An evaluation of web mapping functionality for simple quantitative hydrologic analysis

by Madeline Merck
a Term Project for
CEE 6440 GIS in Water Resources
December 5, 2014

Introduction

Background

HydroViz is a web-based educational system designed to support active learning in the field of Hydrology [1]. The purpose of HydroViz is to explore various hydrologic concepts in the context of solving real world problems. It achieves this through case-based, data- and simulation-driven learning experiences in the form of modules using data, models and analysis. One of the modules currently under development is based on a case study of the Great Salt Lake (GSL) Basin located in Utah, Idaho, and Wyoming. In this module, students will explore the water balance of a closed basin and characterize interactions involving inputs, outputs, bathymetry, elevation, and salinity in the functioning of the system. Topics in the module may include: examination of precipitation-elevation relationships; runoff ratios; changes in lake volume in comparison to inputs; comparison of lake evaporation calculated from mass-balance analysis and meteorological-based estimates; and dilution, salt mass-balance and salinity-evaporation relationships through examination of the lake salinity in relation to its volume.

Project Objectives

In keeping with the web-based platform of HydroViz, there is a need to evaluate whether the exploration of the topics included in the GSL module can be facilitated through an easily accessible and free-to-users online application for geographic information systems (GIS). To achieve this, the necessary infrastructure has been established, including a GIS server that is accessible through a simple web map interface. The locally hosted GIS server provides services including maps, datasets, and geoprocessing and analysis tools that were developed in support of a HydroViz GSL module pilot study. This paper describes the process of establishing the necessary infrastructure (GIS server and web map interface), processing various datasets for use as project map layers, and developing the geoprocessing and analysis tools to be accessed through the online application for GIS in support of the pilot study. Concluding remarks will include evaluation of the online application for GIS.

Deliverables

The products resulting from this work are: 1) a functioning GIS server; 2) a web map interface to access the services available on the GIS server; 3) a project map made available on the GIS server; and 4) customized geoprocessing and analysis tools made available on the GIS server.

Methods

Infrastructure: Server & Web Map Interface

Various options for the necessary infrastructure were investigated, including: the use of maps and applications available through arcgis.com (AGOL) [2], in which case it would not have been necessary to establish a server or develop a web map interface; QGIS, a free and open source GIS software [3]; and ArcGIS for Server, which has all the capabilities of ArcGIS for Desktop and that of a server. Although using AGOL would have been a simple design approach due to existing web services and mapping applications, unfortunately the functionality of ESRI's 'maps and apps' is not yet sufficient to support the needs of HydroViz. And, although QGIS is an attractive option because it is free and easily accessible, using QGIS would still have required the use of a server and development of a web map interface to store and access the necessary data and tools. Therefore, ArcGIS for Server was chosen as an all-in-one software for establishing the server and managing the web map interface.

The GIS server was established on a virtual machine located at Utah State University. The virtual machine was first configured with Windows Server 2008 R2 operating system [4]. Then ArcGIS Enterprise for Server and ArcGIS Web Adaptor 10.2.2 were installed and configured using the methods outlined in ESRI's ArcGIS resources [5]. The web map interface was developed using the ArcGIS API for JavaScript available through ArcGIS for Developers [6]. The format of the web map interface is based on several ESRI templates (also available through ArcGIS for Developers), which have been modified to fulfill the basic needs of this pilot project. The JavaScript code for the web map interface was constructed using the editor available through ESRI's ArcGIS API for JavaScript Sandbox (Figure 1). The template-based interface has evolved as needed with the construction of project map layers and tools, however the goal has remained to keep it simple.

Processing National GIS Datasets

Static project map layers are necessary inputs to the geoprocessing and analysis tools. These layers were derived from several of the datasets typically accessed through ESRI's GIS servers, including the National Hydrography Dataset (NHD) [7], the National Elevation Dataset (NED) [8], and the Watershed Boundary Dataset (WBD) [9]. Due to the cumbersome size of these datasets, downloading and processing times can be prohibitively long. Therefore, these datasets were preprocessed: each was downloaded and clipped to the HydroViz GSL areas of interest, which include the Bear River, Weber River, and Jordan River watersheds. Once processed, the data were published on the GID server as map layers.

Developing Tools

Three tools were developed in support of the HydroViz pilot study: *getWebData*, *getAnnuals*, and *Delineate*. Once the tools were developed in ArcGIS for Desktop, all three were published to the server (Figure 2). Python code for each of the tools is included in Appendix B.

getWebData

An often difficult task for both students and researchers alike includes not only acquiring data from various sources but also assembling the data in a common location for analysis or processing. Furthermore, time series data typical of hydrologic research can require frequent updates of large files with thousands of data values. There is also the issue of storing and organizing these types of large datasets. The *getWebData* tool serves as an example of an approach to data acquisition and use through web services. This approach alleviates the need to manually access various data sources and store the datasets, and can also makes fast work of updating time series datasets. In addition, because the datasets are translated into shapefiles within the tool, measurement locations (e.g., gage, station, or grid point) and spatial patterns in the data can be easily visualized.

The three data types accessed through the *getWebData* tool are snow water equivalent (SWE), precipitation, and streamflow. Each is acquired using a different type of web service and retrieved in a different format type. The SWE are annual peak Snow Telemetry (SNOTEL) data that are accessed using Simple Object Access Protocol (SOAP) web services through the Natural Resources Conservation Service's (NRCS) Air-Water Database (AWDB) web service [10]. Data and metadata are called using getStations, getStationMetadata, and getPeakData methods and are returned in XML format. The precipitation data are gridded (4 km x 4 km) estimates of annual mean that are accessed through the National Weather Service's (NWS) Advanced Hydrologic Prediction Service (AHPS) File Transfer Protocol (FTP) server [11]. These data are returned as gzip compressed tar archives that must be unzipped and extracted before the shapefiles contained within can be accessed. The streamflow data are US Geological Survey (USGS) annual mean gage measurements that are accessed using REpresentation State Transfer (REST) web services through the USGS's Beta Statistics web service [12]. These data are returned as tab-delimited text. All three data types are access and processed within the Python code when the *getWebData* tool is run.

The *getWebData* tool was developed in Python using the ArcPy package [13]. As with all the tools, it is designed to be run in the web map interface but can also be run in ArcGIS for Desktop. It has four user inputs: Watershed, Data Type, Water Year, and a file name and path for the output shapefile of the acquired data. To simplify the tool, focus its use, and help direct the students, some of the user inputs (Watershed, Data Type, and Water Year) are dropdown menu options in the tool popup window. The Watershed options are: Bear, Weber, and Jordan River watersheds. The Data Type options are: Snow Water Equivalent, Precipitation, and Streamflow. And the Water Years span from 2006 to 2014, which were chosen so that a result for each data type is available for each water year and watershed. Once the *getWebData* tool is run, the resulting output shapefile is automatically added to the map as a data layer and becomes available for use in other tools available on the server.

getAnnuals

The *getAnnuals* tool is simple tool used to calculate the annual volume of SWE, precipitation, and streamflow for a given watershed and water year. This tool uses the output from the *getWebData* tool as inputs along with a user specified gage (within the streamflow data) and a DEM of the watershed. The volume of SWE is calculated by first establishing an elevation-SWE relationship (through linear regression) using all SNOTEL

station data; the relationship is then used to estimate the SWE throughout the watershed based on the input DEM. The volume of precipitation is calculated using the mean precipitation of the entire input grid. The volume of streamflow is calculated using the annual mean value returned by the *getWebData* tool of the user specified gage. All volumes are based on the area of the input DEM. The resulting volumes are displayed in a results popup window.

