A GEOGRAPHIC INFORMATION SYSTEMS (GIS) MANUAL FOR THE DEVELOPMENT AND PROGRAMMING OF THE RIVERSCAPE ANALYSIS PROJECT (RAP) DATABASE AND INTERFACE

Prepared for:

Gordon and Betty Moore Foundation

By

John A. Lucotch, Niels K. Maumenee, Diane C. Whited,
Samantha D. Chilcote, John S. Kimball and Jack A. Stanford

Flathead Lake Biological Station
The University of Montana
32125 Bio Station Lane
Polson, MT 59860-6815
Phone: 406-982-3301
Fax: 406-982-3201
john.lucotch@umontana.edu
www.umt.edu/flbs

Submitted November 5, 2009

Flathead Lake Biological Station Technical Report #204-09

Citation:

Lucotch, J. A., N. K. Maumenee, D. C. Whited, S. D. Chilcote, J. S. Kimball and J. A. Stanford. 2009. A Geographic Information Systems (GIS) Manual for the Development and Programming of the Riverscape Analysis Project (RAP) Data base and Interface. FLBS Report No. 204-09. Prepared for Gordon and Betty Moore Foundation, Palo Alto, California by Flathead Lake Biological Station, The University of Montana, Polson, Montana. 52 pp.

INTRODUCTION

The Riverscape Analysis Project (RAP), funded by the Gordon and Betty Moore Foundation, is a remote sensing based classification of salmon rivers across the North Pacific Rim. The goal of this work is to produce a web-accessible decision support data base that will assist salmon conservation around the Pacific Rim based on a robust, hierarchically nested classification (typology) of rivers and river habitats. This classification is aimed at the conservation of existing and potential salmon production in the context of the complexity of river habitats that salmon use for spawning and rearing.

A key component to this study is that salmon productivity in freshwater is linked to complex biologic and hydrogeomorphic pathways that we refer to as a shifting habitat mosaic (SHM) (Stanford et al. 2005). Specifically, our approach assumes that the greater the biophysical complexity of the river systems, the greater the production potential. Potential correlations between complexity and production are examined through the use of a spatially explicit relational data base that links physical landscape complexity to available spawner-recruitment data. The objective of this project is to quantify the distribution and abundance of riverine habitat types important to Pacific salmon production at a variety of scales throughout the range of Pacific salmon.

BACKGROUND

RAP is a spatially explicit data base that allows rivers to be ranked using different combinations of physical metrics in order to target different salmon species based on habitat preferences (for example, sockeye salmon in lakes as opposed to Chinook salmon in riverine areas) or habitat to be maximized across all salmonid assemblages. The magnitude of human impacts can also be included in a ranking, allowing not only potential sources of stress but also potential conflicts with human activities to be assessed. The RAP data base provides a publicly accessible, scientifically based decision making framework, offering a valuable tool to the scientific and management community throughout the range of Pacific salmon.

This report describes the technical aspects of the development of the RAP data base and the publicly accessible web interface. We discuss the acquisition, processing and classification of raw imagery and the programming required to compute metrics and rank each watershed. Lastly, this report describes the approach and steps to develop tools for the public web interface to the RAP data base.

STATEMENT OF THE PROBLEM

The raw data base is the Pacific Rim landscape derived from satellite imagery (reprocessed Landsat 30 meter GSD [Ground Sample Distance] from the NASA Jet Propulsion Lab, Pasadena CA) linked to regional digital elevation models (SRTM 90 m). The digital elevation model (DEM) is used in conjunction with the Landsat imagery to extract the drainage network, define unconstrained floodplain reaches, refine watershed boundaries and the flow network, delineate river channel boundaries and define channel complexity. A Normalized Difference Vegetation Index (NDVI) threshold is used to estimate bankfull of channel areas as

well as distinguish lakes by separating the near-infrared reflectance of lake features from photosynthetically active vegetation. The RAP system is analyzed to delineate physical landscape features at coarse (catchments, river networks, streams, flood plains, lakes, glaciers) and fine (river channel and floodplain attributes that relate to salmon habitat) scales.

Metrics describe characteristics such as average elevation, floodplain abundance, lake area and channel sinuosity. A ranking system that we developed examines potential productivity of metrics based on their suite of habitat characteristics. RAP metrics are first scaled to eliminate incompatible units and used to create comparable metrics. RAP metrics are standardized according to watershed size and then rescaled along a range of 0 to 1 to eliminate incompatible units. Rivers are ranked based solely on lake features; based on all features except lakes; based on all physical features; and based on physical features in addition to human impacts.

