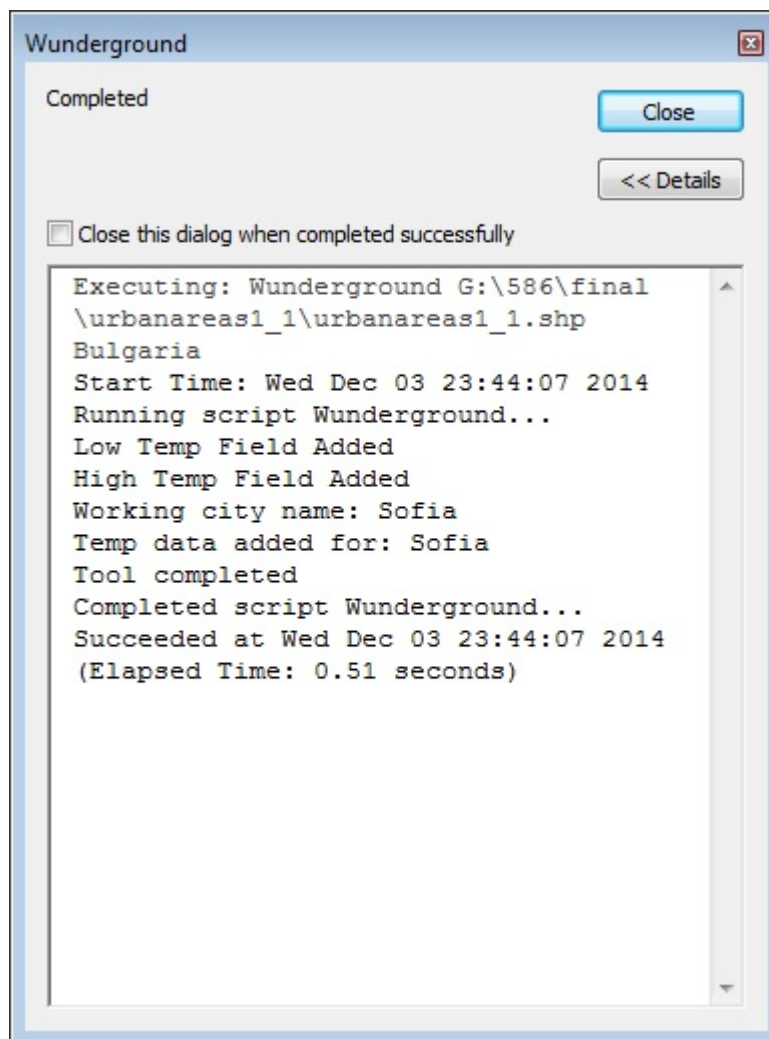


Figure 15

	pop2005	pop2010	pop2015	pop2020	pop2025	pop2050	lowTemp	highTemp
1	1170	1180	1210	1230	1240	1236	28	40
2	920	960	1030	1170	1310	1446		
3	60	930	1020	1180	1320	1461		
4	3930	4090	4350	4840	5360	5869		
5	1780	1800	1850	1880	1880	1883		
6	1360	1470	1650	2030	2460	2911		
7	3200	3350	3570	3920	4240	4499		
8	760	800	850	940	1030	1105		
9	1770	1810	2110	2420	2720	2908		

Figure 16

This program solves the problem of retrieving climate data for a given set of cities and writes the results into the attribute table. This enables the creation of GIS applications that can select locations based on this climate data. The inspiration for this project was travel; however, this functionality could also be useful in agriculture and climate research. When I began this project I set out to retrieve climate data and insert it into the GIS, and this application has had moderate success in doing that.

## Conclusion

There are a number of problems with this project; however, I am generally pleased with the results of my first iteration. I was able to successfully contact the REST service, obtain climate data, and write it to the attribute table, which was my initial goal. That said, the current method is clunky at best and should use location, rather than place name, in the request for data. This would dramatically improve the results by eliminating all of the data formatting currently required. Instead of matching city and country name spellings in the attribute table with those in the Weather Underground dataset, we could use a location which should generally match in any two data sets. Moreover, this would remove the constraint that city and country name need to be in the attribute table. Instead, data could be obtained for arbitrary locations using data that is generally available in a GIS.