Delineate

The purpose of the *Delineate* tool is to delineate a watershed from a user specified point within the NHD network [2]. This tool makes use of the Point Indexing and Navigation Delineation services accessible through the Environmental Protection Agency's (EPA) Watershed Assessment, Tracking & Environmental Results System (WATERS) Hypertext Transfer Protocal (HTTP) web service [14]. The input is simply a user specified point and a file name and path for the output shapefile of the delineated watershed. The output shapefile is automatically added to the map as a data layer.

Results

Infrastructure: Server & Web Map Interface

The web map interface (Figure 3) is a simple basemap with the following functionality: zoom, basemap toggle, location search, and a legend with two dynamic panes. The zoom function is a simple in/out zoom and the basemap toggle function switches between a topographic basemap and a satellite image. The search function allows students unfamiliar with the GSL to orient themselves and to search for points of interest within the study area. The legend has one pane for active layers and another pane for the *getWebData*, *getAnnuals*, and *Delineate* tools. JavaScript code for the web map interface is included in Appendix C.

Tools

The *getWebData*, *getAnnuals*, and *Delineate* tools (Figures 4-6, respectively) have all been published to the toolbox folder on the GIS server and are now available through REST services. However, the tools are not available on the web map interface at this time as the JavaScript code necessary to call each tool service has not been successfully completed. If the web map interface were up and running as planned, then the tool buttons in the legend pane would be active. When a tool is run through the web map interface, the inputs are chosen through dropdown menus as they would be if run in ArcGIS for Desktop. These inputs (or parameters) are then passed from the web map interface (JavaScript code) to the server where the chosen tool is run (Python code). Upon successful completion of the tool, the result is passed back to the web map interface (JavaScript code) and, depending on which tool is run, a layer is added to the web map or a popup window displays the result. At present, the tools are failing when run through the web map interface, resulting in an "esriJobFailed" error message. Although the tools cannot be run successfully through the web map interface at this time, they can be run through ArcGIS for Desktop.

Discussion & Conclusion

The purpose of this project was to evaluate whether the exploration of the topics included in the GSL module can be facilitated through an easily accessible and free-to-users online application for GIS. This was achieved by establishing the necessary infrastructure, including a GIS server that is accessible through a web map interface, and providing the necessary data and tools, including geoprocessing and analysis tools and map data layers in support of a HydroViz GSL module pilot study.

The functionality of the combination of ArcGIS for Server, ArcGIS Web Adaptor, and the web map interface is very well suited to the needs of HydroViz. ArcGIS for Server has all the capabilities of ArcGIS for Desktop but with the option of tailoring custom tools to the needs and abilities of the students and to make them available at no cost to the student. In addition, the web map interface is completely customizable giving the designer infinite freedom. However, there is an almost prohibitively steep learning curve associated with the development of this type of an application. That said, the fact that the online application for GIS developed for this project was done so by someone previously unfamiliar with ArcGIS, Python, and JavaScript suggests that it is not an impossible task.

Accessing data through web services within the geoprocessing and analysis tools is not only an effective method for reproducibility but it also puts the onus of storing and organizing the data on the responsible agency. Also, accessing services such as those used in the *Delineate* tool save modeling and processing time. However, there are costs associated with using these types of services: each one is different and therefore requires different knowledge and skills. In addition, there is no guarantee the web address for the service or the necessary retrieval method will remain consistent over time, requiring upkeep of the various services on the GIS server. Furthermore, there is no way to debug or fix errors in the web service; for example, possibly like those errors visible in the *Delineate* tool (Figure 6).

If the online application for GIS developed for the HydroViz pilot study is further pursued, there are a few recommendations that should be considered. First, the design of the system should be performed by someone not only familiar with hydrologic concepts but also at least a working knowledge of various GIS software. Second, the tools and web map interface should be developed by a computer software engineer well versed in not only web design and development but also the various ArcGIS APIs, as there are several. And finally, careful consideration should be taken when selecting the data used in the various tools and data layers as none are without flaws.

References

[1] HydroViz. http://www.hydroviz.org *accessed 5 December 2014.*

[2] ArcGIS Online. http://arcgis.com *accessed 5 December 2014.*

[3] QGIS. http://qgis.org *accessed 5 December 2014.*

[4] Microsoft, "Windows Server 2008 R2". http://www.microsoft.com/en-us/server-cloud/products/windows-server-2012-r2/ *accessed 5 December 2014.*

[5] ArcGIS Resources. "ArcGIS for Server (Windows)." http://resources.arcgis.com/en/help/main/10.2/index.html#//0154000002np000000 *accessed 5 December 2014.*

[6] ArcGIS for Developers. "ArcGIS API for JavaScript." https://developers.arcgis.com/javascript/ accessed 5 December 2014.

[7] US Geological Survey. "The National Map." http://nhd.usgs.gov/ *accessed 5 December 2014.*

[8] US Geological Survey. "National Elevation Dataset." http://ned.usgs.gov/ *5 December 2014.*

[9] Natural Resources Conservation Service. "Snow Telemetry (SNOTEL) and Snow Course Data and Products." http://www.wcc.nrcs.usda.gov/snow *accessed 5 December 2014.*

[10] National Weather Service. "Advanced Hydrologic Prediction Service." http://water.weather.gov/ahps *accessed 5 December 2014.*

[11] US Geological Survey. "REST Web Services." http://waterservices.usgs.gov/rest accessed 5 December 2014.

[12] ArcGIS Resources. "ArcPy." http://resources.arcgis.com/en/help/main/10.1/index.html#//000v000000v7000000 *accessed 5 December 2014.*

[13] EPA Water. "EPA WATERS Web Services." http://water.epa.gov/scitech/datait/tools/waters/services/index.cfma*, accessed 5 December 2014*.