APPROACH

Data Base Construction

Digital elevation models

The first step in our hydrological processing is to find a Digital Elevation Model (DEM) of the desired location. We use 90 meter SRTM data in the United States, British Columbia and Kamchatka as well as the 60 meter USGS data in Alaska. The Wild Salmon Center Ecoregions (Pinsky et al. 2009) are used to define the study area boundary. This boundary is buffered to approximately 10 kilometers to make sure all of the desired hydrological extents are within the clipped DEM.

After the DEM is clipped to the desired study area, we begin processing it within ArcHydro (Maidment 2000). Many DEMs contain sink errors or inconsistencies and thus produce incorrect hydrologic areas. If the DEM has negative values indicating sinks, a conditional statement is written:

Con (DEM<0,0,DEM)

This conditional statement states that if the DEM has a value less than zero, it is equal to zero. If the DEM value is greater than zero, we retain the original DEM value.

We then manually digitize a line feature class to burn onto the DEM using the DEM reconditioning tool. This assists ArcHydro by deriving a good set of raster outputs that match the Landsat Mosaic instead of the DEM derived stream.

The final step before we begin the drainage system processing is to setup a location for the layer export. This location is set under the ApUtilities, Set Target Location and Hydro Configuration.

The flow chart displays the order of processing to obtain outputs for catchment processing:

1) DEM Reconditioning

2) Fill Sinks

3) Flow Direction (D8 Algorithm)

4) Flow Accumulation

5) Stream Definition

(Tool that allows you to define how many pixels drain into each pixel, 20 kilometers in this case.)

6) Stream Segmentation

7) Catchment Grid Delineation

8) Catchment Polygon Processing

9) Drainage Line Processing

10) Adjoint Catchment Polygon Processing

After all of these rasters and polygons are developed, we began catchment delineation.

Catchment delineation

Catchments are defined as a location where either a river or a stream drains into the Pacific Ocean or a fjord. This is accomplished with the catchment delineation tool inside ArcHydro. Due to some errors with the digital elevation model, new catchments sometimes overlap or contain gaps. In order to correct this problem, some topology rules are used to clean polygons with errors. There are two specific topology rules that are used 1) polygons must not overlap and 2) polygons must not contain gaps. These two rules correct the DEM errors and allow for a continuous hydrological catchment feature class.

After the catchments are topologically correct, a series of elevation statistics are calculated for each catchment. These statistics are calculated through local parameters in ArcHydro attribute tools. These parameters are defined in xml and can be altered for specific situations, such as catchment parameters. In the xml, catchment statistics are calculated on the DEM that encompasses the catchment. We calculate catchment area, catchment elevation maximum, catchment elevation minimum, catchment elevation mean, catchment elevation median, catchment elevation range and catchment elevation standard deviation. We also derive

two additional statistics, catchment perimeter to area ratio and the coefficient of catchment elevation.

Floodplain delineation

In order to define flood plains, we obtain all the requirements for the inputs into the Floodplain ESRI Arc Marco Language (AML) program developed by Scott Bassett at the University of Nevada Reno. The first input needed is a set of drainage lines within a catchment and their respective stream order. The drainage lines are obtained from the DEM processing. We manually alter them so the drainage lines correctly match the rivers in the Landsat Mosaic. We then manually assign stream order values according to Strahler Stream Order (see *Stream Order Calculation* below) (Strahler 1952). Next, we convert the feature class into a coverage. This coverage and the filled DEM are the inputs for the Floodplain ESRI Arc Marco Language (AML) program.

The AML does a series of processing tasks on the drainage lines and DEM to create possible floodplain areas. The first part of the AML defines the width of the buffers on each segment of the drainage line depending on its stream order. A first order segment would get a buffer width corresponding to its order and a second order segment would get another buffer width corresponding to its order. The buffer width directly increases with an increase in stream order. Next, the program finds areas that are equal to the streams elevation and greater than the streams elevation by a certain elevation value. This value is defined in the AML file and can be changed depending on the buffer distance (Table 1). The elevation values are grouped, shrinking the original buffer. This allows the program to eliminate areas of steep slopes within the newly defined floodplain area. The AML returns a raster of areas fitting those criteria.

A shrink and expand command is used to resample the DEM in order to determine the knick points of the flood plain. We use the shrink geoprocessing tool to shrink the raster by 15 pixels and the expand geoprocessing tool to expand the raster by 15 pixels. After these two geoprocessing tasks are complete, we convert the raster into a polygon, completing floodplain delineation.