I would have liked to have an interactive selection via the mouse; however, this proved to be too difficult in this early stage of my experience with programming for ArcGIS. Interactive

selection presented several problems: constraining the selection to data within the attribute table, passing values from ArcMap to my script, and the general mechanics of developing the interactive selection. I suspect that when properly implemented the interactive selection would detect a null selection and not continue execution; however, I was not able to determine how to do that. Working from the example in the Model Builder textbook, I was not able to reach this goal. Moreover, I need to spend some more time understanding how Model Builder and Python interact. I found it difficult to mix the two and keep all of the parameters and processes straight. Again, working from the examples in the book I was not able to achieve my goal. I believe that one reason for this is that the examples implement simpler versions of the tools that I was attempting to make, and I was not able to scale them up.

I learned, near the end of the process, that using the with cursor as cursor data format will greatly improve my application because it always releases locks on the data set when it closes.<sup>3</sup> My cursor code, on the other hand, does not release locks when it fails which creates several problems. One example, testing the existence of fields and creating them. My first three attempts at this failed and produced errors, which I expected. They also left all of my data in a locked state which I did not expect. I employed two strategies to work around this: close and reopen ArcMap, or comment out most of my code and rerun the script. Both of these solutions released all of the locks on my data set.

I did find that the online GIS community is vibrant, and I was able to solve a number of problems by reading through online resources, especially [gis.stackexchange.com](http://gis.stackexchange.com). There are numerous questions with clear and practical answers. I also looked at the ESRI developer board and posted one question there, but have yet to receive a response. My question, how do I write an extension that will connect my G drive when ArcMap opens? It's only a couple of clicks, but I would love to avoid that particular quirk of the USC GIS server. Unfortunately, the mechanism doesn't appear to exist right now, and the best solution is to copy a binary file that contains the connection data, which is undesirable.

One final improvement that I would make would be to add more analysis and visualization functions. The final GIS should allow the user to query and see the results graphically. A simple way to do this would be symbology on the map, though I do believe that this should ultimately be a web application. The final GIS should have additional functions to retrieve data such as precipitation and immediate weather.

Finally, while outside of the scope of this project, there is an option to download radar graphics, and I am really curious about whether or not they can be loaded into a GIS and georeferenced automatically. The request includes the 4 boundaries, which should be the same as the map extent. This suggests that aligning the corners of the map and raster would produce useful results. Sadly, I have not yet had time to play with this.

---

<sup>3</sup> <http://resources.arcgis.com/en/help/main/10.2/index.html#//018w00000014000000>

## References

Weather Underground API

<http://www.wunderground.com/weather/api/>

Python HTTPLIB

<https://docs.python.org/2/library/httpplib.html>

Python JSON

<https://docs.python.org/2/library/json.html>

David W. allen, Getting to know ArcGIS Modelbuilder (Redlands: ESRI Press, 2011)

Paul A. Zanbergen, Python Scripting for ArcGIS (Redlands: ESRI Press, 2013)

The World Database of Large Urban Areas (2009) [downloaded shapefile]. Nordpil, Stockholm University, Stockholm, Sweden. URL: <https://nordpil.com/resources/world-database-of-large-cities/> [November 2014]

## Acknowledgements

I would like to thank Jennifer Swift of the USC Spatial Sciences Institute for her patience and guidance throughout this project. I would not be where I am today without her assistance. I would like to thank my colleagues David White, Matthew Kline, and Kristina Furukawa for their valuable feedback throughout the course. Finally, I would like to thank my wife, Mi Kyung Kim, for her patience and support throughout this course.

## About the Author

Charles Hall is a graduate student in the USC Spatial Sciences Institute. He received his BA in history from UCSD and has worked as both a programmer and system administrator for nearly twenty years. He is currently working to develop his GIS skillset and integrate it with his existing technical skills. He is very interested in GIS application in the humanities and social sciences including archaeology, food, and history.

For more information please contact:

Charles Hall

Programmer / Analyst III - USC Viterbi School of Engineering

Graduate Student - USC Spatial Sciences Institute

[charlemh@usc.edu](mailto:charlemh@usc.edu)



## Index of Figures

1. The World Database of Large Urban Areas
2. Python replace function
3. Aggregated city names
4. Python split function
5. Check for field existence and create if absent
6. Simple GET request test
7. Update cursor with where clause
8. HTTP GET request to Weather Underground API
9. GPToolDialog from add-in code
10. Flow of control in program
11. Development cycle
12. Add-in executable (Installation instructions)
13. Attribute table prior to tool execution
14. Input parameter example
15. Script dialog window with results
16. Updated attribute table