Appendix A: Figures

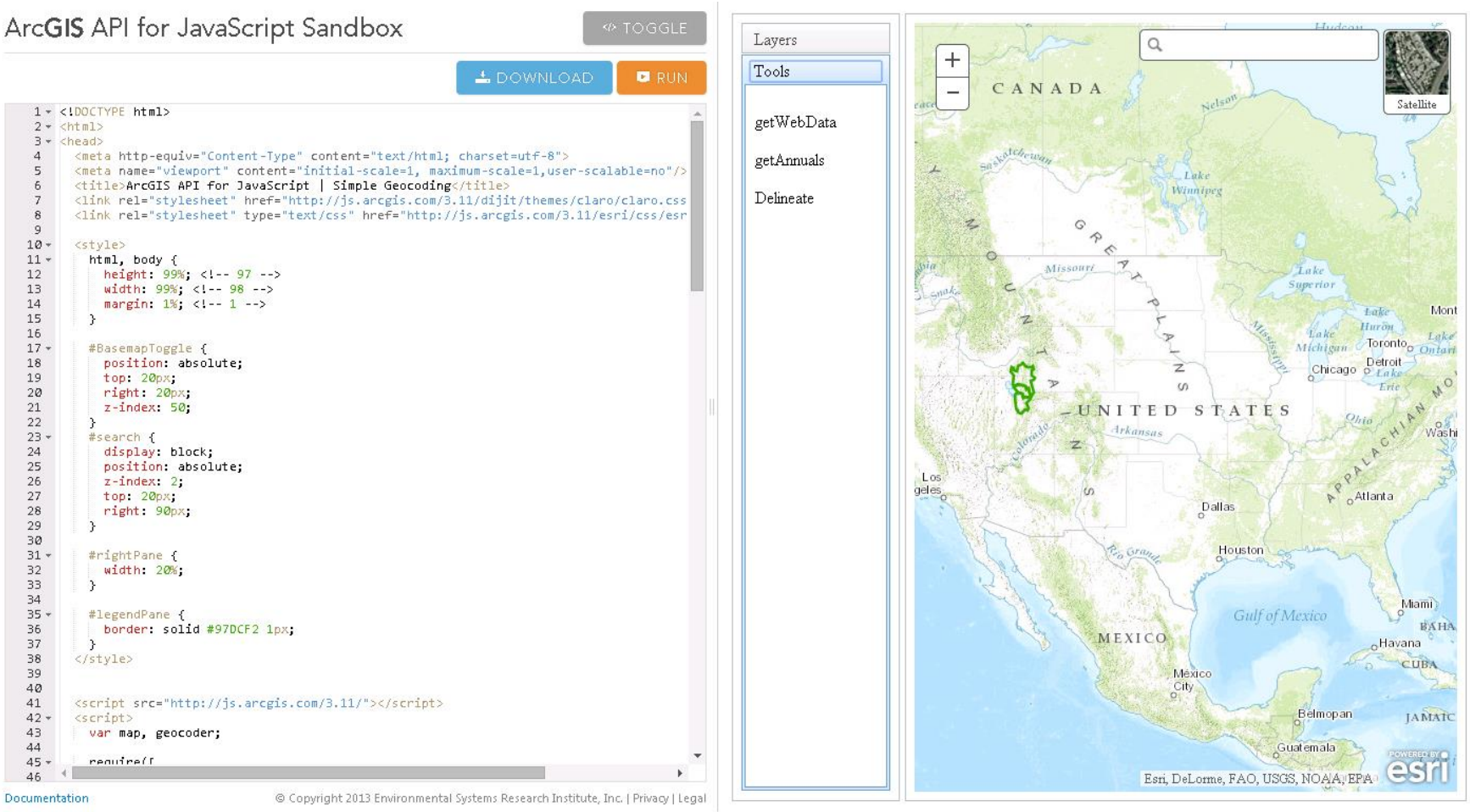
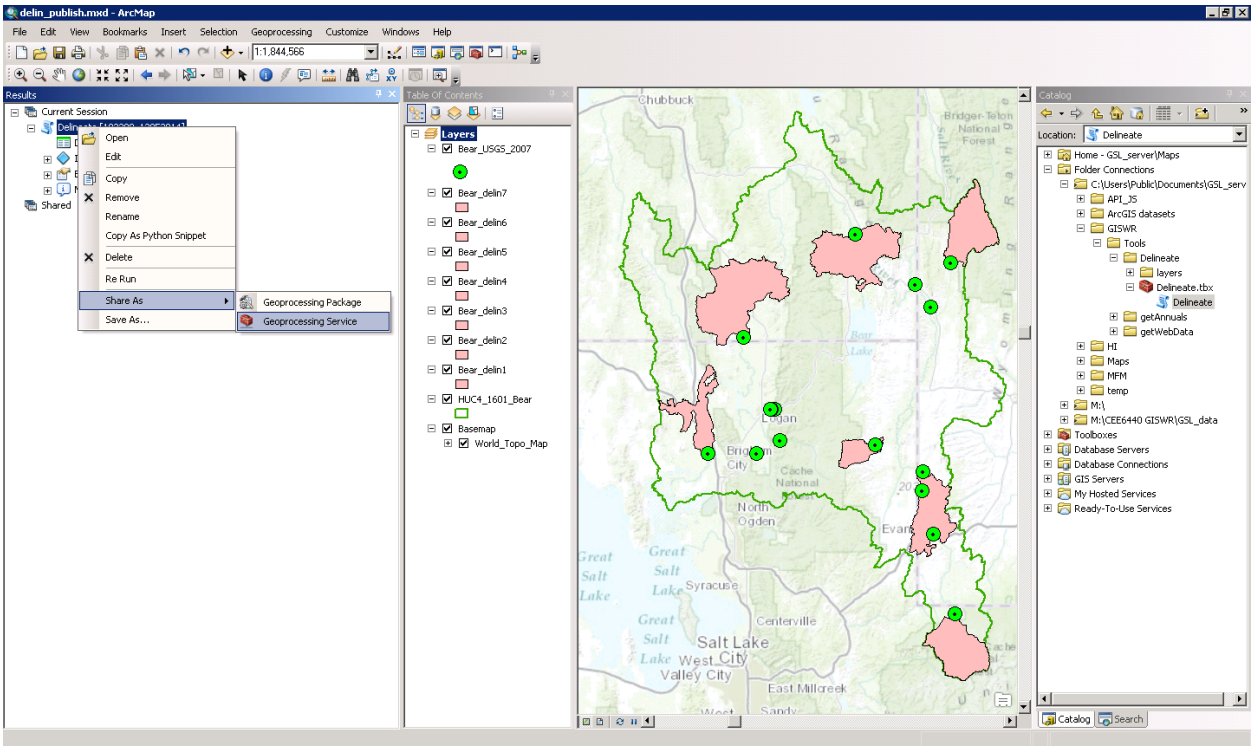


Figure 1: A screenshot of the process of developing the web map interface using the ArcGIS API for JavaScript Sandbox available through ArcGIS for Developers. The left pane is a JavaScript editor and the right pane is the resulting web page. The right pane in this figure is also the web map interface developed for this project.



ArcGIS REST Services Directory Login | Get Token

Home > services > GISWR_Tools Help | API Reference

JSON | SOAP

Folder: GISWR_Tools

Current Version: 10.22

View Footprints In: ArcGIS.com Map

Services:

- GISWR_Tools/Delineate (GPServer)
- GISWR_Tools/getAnnuals (GPServer)
- GISWR_Tools/getWebData (GPServer)

Supported Interfaces: REST SOAP Sitemap Geo Sitemap

Figure 2: A screenshot of the process of publishing a tool (top) and the REST web service page showing the three published tools (bottom).