The following flow chart describes the steps we use for processing flood plains:

1) Set paths in AML file in the workspace directory

2) Get drainage line to arc coverage

3) Begin AML session

4) Make sure your DEM is 32bit unsigned and under 2 GB

5) Set path to drainage line location w c:\workspace\streamorder

6) Type the word tables and select the coverage name.aat

7) Alter streamorder then insert order and press return three times and enter quit

8) Set your workspace to the AML file location w c:\workspace\aml

9) Then run &r floodplain

10) Take fpout_grid and run shrink <in> <out> <15> <1>

11) Then run an expand <in> <out> <15> <1>

12) Raster to polygon command <in> <out> <no_simplify>

Water classification

Water classification is tricky due to varying river sizes as well as cloud cover over the Pacific Rim. Large flood plains can easily be discerned on both Landsat and Quickbird, allowing a simple NDVI calculation to be used to classify water. We select values less than -.25 for channels containing water. We select values less than 0 for water-only areas plus the parafluvial full areas, which we defined as areas full during the height of runoff. In some cases, small but strategic flood plains, either with parafluvial full areas or just plain water, can not be discerned in Landsat. The NDVI calculation did not accurately cover the smaller areas where mixed pixels become important. For these cases that fall into the mixed pixel classification where the river becomes smaller or is covered by clouds, the following equation is used:

Water Mask = (NIR + Red + .5)/ (Green)

This equation allows for the majority of all water pixels to be highlighted, creating a watermask. This equation is also very useful in detecting the river channels under clouded areas. Unfortunately, if the clouds contained high levels of moisture, then this equation will not work.

Mid-channel classification

After water areas are classified, the water raster is clipped to the floodplain feature class. This operation gives a raster of water confined only to floodplain areas. Next, the floodplain watermask raster is converted into a binary raster where the two categories, water and nonwater, were represented. The raster is then converted into a shapefile. The new watermask is in a vector form with remote sensing data defined as water. The watermask is later used in conjunction with a mid-channel line feature, to calculate the number of separations and returns within a flood plain.

Two separate models, Envelope and Line-Work, are used to obtain a mid-channel line feature. The first task is to densify the watermask polygon using densify cal file in field calculator under the shape column. This allows Thiessan Polygons, the framework behind the Line-Work model, to more accurately follow the river pattern. Then the Envelope model is run on the selected watermask flood plain. We evaluate the results of the model to make sure each

tributary of the flood plain has its envelope cut so that the Line-Work model will run into the tributary areas.

After the envelope editing is completed, we edit the new envelope attribute table. In order to create the spatial join in the Line-Work model, we create a long integer column named SHORE with the object ids of the envelope, defining the shore ids. After populating this column, we are able to run the Line-Work model. The Line-Work model takes the input floodplain watermask and turns all the vertices of the polygon into points. These points are used as input to the Thiessan Polygon geoprocessing tool.

Thiessan Polygons are created representing the river's mid-channel and are spatially joined to the envelope feature class. This new Thiessan Polygon feature class is clipped to the input floodplain watermask, or floodplain boundary. Next, the clipped Thiessan Polygons are dissolved based on the Shore column that was created earlier. At this point, the polygon is converted to a line shapefile where the new feature class contains both a shore and a mid-channel of the flood plain. The mid-channel is selected through an attribute query and exported into a new feature class. For this new mid-channel feature class, three new topology rules were created. The rules are 1) no dangles, 2) no pseudo-nodes and 3) must not overlap. These rules allow for the mid-channel feature class to be geometrically correct so a network can be built to calculate the length down stream.

Calculating nodes for separation and returns

Separations and returns are calculated with nodes using the mid-channel classification. A node is defined as a point vector and represents two channels separating or coming back together. Nodes are a measure of the complexity of a river within a flood plain. The mid-channel line feature is converted to a shapefile in order to classify nodes. Node abundance is calculated using the Hawths Tools (Beyer 2004), which does not support file geodatabase feature classes. We use the Hawths Tools under Vector Editing Tools and then Intersected Lines (Make Points). We insert the line shapefile into the input dialog box and run the program. The resulting shapefile is a point shapefile that represents each time the river separates and reconnects to itself. Next, x and y coordinates are added to each point and dissolved. This ensurs there are no multiple points at the same location. A new node shapefile is created that shows the total number of nodes per flood plain. The node shapefile is attached to the floodplain feature in which the node is located using the following flow of methods:

1) Assign Hydro ID to the Floodplain Feature Class

2) Remove the Hydro ID from the Nodes Feature Class

3) Run a Spatial Join making the Nodes the Target Feature Class

4) Run a Frequency in the Statistics in the Analysis Tools making sure the frequency is the Hydro ID

5) Join the Frequency Table to the Floodplains Feature Class

6) Export the Data to a Floodplain Feature Class with the new node number

We use the following alternative method to calculate nodes for a mid-channel classification. We take the desired mid-channel line and dissolve it by just entering just the polyline tool. Next, we enter the dissolved polyline into the intersect tool, selecting the output as a point. Therefore, the output is a point where the mid-channel separates and returns. This method results in faster processing speeds than the earlier method, so it is primarily used in RAP processing.