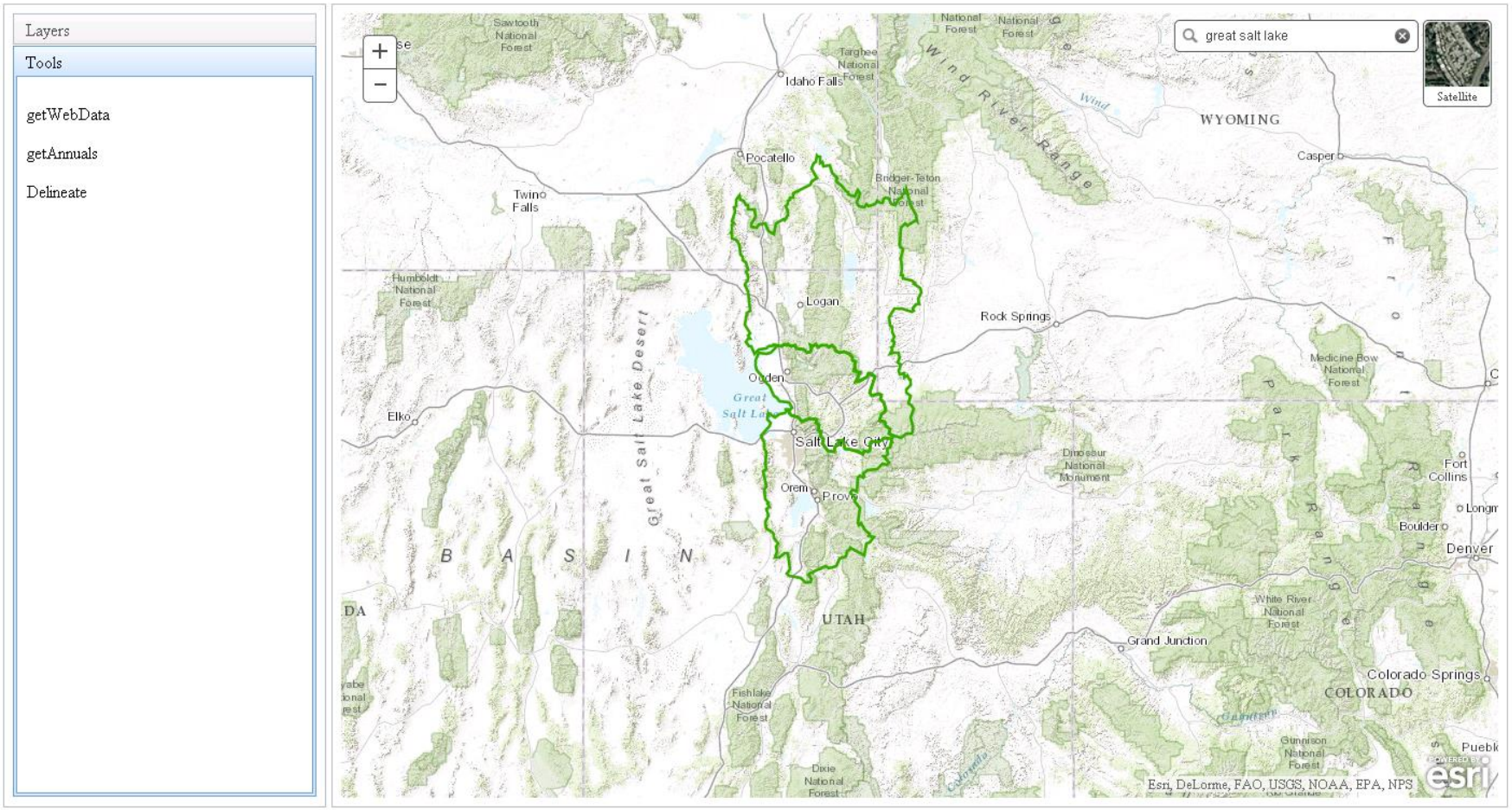


Figure 3: A screenshot of the web map interface. At left is the legend with two panes, one for active layers (not shown) and one for available tools. At upper left is the zoom function. At upper right is the search function and basemap toggle function.

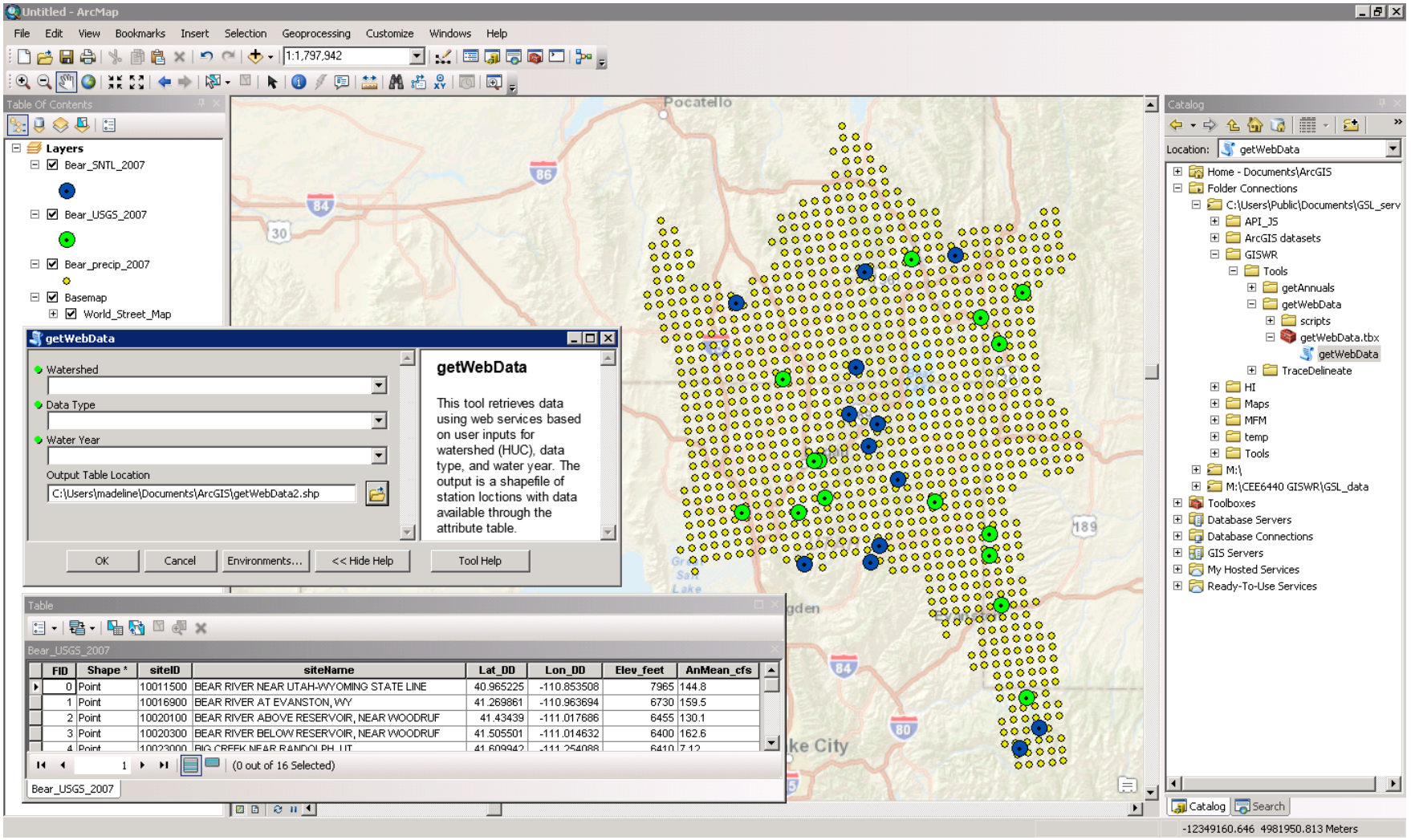


Figure 4: A screenshot of the *getWebData* tool. The getWebData popup window includes the following user inputs: Watershed, Data Type, Water Year, and Output location. The attribute table below shows the attributes of the acquired streamflow data shapefile. The map is populated with SWE, precipitation, and streamflow for the Bear River watershed for the 2007 water year.

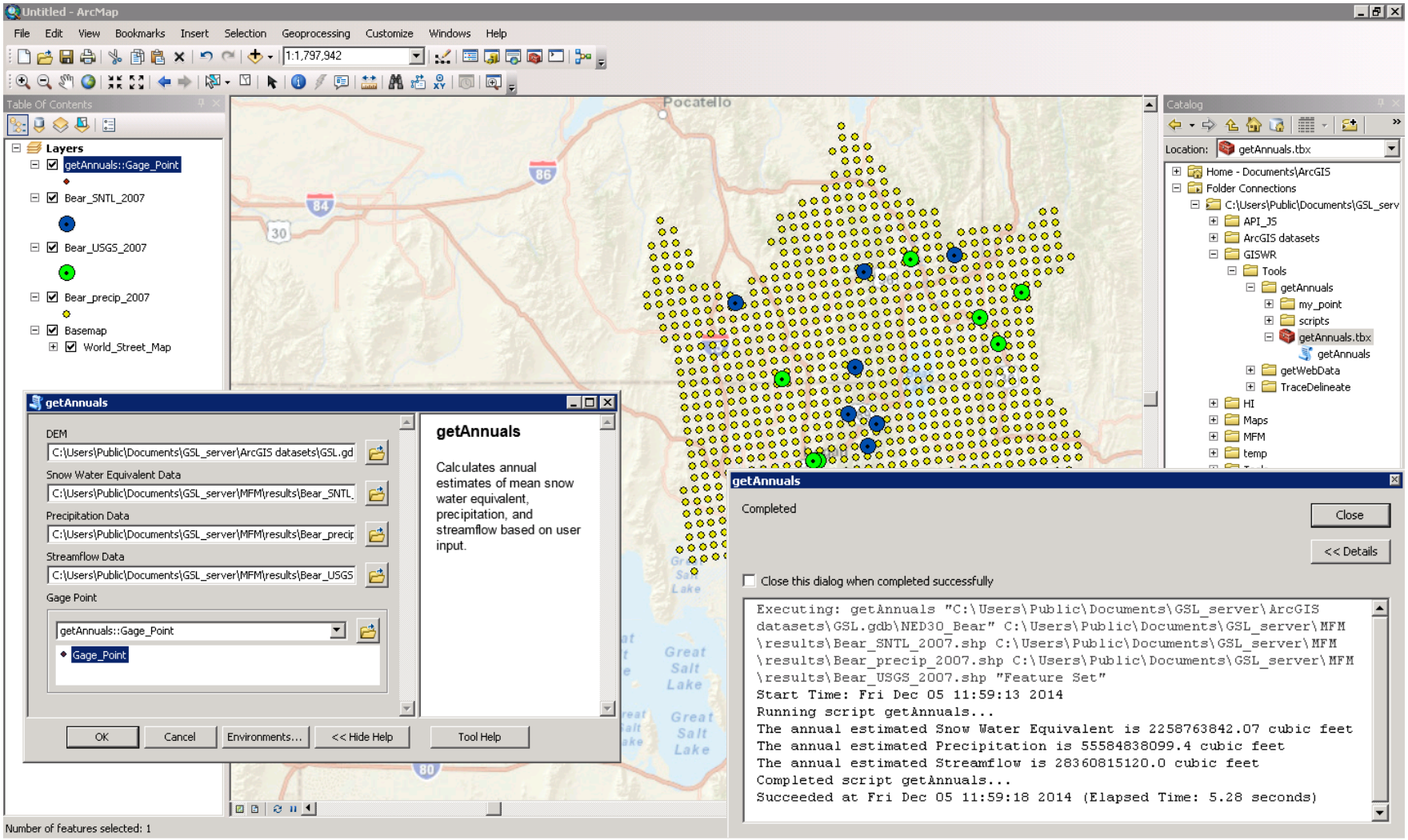


Figure 5: A screenshot of the *getAnnuals* tool. The getAnnuals popup window at left includes the following user inputs: DEM, Snow Water Equivalent Data, Precipitation Data, Streamflow Data, and Gage Point. The getAnnuals window at right is the superimposed results popup window showing the resulting annual volumes.

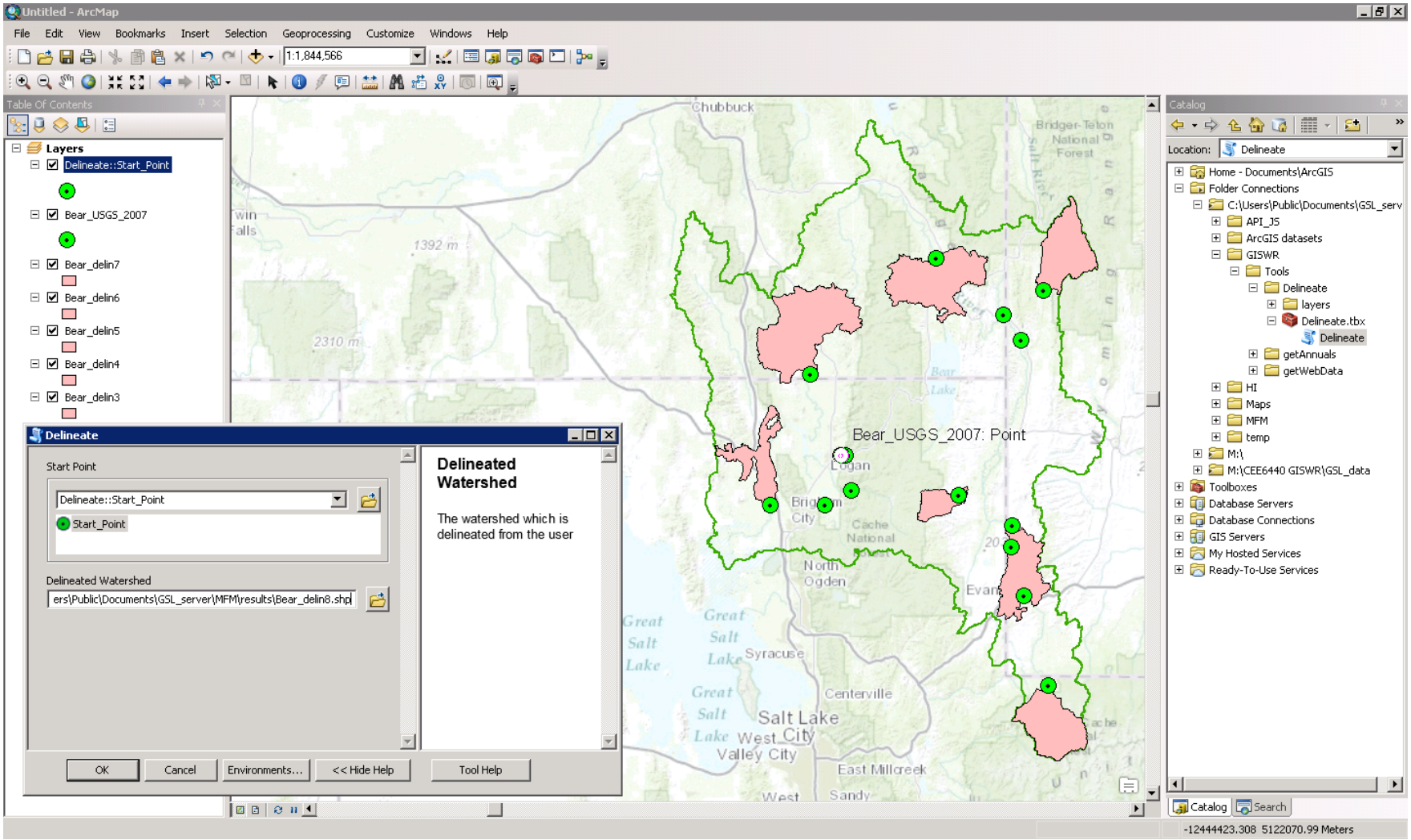


Figure 6: A screenshot of the *Delineate* tool. The Delineate popup window at left includes the following user inputs: Start Point and Delineated Watershed output location. The map is populated with USGS streamflow gage data from which various subwatersheds were attempted to be delineated. It is obvious that this tool is not always accurate. However, it has not yet been established whether this is a limitation of the web service (due to large delineated areas) or of the python code within the tool.