Catchment stream order

Stream Order is calculated for each catchment by taking the flow direction raster and the link stream raster developed from the DEM processing and inputting that into Spatial Analyst's Stream Order geoprocessing tool. The resulting raster is an output of the drainage line of the catchment with an attached stream order. This operation can also be done manually, if the drainage lines are already created. Strahler Stream Order takes each drainage line and assigns it a first class stream order. When two first class stream orders intersect that next line segment then becomes a second order stream. This process continues until every line segment has a stream order.

Sinuosity

Sinuosity was calculated by using a Visual Basic (VB) script in the field calculator within a sinuosity feature class' attribute table. Using the main mid-channel feature class created earlier, we select all of the primary channels within the flood plain and export the mid-channel feature to a new shapefile. The primary mid-channel is sometimes fragmented; therefore, we use the following steps to ensure that all of the mid-channel is contained solely within its unique flood plain. The following flowchart lists the steps we use in this process.

1) Select All Primary Mid-Channel from all of the Mid-Channel

2) Export to New Shapefile

3) Clip that to the flood plains

4) Start Editing and Create Multipart Features

5) Save Edits

6) Erase the HydroID in the clipped Mid-Channel

7) Spatial Join Floodplains to Mid-Channel

8) Dissolve the Mid-Channel based on the HydroID

9) Join to Floodplain based on HydroID after sinuosity is calculated

The VB script for calculating sinuosity uses the From Point To Point concept. It takes the x, y coordinate of the From Point, the first digitized point, as well as the x, y coordinate of the To Point, the last digitized point, and records them. The script then takes the x, y coordinates of the two points and runs the following mathematical equation:

Sqr((pFPoint.X - pTPoint.X)^2 + (pFPoint.Y - pTPoint.Y)^2)

where as pFPoint represents the Front Point and pTPoint represents the To Point.

This value is the denominator in the final sinuosity calculation. The numerator is the length of the actual line using the shape length field and is divided by denominator. In some instances, the pTPoint is not at the end of the line feature but is in a different location. This does not accurately represent the true value. In order to correct this problem, four VB scripts are used to calculate the minimum and maximum of both the x and y values. These new values, values from the current spatial reference system, are plugged into the equation above. This corrects the very high sinuosity values.

Lake delineation

Lakes are gathered through the use of remote sensing. Our remote sensing method for lake delineation allows lake boundaries to be more accurately represented than by digitizing the whole lake and its associated islands. It also allows for faster processing speed. The following equation is used to separate water features from all of the material in the Landsat Mosaic:

Water raster = (NIR + Red + .5)/ (Green)

This equation produces values containing decimals. In order to transfer raster data to vector data in ArcGIS, the data has to be in an integer format. We use the integer geoprocessing tool in Spatial Analyst to break the values into separate integer classes. The resulting raster contains classes based on their integer value, highlighting lake water. Lake classes are then turned into a binary raster and exported to a vector shapefile. This approach captures water most accurately from small scales to large scales. Sockeye lakes are further defined as lakes that intersect rivers and streams, allowing fish movement in and out of water bodies.

Mean floodplain elevation

Mean floodplain elevation could be estimated by adding a centroid point to each flood plain within each floodplain feature class. This point could be used to identify an elevation location. However, we find this method is not the most accurate because of the irregular shape of different flood plains. Instead, we extract the mean elevation values from DEMs within the flood plain.

The first step is to clip the digital elevation model to the boundary of the flood plain. Then the floodplains rasters are turned into a shapefile. This creates a very large shapefile due to the number of pixels within each flood plain. Next, each pixel is given its own attribute. In order to find the mean elevation, a spatial join is completed, joining all of the elevation pixels to

the floodplain's HydroID. For example, a flood plain can contain 24 elevation pixels with the Hydro ID of 6, the floodplain's HydroID. After these HydroIDs are joined, the pixels are dissolved based on the HydroID. The mean of the elevational values within the flood plain is calculated and attached to the HydroID in an output table. This table is then joined to the floodplain feature class SDE.