Appendix B: Python Code for Tools

getWebData

```
#############################################################################
#
#This script uses web services to access data and metadata and then
#creates shapefiles of the measurement stations with annual data.
#Precipitation uses NWS to access RFC estimates (data = annual mean rainfall).
#Snow Water Equivalent uses NRCS to access SNOTEL (data = annual peak SWE).
#Streamflow uses USGS to access statistics of gage measurements (data = annual mean flow).
#
#Help with the various NWS call options (ftp hosted service):
#http://water.weather.gov/precip/download.php
#
#Help with the various NRCS call options and WSDL (SOAP web services):
#http://www.wcc.nrcs.usda.gov/web_service/AWDB_Web_Service_Reference.htm
#http://www.wcc.nrcs.usda.gov/awdbWebService/services?WSDL
#
#Help with the various USGS call options (REST web services:
#http://waterservices.usgs.gov/rest/
#
#############################################################################

#### Import necessary modules...
## for use outside of ArcGIS and when debugging...
#import sys
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\arcpy')
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\bin')
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\ArcToolbox\\Scripts')

import arcpy
from arcpy import env
from arcpy.sa import *
from suds.client import Client
import numpy
import json
import os
import urllib
import urllib2
import tarfile

#############################################################################

# allow overwriting of output files (shapefiles in particular)
arcpy.env.overwriteOutput = True

## hardcoding for use outside of ArcGIS and when debugging...
#huc = 1601             #1601=Bear, 160201=Weber, 160202=Jordan
#ws = 'Bear'
#waterYear = 2006
#elementCd = 'WTEQ' #
```

```
#networkCd = 'USGS' # NWS, SNTL, USGS
#outPath = r'C:\Users\...\dataShapeTEST.shp' # change path as necessary

#### Get script inputs...
# These are the user inputs that will be seen in the ArcGIS tool
# popup window when the tool is opened.

# get HUC (0): string
ws = arcpy.GetParameterAsText(0) # Bear, Weber, Jordan --> 1601, 160201, 160202
arcpy.AddMessage(ws)
wss = {"Bear":1601,"Weber":160201,"Jordan":160202}
huc = wss[ws]

# get data type (1): string
dType = arcpy.GetParameterAsText(1)
arcpy.AddMessage(dType)
networks = {"Snow Water Equivalent":"SNTL","Precipitation":"NWS","Streamflow":"USGS"}
networkCd = networks[dType]

# get water year (2): string
waterYear = int(arcpy.GetParameterAsText(2))
arcpy.AddMessage(waterYear)

# get output path for table of long, lat, name (3): shapefile
outPath = arcpy.GetParameterAsText(3)
arcpy.AddMessage(outPath)

########################################################################

##### PRECIPITATION #####
if networkCd == 'NWS':

   # download and unzip precipitation shapefile (tar.gz) from
   # http://water.weather.gov/precip/about.php

   os.chdir(str(r"C:\Users\Public\Documents\GSL_server\MFM\results\junk"))

   urllib.urlretrieve("http://water.weather.gov/precip/p_download_new/"+ \
   str(waterYear) +"/10/01/nws_precip_wateryear2date_observed_shape_"+ \
   str(waterYear) +"1001.tar.gz", "nws_precip_wateryear2date_observed_shape_"+ \
   str(waterYear) +"1001.tar.gz")
   tar = tarfile.open("nws_precip_wateryear2date_observed_shape_"+ str(waterYear) +"1001.tar.gz")
   tar.extractall()
   tar.close()
   precip_all = str("nws_precip_wateryear2date_observed_" + str(waterYear) + "1001.shp")

   # clip precipitation shapefile for each huc
   hucRef = {"Bear":"HUC4_1601_Bear","Weber":"HUC6_160201_Weber","Jordan":"HUC6_160202_Jordan"}
   hucShapeName = hucRef[ws]
   hucShape = str(r"C:\Users\Public\Documents\GSL_server\ArcGIS datasets\GSL.gdb/" \
   + str(hucShapeName))
   arcpy.Clip_analysis(precip_all, hucShape, outPath)
   arcpy.MakeFeatureLayer_management(outPath,outPath)
```

```
##### SNOTEL #####
else:
   if networkCd == 'SNTL':

      # set data constraints
      stationId = []
      duration = 'DAILY'    # getPeakData only supports daily duration at this point.
      ordinal = 1           # default
      heightDepth = None     # default
      getFlags = True        # data flags
      logicalAnd = True
      alwaysReturnDailyFeb29 = False

      # define wsdl URL (used for both SNOTEL and USGS) *** SOAP web services ***
      NRCS = Client('http://www.wcc.nrcs.usda.gov/awdbWebService/services?WSDL')

      # -> getStations: *** SOAP web services ***
      # returns: a list of strings that are the stationTriplets for
      # the stations meeting the criteria for SNOTEL or USGS sites.
      stations = NRCS.service.getStations(
         stationIds = stationId,
         hucs = huc,
         networkCds = networkCd,
         logicalAnd = logicalAnd
         )
      numberofstations = len(stations)

      # -> getStationMetadataMultiple *** SOAP web services ***
      # returns: station name, lat, long, elevation and other information
      stationsMetadata = NRCS.service.getStationMetadataMultiple(
         stationTriplets = stations
         )

      # -> getPeakData *** SOAP web services ***
      # returns: peak SWE data values for each water year
      elementCd = 'WTEQ'
      data = NRCS.service.getPeakData(
         stationTriplets = stations,
         elementCd = elementCd,
         ordinal = ordinal,
         heightDepth = heightDepth,
         duration = duration,
         getFlags = getFlags,
         beginYear = waterYear,
         endYear = waterYear,
         )

      # initialize lists for data acquisition
      stationData = []
      index = 0

      # create array of information needed to create shapefile with attribute table
```

```
    for dataset in data:
        if 'values' in dataset:
            a = stations[index]    # site number
            b = stationsMetadata[index]['longitude']    # longitude
            c = stationsMetadata[index]['latitude']    # latitude
            d = stationsMetadata[index]['name']    # station name
            e = stationsMetadata[index]['elevation']    # elevation
            f = dataset['values'][0]    # data value (peak annual SWE, inches)
            stationData.append((a, (b,c), d,c,b,e,f))
        index += 1

    # define headers and data format
    dtype=[('siteID', (str, 40)), ('lonlat', '<f8', 2), ('siteName', (str, 40)),\
    ('Lat_DD', '<f8'),('Lon_DD','<f8'),('Elev_meter','<f8'),('AnPeak_in', (str, 40))]

##### USGS #####
  elif networkCd == 'USGS':

    # initialize lists for data acquisition
    huc8s = {"1601":"16010101,16010102,16010201,16010202,16010203,16010204","160201":"16020101,\
    16020102","160202":"160201,160202,160203,160204"}
    huc8 = huc8s[str(huc)]
    stationData = []
    index = 0

    # get stations in HUC
    stationURL = "http://waterservices.usgs.gov/nwis/site/?format=rdb&huc=" + huc8 + \
    "&siteStatus=active&hasDataTypeCd=ad"
    stationPage = urllib2.urlopen(stationURL).read()
    stationsplit = stationPage.split('\n') #splits page by lines

    # get
    for line in range(0,len(stationsplit)):
        tab1 = stationsplit[line].split('\t') #splits the last line by tabs
        if tab1[0] == "USGS":
            #station = tab1[1]
            usgsURL = "http://waterservices.usgs.gov/nwis/stat/?format=rdb&sites=" \
            + str(tab1[1]) + "&startDT=" + str(waterYear-1) + "&endDT=" + str(waterYear) + \
            "&parameterCd=00060&statReportType="\
            +"annual&statTypeCd=mean&statYearType=water&missingData=off"
            usgsPage = urllib2.urlopen(usgsURL).read()
            tab2 = usgsPage.split('\n')[-2].split('\t') #splits the last line by tabs
            if tab2[0] == 'USGS':
                a = tab1[1]    # site number
                b = tab1[5]    # longitude
                c = tab1[4]    # latitude
                d = tab1[2]    # station Names
                e = tab1[8]    # elevation
                f = tab2[-1]    # data value (mean annual flow)
                stationData.append((a, (b,c), d,c,b,e,f))
                index += 1

    # define headers and data format
```

```
    dtype=[('siteID', (str, 40)), ('lonlat', '<f8', 2), ('siteName', (str, 40)),\
    ('Lat_DD', '<f8'),('Lon_DD','<f8'),('Elev_feet','<f8'),('AnMean_cfs', (str, 40))]

  ##### SNOTEL or USGS #####
  # create numpy array of data and metadat for shapefile and attribute table
  in_array = numpy.array(stationData, dtype=dtype)
  SR = arcpy.SpatialReference("NAD 1983")

  # create shapefile and add to the map
  arcpy.da.NumPyArrayToFeatureClass(in_array, outPath, ['lonlat'], SR)
```