Table joining

The output tables showing all of the feature metrics were developed using the HydroID of the feature. In essence, the HydroID is assigned to each feature in the feature class. Furthermore, all of the metrics dealing with complexity within the flood plain are given a DrainID, an ID relating them to the flood plain in which they are located. The flood plains then are related to the catchment's HydroID by giving them a DrainID that also contains the HydroID of the catchment.

An example of this is the way we relate the node feature class to the floodplain feature class. A spatial join is undertaken by joining the flood plains to the node feature class. The node feature class must not have a column HydroID or the spatial join would not be effective. After the spatial join is finished and the new feature class is output from the geoprocessing model, the HydroIDs are inserted into a new column containing the title FloodplainID. Since there are multiple nodes within the flood plain, a frequency calculation is used in order to get a count of the number of nodes within the flood plain. The floodplain feature class is then joined to the node feature class. We use the same process with other metrics.

Floodplain landcover delineation

In order to delineate floodplain landcover, we first clip the landcover polygon to the flood plains. This shows all of the landcover types within the flood plains. Then we clip the watermask to the landcover type. This gives us the landcover minus the river area and enables us to calculate statistics on the landcover area within each flood plain. In cases where we do not have an accurate landcover feature class, we subtract the area of the flood plain minus the area of the watermask within the flood plain to get a percent vegetation calculation.

Data Base Construction Using Python

Python facilitates Geographic Information System (GIS) development; Python scripts allow analysts and other individuals to run certain geographical analyses without having extensive GIS experience. Scripts built within Python can call the ArcGIS geoprocessing functions performing all sorts of more complex geoprocessing tasks. Python scripts can be used inside the suite of ArcGIS programs or adapted into the web interface for the ArcGIS Server using the RAP Website. Along with these scripts, the new feature set variable type allows the user to also enter their own data. This means the data base is not limited to data produced by Flathead Lake Biological Station (FLBS). Based on the RAP principles, users can also develop their own data. We have also used Python to create scripts for all of the processes described above in previous sections due to its ease of use and faster processing of physical features across all Pacific Rim watersheds.

Mid-channel

The goal of the script is to develop a mid-channel line for water features that were classified by remote sensing techniques or by digitizing. This is a script version of the mid-channel toolbox and simplifies the multistep process undertaken using the mid-channel toolbox (Mid-Channel).

Nodes

The goal of this script is to develop nodes from the mid-channel line. The mid-channel line is dissolved so all of the pseudonodes will be removed. After pseudonodes are removed, an intersection command is used to output points at every intersection point. This script is better than the initial process described above because the script automatically solves the multipoint problem in the earlier node process description (Nodes).

Riparian area

The goal of this script is to calculate the amount of riparian area within the flood plain. The location of the input watermask is deleted from the flood plain. The remaining flood plain area is calculated, yielding a measure of the amount of riparian area left in the flood plain (Riparian).

Sinuosity and dynamic sinuosity internet

The goal of this script is to provide users the opportunity to define any specified area and find the sinuosity of the river in that area. This again uses the feature set with a line symbology. This script finds the maximum and minimum of the input polyline in both the x and y directions and uses those numbers to calculate the sinuosity. The total line length is divided by the x,y calculations as described above (Sinuosity).

Watershed parameters

The goal of this script is to produce an attribute table that corresponds to the RAP Watershed Parameters. This script takes the input watershed and DEM to run zonal statistics. These statistics are then attached to the input flood plain in the naming convention of the FLBS RAP Project (Watershed).

RAP WEBSITE USING ARCGIS SERVER

Geoprocessing

Steps to create a geoprocessing service in arcmap or a web adf

We published our toolboxes in ArcGIS Server. Each toolbox contains the Python script and a model so a few Server special instances can be run. The script has a source that is the same as the machine running ArcGIS Server. The script has a data type as a feature class. This serves as an input and output because this is the only type accepted as intermediate or output data in Server. After these steps are complete, a model is built.

Our first step is to create a feature set variable and then define the variable types. A feature set is a variable allowing the user to draw a point, line or polygon as the input to a certain tool. These are the only types of data supported by ArcGIS Server. In order to fully finish the feature set, we import schema and symbology from a point, line or polygon. Next, we label our intermediate data with a scratchworkspace label. The scratchworkspace label allows the intermediate data to be held on the machine temporally creating benefits for computer storage. This also allows Server to output the data to WEB ADF or ArcMap. We did not use the scratchworkspace symbology until the final output. Next, we set the scratchworkspace under the environments tab of the model, leaving the location blank and set the input and output as parameters. To address the important issue concerning design appeal of the model inside Server, we labeled variables inside our model so the model is more intuitive to users. Then we publish our toolbox.

We moved the toolbox to a folder on the server machine, and we set the appropriate sharing and security so the home ArcGIS Server account (i.e., RAP) and the SOC account are permitted to read the folder. We also put the Python script associated with the published script on the Server computer but in a different folder than the toolbox. We are then able to publish the toolbox as a GIS Resource. Finally, we find our toolbox in the resource table and click next then finish. Our toolboxes are published so that they can be used with the correct web address in an ArcMap session.

We are also able to use the toolbox in a WEB ADF because it is attached to the RAP website. Under the Tasks side panel, we add a geoprocessing task. After this is done, we move the supporting service tab and add the service toolbox to the selected service. Then we move back to the selected tasks tab, highlight the geoprocessing tasks and select configure. Here we name the different tools and give help tips. We ensuret we select the correct model since we had more than one model in a toolbox. This allows tools to be run on the ArcGIS Server Web ADF.

ArcGIS server geoprocessing Python scripting

We write the input and output files for the Python scripts to either a shared folder or to the scratchworkspace depending on the purpose of the script (Appendix A). We find that a local folder works much better than the scratch workspace when using rasters. Furthermore, in Python when using the calculate field, ArcGIS Server does not recognize Visual Basic like ArcGIS

Desktop does. ArcGIS Server only recognizes Python and requires that all field calculations are enclosed in "!variable!". Lastly, when we use scripts in ArcGIS Server, we find that we can not calculate fields from joined tables. We have to either copy the join to a new feature class or table before fields are calculated, or delete the join.

RAP Geoprocessing Scripts

Watershed query

The goal of this tool is to allow the user to query a specific watershed and return the shapefile along with all of the RAP attributes with the option to download a summary shapefile or csv file(Watershed Query).

Variable query

The goal of this script is to allow users to query the flood plain, node or waterbody feature class for a minimum and maximum variable number (i.e., minimum nodes 4 maximum nodes 15) (Variable Query).

The goal of this script is to query watersheds that are similar to a watershed of interest based on ranked attributes such as flood plains, nodes or waterbodies (Similarity Query).

Rank query

The goal of this script is to allow users to search for watersheds with a minimum and maximum rank (i.e., Highest Rank 1 Lowest Rank 10) (Rank Query).

SaRON query

The goal of this script is to provide users the ability to draw a bounding box around SaRON points and return a website link for the SaRON data base in order to request available data (SaRON Query).

RAP table build

The goal of this script is to provide users the ability to build tables on certain watershed parameters instead of the entire suit of RAP variables (RAP Table).

Dynamic subwatershed delineator

In order to create the dynamic subwatershed delineator, we perform some additional processing of rasters so that the tool works properly. The goal of the subwatershed delineator is to allow users to dynamically build watersheds based on their areas of interests and have those watersheds populated with RAP variables. Many DEMs do not accurately portray the earth's surface. Therefore, we first burn the hydrological system into the DEM to accurately portray the true river landscape and not the DEM interpretation (Callow et al. 2006). The first step to do this

is to digitize the stream network based on aerial photographs or satellite imagery. Then we convert the digitized network to a raster with 90 meter cells (the same size as the DEM resolution) using the polyline to raster tool in the conversion toolbox. We convert all raster pixels to zero values using a conditional equation in the Spatial Analyst Raster Calculator:

con (streamraster < or equal to-1, 1, 0)

Then we raise the original DEM's elevation to 100 meters above its original value:

original DEM + 100

Next, we reassign the original DEM values to the zero pixel values in the streamraster using the equation:

streamraster + original DEM

The final step is to merge the raised DEM (+100) with the streamraster that has the original DEM values using the equation:

merge (raiseddem, newstreamraster)

The burned stream network DEM is inserted into the fill, flow accumulation and flow direction tools. In the Stream Definition tool, we set the stream initiation to 0–20 KM. The stream definition raster is inserted into the stream segmentation tool. This method of burning is used because it reduces the errors in sinuous and flat areas that are normally ignored in the regular hydrological processing (Subwatershed).