getAnnuals

```
################################################################################
#
# This script calculates annual average volumes for snow water equivalent,
# precipitation, and streamflow based on user inputs. A SWE-elevation relationship
# is calculated using linear regression based on input SWE data and input DEM.
# The relationship is used to estimate volume for the watershed based on the area
# of the DEM. Average annual precipitation depth is calculated for the input data
# and yearly volume for the watershed is based on the area of the DEM. Annual
# streamflow is # based on annual mean as imported from the getWebData tool.
#
################################################################################

##### Import necessary modules...
# for use outside of ArcGIS and when debugging...
#import sys
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\arcpy')
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\bin')
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\ArcToolbox\\Scripts')#

import arcpy
from arcpy import env
from arcpy.sa import *
from suds.client import Client
import json
import numpy as np

##### Get script arguments...
# These are the user inputs that will be seen in the tool
# dialog window when the tool is run... just brainstorming...

# Allow files to be overwritten
env.overwriteOutput = True

dem = arcpy.GetParameterAsText(0)
fileSWE = arcpy.GetParameterAsText(1)
filePrecip = arcpy.GetParameterAsText(2)
fileUSGS = arcpy.GetParameterAsText(3)
pointUSGS = arcpy.GetParameter(4)
#if pointUSGS == '#' or not pointUSGS:
```

```
#           pointUSGS = "in_memory/{87AF799A-1608-483B-9022-3AA58EFEF329}"

### For testing...
#dem = r'C:\Users\Public\Documents\GSL_server\ArcGIS datasets\GSL.gdb\NED30_Bear'
#fileSWE = r'C:\Users\Public\Documents\GSL_server\MFM\results\Bear_SNTL_2010.shp'
#filePrecip = r'C:\Users\Public\Documents\GSL_server\ArcGIS
datasets\NWS_precip\huc_precip\precip_Bear_2010.shp'
#fileUSGS = r'C:\Users\Public\Documents\GSL_server\MFM\results\Bear_USGS_2010.shp'
#outpath = r'C:\Users\Public\Documents\GSL_server\MFM\results\results.csv'

##### SWE #####

# convert rasters to arrays and get raster metadata
dem_Raster = arcpy.Raster(dem)
dem_array = arcpy.RasterToNumPyArray(dem, nodata_to_value=0)
lowerLeft = dem_Raster.extent.lowerLeft
x_cell_size = dem_Raster.meanCellWidth
y_cell_size = dem_Raster.meanCellHeight

# Create the search cursor
rows = arcpy.SearchCursor(fileSWE)

# Call SearchCursor.next() to read the first row
row = rows.next()
elevation = []
dataSWE = []

# Start a loop that will exit when there are no more rows available
while row:

   # build the data arrays
   elevation.append(row.Elev_meter)
   dataSWE.append(float(row.AnPeak_in))

   # Call SearchCursor.next() to move on to the next row
   row = rows.next()

# Translate to np.array (DEM is in meters, need data in feet)
peakSWE = np.array(dataSWE)                  # inches
elevation = np.array(elevation)                  # feet
x_grid = x_cell_size / 0.3048            # feet
y_grid = y_cell_size / 0.3048            # feet

# Run the regression to get the slope and intercept...
regression = np.polyfit(elevation, peakSWE, 1)

# Calculate the annual SWE volume in cubic feet
SWE_array = regression[0]*dem_array + regression[1]
clipSWE = SWE_array.clip(0)
meanSWE = np.mean(clipSWE[np.nonzero(clipSWE)])
countSWE = np.count_nonzero(clipSWE)
rasterArea = countSWE * x_grid * y_grid          # square feet
annualSWE = rasterArea * meanSWE / 12    # cubic feet
```

```
arcpy.AddMessage('The annual estimated Snow Water Equivalent is ' + str(annualSWE) + ' cubic feet')

##### Precip #####

# Create the search cursor
rows = arcpy.SearchCursor(filePrecip)

# Call SearchCursor.next() to read the first row
row = rows.next()
dataPrecip = []

# Start a loop that will exit when there are no more rows available
while row:

   # build the data arrays
   dataPrecip.append(float(row.Globvalue))

   # Call SearchCursor.next() to move on to the next row
   row = rows.next()

meanPrecip = np.mean(dataPrecip)              # inches
annualPrecip = rasterArea * meanPrecip / 12    # cubic feet
arcpy.AddMessage('The annual estimated Precipitation is ' + str(annualPrecip) + ' cubic feet')

##### USGS #####

# snap inputPoint to user input fileUSGS point shapefile
arcpy.Snap_edit(pointUSGS, [[fileUSGS,"VERTEX",50]])

# create feature set
f = arcpy.FeatureSet(pointUSGS)

# parse out the geometry
geom = json.loads(f.JSON)['features'][0]['geometry']
cursor = arcpy.UpdateCursor(f, "","NAD 1983")
dsc_f=arcpy.Describe(f)
for row in cursor:
   shape=row.getValue(dsc_f.shapeFieldName)
   geom = shape.getPart(0)
   lon = geom.X
   lat = geom.Y

# Create the search cursor
rows = arcpy.SearchCursor(fileUSGS)

# Call SearchCursor.next() to read the first row
row = rows.next()

# Start a loop that will exit when there are no more rows available
while row:

   # build the data arrays
   if abs(lat - row.Lat_DD) < 0.000001:
```

```
      if abs(lon - row.Lon_DD) < 0.000001:
         meanUSGS = float(row.AnMean_cfs)

   # Call SearchCursor.next() to move on to the next row
   row = rows.next()

annualUSGS = meanUSGS * (365.25 * 24 * 60 * 60)    # cfs to cu. ft. per year
arcpy.AddMessage('The annual estimated Streamflow is ' + str(annualUSGS) + ' cubic feet')
```