SUMMARY AND CONCLUSIONS

RAP is a data base, a query system and a web-based dissemination tool. The objective of RAP is to build a framework to support remote sensing and classification of the physical landscapes of rivers around the Pacific Rim. This report details methods for compiling the necessary imagery, analyzing a set of physical metrics and creating a series of web tools to query the main data base. Currently, we are refining the metrics in the data base, the web interface and compiling as much biological data as possible on the escapement, harvest and stock recruit ratios of rivers around the Pacific Rim. Soon we will perform a statistical analysis directly relating the physical landscape metrics to species specific production of juvenile wild Pacific salmon. We are developing a series of video tutorials that show how to use the web interface and the various tools, and how to use the information in the data base to address conservation and management questions related to wild pacific salmon and human activities. Lastly, we are incorporating climate change models into the RAP data base to make predictions about the range of wild salmon across the North Pacific Rim under variable climate conditions. We encourage you to visit the RAP website at http://typology.ntsg.umt.edu/.

REFERENCES

Beyer, H. L. 2004. Hawth's Analysis Tools for ArcGIS. Available at http://www.spatialecology.com/htools.

Callow, J. N., K. P. Van Niel and G. S. Boggs. 2007. How does modifying a DEM to reflect known hydrology affect subsequent terrain analysis? Journal of Hydrology 332:30–39.

Maidment, D. R. (ed.). 2002. Arc Hydro: GIS for Water Resources. ESRI Press, Redlands, California. 224 pp.

Pinksy, M. L., D. B. Springmeyer, M. W. Goslin and X. Augerot. 2009. Rangewide selection of catchments for Pacific salmon conservation. Conservation Biology 23(3):680–691.

Strahler, A. N. 1952. Hypsometric (area-altitude) analysis of erosional topography. Geological Society of America Bulletin 63:1117–1142.

TABLES

Table 1. Buffers around stream channels are used to set the maximum extent of flood plains during extraction from DEMs. Mountain and tundra elevations refer to the maximum allowable elevation above streambeds of flood plains in each biome.

Stream Order	Mountain elevation (m)	Mountain buffer distance (m)	Tundra elevation (m)	Tundra buffer distance (m)
1	1	300	1	400
2	1	600	2	800
3	1.5	1200	2.5	1200
4	2	1500	3	1800
5	2	1750	3	2400
6	3	2000	4	3600
>6	4	2500	4	4800

APPENDIX A

Mid-Channel. Mid-Channel Script

```
gp.Workspace = "C:/typology/typology_dataus/yakima"
out_gdb = gp.Workspace + "/midchan.gdb"
if not gp.Exists(out_gdb):
   gp.createfileGDB_management(os.path.dirname(out_gdb),os.path.basename(out_gdb))
INPUT = "C:/typology/typology_dataus/yakima/yakimawater.shp"
Output = "yakimamidchan.shp"
gp.overwriteoutput = 1
gp.CopyFeatures_management(INPUT,"/midchan.gdb/news")
gp.FeatureEnvelopeToPolygon_management("/midchan.gdb/news", "/midchan.gdb/Envelope",
"SINGLEPART")
gp.Erase_analysis("/midchan.gdb/Envelope", "/midchan.gdb/news",
"/midchan.gdb/Envelope_erase")
gp.MultiparttoSinglepart_management("/midchan.gdb/Envelope_erase",
"/midchan.gdb/envelopes")
gp.Addfield_management("/midchan.gdb/envelopes", "SHORE", "LONG")
gp.CalculateField_management("/midchan.gdb/envelopes", "SHORE", "[OBJECTID]", "VB")
gp.FeatureVerticestoPoints_management("/midchan.gdb/news", "/midchan.gdb/shore_points",
"All")
gp.CreateThiessenPolygons_analysis("/midchan.gdb/shore_points", "/midchan.gdb/thiessen")
gp.SpatialJoin_analysis ("/midchan.gdb/thiessen","/midchan.gdb/envelopes",
"/midchan.gdb/thiessen_join")
gp.Clip_analysis("/midchan.gdb/thiessen_join","/midchan.gdb/news",
"/midchan.gdb/thiessen_clip")
gp.Dissolve_management("/midchan.gdb/thiessen_clip", "/midchan.gdb/dissolve_clip",
"SHORE")
gp.PolygonToLine_management("/midchan.gdb/dissolve_clip", "/midchan.gdb/dissolve_line")
gp.Select_analysis("/midchan.gdb/dissolve_line","/midchan.gdb/final" ,"\"LEFT_FID\" > -1")
gp.CopyFeatures_management("/midchan.gdb/final",Output)
```

Nodes. Nodes Script

```
gp.Dissolve_management(inpfc, "inpfc.shp", "", "", "SINGLE_PART", "DISSOLVE_LINES")
gp.Intersect_analysis("inpfc.shp","nowr.shp","ALL","","POINT")
gp.Dissolve_management("nowr.shp", oupfc, "", "", "SINGLE_PART", "UNSPLIT_LINES")
```