Delineate

```
################################################################################
#
# This script delineates a watershed in the NHD network. It uses the
# EPA WATERS Web Services, in particular the Point Indexing service
# and the Navigation Delineation service.
# These services are described here:
# http://water.epa.gov/scitech/datait/tools/waters/services/index.cfm
#
################################################################################

##### Import necessary modules...
# for use outside of ArcGIS and when debugging...
#import sys
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\arcpy')
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\bin')
#sys.path.append('C:\\Program Files\\ArcGIS\\Desktop10.2\\ArcToolbox\\Scripts')#

import arcpy
import json
import urllib2

################################################################################

#### Get script inputs..

arcpy.env.overwriteOutput = True

fs = arcpy.GetParameter(0)
#if fs == '#' or not fs:
#           fs = "in_memory/{87AF799A-1608-483B-9022-3AA58EFEF329}"

# create feature set
f = arcpy.FeatureSet(fs)

# parse out the geometry
geom = json.loads(f.JSON)['features'][0]['geometry']
cursor = arcpy.UpdateCursor(f, "","NAD 1983")
dsc_f=arcpy.Describe(f)
for row in cursor:
   shape=row.getValue(dsc_f.shapeFieldName)
   geom = shape.getPart(0)
```

```
   lon = geom.X
   lat = geom.Y
arcpy.AddMessage('Selected Point = (%5.3f,%5.3f)'% (lon, lat))

# get output path for table of long, lat, name:
output_path = arcpy.GetParameterAsText(1)
arcpy.AddMessage(output_path)

### For testing...
#lat = 41.836576 #40.995250
#lon = -112.047938 #-110.869464
#output_path = 'C:\\Users\\ ... \\output.shp'

#---------------------------------------------------------------------
# POINT LOCATION: authored by Jon Goodall (goodall@virignia.edu)

def pointIndexing(lon, lat):
   """
   Uses the EPA WATERS Web Services to identify the comid and measure
   along an NHD feature for a given lat/lon

   Parameters:
        lat: the latitude in decimal degrees of the point
        lon: the longitude in decimal degrees of the point

   Returns:
        comID, measure where measure is the fmeasure attribute
        returned by the PointIndexing service.
   """

   #build the point indexing URL
   PtServiceUrl = "http://ofmpub.epa.gov/waters10/PointIndexing.Service?" \
      + "pGeometry=POINT(%s+%s)"%(lon, lat) \
      + "&pGeometryMod=WKT%2CSRID%3D8265" \
      + "&pResolution=3" \
      + "&pPointIndexingMethod=DISTANCE" \
      + "&pPointIndexingMaxDist=25" \
      + "&pOutputPathFlag=FALSE" \
      + "&pReturnFlowlineGeomFlag=FALSE" \
      + "&optNHDPlusDataset=2.1" \
      + "&optCache=1415731048364" \
      + "&optJSONPCallback="

   #load response into JSON object
   response = json.loads(urllib2.urlopen(PtServiceUrl).read())

   #check the status message from the response to see if it worked
   status_message = response['status']['status_message']
   if status_message == "No Results Returned.":
      raise Exception('Point service did not find an NHD feature for ' + \
         'lat=%s, lon=%s. Please double check your coordinates.'%(lat, lon))

   #extract comids and measures
```

```
    comid = response['output']['ary_flowlines'][0]['comid']
    measure = response['output']['ary_flowlines'][0]['fmeasure']

    return comid, measure

#-----------------------------------------------------------------
# WATERSHED DELINEATION

# enable overwriting of output shapefile
arcpy.env.overwriteOutput = True

# get the comid and measure for the two points
comid, measure = pointIndexing(lon, lat)

# build the navigation delineation service URL
ServiceUrl = "http://ofmpub.epa.gov/waters10/NavigationDelineation.Service?" \
    + "pNavigationType=UT" \
    + "&pStartComid=%s"%(comid) \
    + "&pStartMeasure=%s"%(measure) \
    + "&optJSONPCallback="

# load response into JSON object
response = json.loads(urllib2.urlopen(ServiceUrl).read())

# select appropriate list of points
if response['output']['shape']['type'] == 'MultiPolygon':
    # parse out list of delineated coordinates
    lengths = [len(p[0]) for p in response['output']['shape']['coordinates']]
    idx = lengths.index(max(lengths))
    polygon = response['output']['shape']['coordinates'][idx][0]
else: polygon = response['output']['shape']['coordinates'][0]

# create polygon and feature layer
poly_shape = arcpy.Polygon(arcpy.Array([arcpy.Point(*coords) for coords in polygon]))
SR1 = arcpy.SpatialReference("NAD 1983")
SR2 = arcpy.SpatialReference("WGS 1984 Web Mercator (auxiliary sphere)")
tempfile = 'in_memory\\temp9'

arcpy.CopyFeatures_management(poly_shape,tempfile)
arcpy.DefineProjection_management(tempfile, SR1)
arcpy.Project_management(tempfile, output_path, SR2)

arcpy.Delete_management('in_memory')
```

Appendix C: JavaScript code for web map interface

```
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
 <meta name="viewport" content="initial-scale=1, maximum-scale=1,user-scalable=no"/>
 <title>ArcGIS API for JavaScript | Simple Geocoding</title>
 <link rel="stylesheet" href="http://js.arcgis.com/3.11/dijit/themes/claro/claro.css">
 <link rel="stylesheet" type="text/css" href="http://js.arcgis.com/3.11/esri/css/esri.css">

 <style>
  html, body {
   height: 99%; <!-- 97 -->
   width: 99%; <!-- 98 -->
   margin: 1%; <!-- 1 -->
  }

  #BasemapToggle {
   position: absolute;
   top: 20px;
   right: 20px;
   z-index: 50;
  }
  #search {
   display: block;
   position: absolute;
   z-index: 2;
   top: 20px;
   right: 90px;
  }

  #rightPane {
   width: 20%;
  }

  #legendPane {
   border: solid #97DCF2 1px;
  }
 </style>


 <script src="http://js.arcgis.com/3.11/"></script>
 <script>
  var map, geocoder;

  require([
   "esri/map",
   "esri/dijit/BasemapToggle",
   "esri/dijit/Geocoder",
   "esri/layers/FeatureLayer",
   "esri/dijit/Legend",
   "dojo/_base/array",
```

```
    "dojo/parser",
    "dijit/layout/BorderContainer",
    "dijit/layout/ContentPane",
    "dijit/layout/AccordionContainer",
    "dojo/domReady!"
  ], function(Map, BasemapToggle, Geocoder, FeatureLayer, Legend,
          arrayUtils, parser) {

    parser.parse();

    map = new Map("map",{
     basemap: "topo",
     center: [-98, 40], // lon, lat (-112.509788, 41.380117)
     zoom: 4
    });

    var toggle = new BasemapToggle({
     map: map,
     basemap: "satellite"
    }, "BasemapToggle");
    toggle.startup();

    geocoder = new Geocoder({
     map: map
    }, "search");
    geocoder.startup();


  var huc1 = new
FeatureLayer("http://madeline.uwrl.usu.edu/hydroviz/rest/services/GSL_basemap_1_0_test/MapServer/6", {
    mode: FeatureLayer.MODE_ONDEMAND,
    outFields:["*"]
  });
  var huc2 = new
FeatureLayer("http://madeline.uwrl.usu.edu/hydroviz/rest/services/GSL_basemap_1_0_test/MapServer/7", {
    mode: FeatureLayer.MODE_ONDEMAND,
    outFields:["*"]
  });
  var huc3 = new
FeatureLayer("http://madeline.uwrl.usu.edu/hydroviz/rest/services/GSL_basemap_1_0_test/MapServer/8", {
    mode: FeatureLayer.MODE_ONDEMAND,
    outFields:["*"]
  });

  //add the legend
  map.on("layers-add-result", function (evt) {
    var layerInfo = arrayUtils.map(evt.layers, function (layer, index) {
     return {layer:layer.layer, title:layer.layer.name};
    });
    if (layerInfo.length > 0) {
     var legendDijit = new Legend({
      map: map,
      layerInfos: layerInfo
```

```
      }, "legendDiv");
      legendDijit.startup();
     }
    });

    map.addLayers([huc1, huc2, huc3]);
    });
  </script>


</head>

<body class="claro">
<style>
  html, body {
    margin: 0;
  }
</style>
<div id="content"
     data-dojo-type="dijit/layout/BorderContainer"
     data-dojo-props="design:'headline', gutters:true"
     style="width: 100%; height: 100%; margin: 0;">

  <div id="rightPane"
       data-dojo-type="dijit/layout/ContentPane"
       data-dojo-props="region:'left'">

    <div data-dojo-type="dijit/layout/AccordionContainer">
      <div data-dojo-type="dijit/layout/ContentPane" id="legendPane"
           data-dojo-props="title:'Layers', selected:true">
        <div id="legendDiv"></div>
      </div>
      <div data-dojo-type="dijit/layout/ContentPane"
           data-dojo-props="title:'Tools'">
        <p>getWebData</p><p></p>
        <p>getAnnuals</p><p></p>
        <p>Delineate</p>
      </div>
    </div>
  </div>
  <div id="map"
       data-dojo-type="dijit/layout/ContentPane"
       data-dojo-props="region:'center'"
       style="overflow:hidden;">
  </div>
  <div id="BasemapToggle"></div>
  <div id="search"></div>
</div>
</body>

</html>
```