Riparian. Riparian Script

```
gp.Erase_analysis(InputFloodplain, InputWaterMask,"floodplaine.shp")
Area = "float(!SHAPE.AREA!)" ##Calculates Area of New Floodplain Area
gp.Addfield_management("floodplaine.shp","FP_RPAr","DOUBLE")
gp.CalculateField_management("floodplaine.shp", "FP_RPAr", Area, "Python")
rows = gp.searchcursor("floodplaine.shp")##Finds the new Area Calculation
row = rows.next()
rarea = row.getvalue("FP_RPAr")
del rows, row
gp.Addfield_management(InputFloodplain, "FP_RPAR", "DOUBLE")
gp.CalculateField_management(InputFloodplain, "FP_RPAR", rarea, "VB")##Adds the new area
calculation from the search cursor
gp.Delete_Management("floodplaine.shp")
del cur
```

Sinuosity. Sinuosity Script

```
gp.Addfield_management(sin, "min_y", "double")
gp.Addfield_management(sin, "min_x", "double")
gp.Addfield_management(sin, "max_y", "double")
gp.Addfield_management(sin, "max_x", "double")
gp.Addfield_management(sin, "xydiff", "double")
gp.Addfield_management(sin, "linelen", "double")
gp.Addfield_management(sin, "sinuosity", "double")
#Expression to break up shapefile extent
yminExpression = "float(!SHAPE.EXTENT!.split()[0])"
xminExpression = "float(!SHAPE.EXTENT!.split()[1])"
ymaxExpression = "float(!SHAPE.EXTENT!.split()[2])"
xmaxExpression = "float(!SHAPE.EXTENT!.split()[3])"
linelengthExpression = "float (!SHAPE.LENGTH!)"
#Calculates New Fields
gp.CalculateField_management(sin, "min_x", xminExpression, "Python")
gp.CalculateField_management(sin, "max_x", xmaxExpression, "Python")
gp.CalculateField_management(sin, "min_y", yminExpression, "Python")
gp.CalculateField_management(sin, "max_y", ymaxExpression, "Python")
gp.CalculateField_management(sin, "linelen", linelengthExpression, "Python")
#Calculates the Sinuosity of the particular line feature
import math
cur = gp.UpdateCursor (sin)
row = cur.Next()
while row:
   row.SetValue("xydiff",math.sqrt(((row.max_x-row.min_x)*(row.max_x-
row.min_x))+((row.max_y-row.min_y)*(row.max_y-row.min_y))))
   cur.UpdateRow(row)
   row = cur.Next()
del cur
cur = gp.UpdateCursor (sin)
row = cur.Next()
while row:
   row.SetValue("sinuosity", row.linelen/row.xydiff)
   cur.UpdateRow(row)
   row=cur.Next()
del cur
#gp.Deletefield_management(sin, "min_y; min_x; max_y; max_x; xydiff; linelen")
gp.Copyfeatures_management(sin, finalsib)
```

Watershed. Watershed Script

```
import arcgisscripting, sys
gp = arcgisscripting.create()
#Desired Workspace
gp.Workspace = "C:/student/PYTH/database/Hydro"
gp.overwriteoutput = 1
inputs = sys.argv[1]
inputdem = sys.argv[2]
output = sys.argv[3]
#Runs Zonal Statistics on the Digitial Elevation Model
if gp.CheckExtension ("spatial") == "Available":
   try:
      gp.CheckOutExtension("spatial")
      gp.ZonalStatisticsAsTable_sa(inputs, "HydroID", inputdem, "b", "DATA")
      gp.CheckInExtension ("spatial")
   except:
      print gp.GetMessages(2)
else:
   print "Spatial Analyst not available"
###Makes Input Shapefile a Feautre Layer
gp.Copyfeatures_management (inputs, "test")
gp.Makefeaturelayer_management("test.shp", "new")
###Joins Zonal Stats Table to the Feature Layer
gp.Addjoin_management("new", "HydroID", "b", "VALUE")
###Exports Joined Feature Layer to a New Shapefile"
gp.Copyfeatures_management ("new", "rest.shp")
###Delete Unnecessary Columns
gp.Deletefield_management("rest.shp", "b_SUM; b_VARIETY; b_MAJORITY; b_MINORITY;
b_COUNT")
##Add New Desired Columns
gp.Addfield_management("rest.shp", "WS_PA", "double")
gp.Addfield_management("rest.shp", "WS_Z_CV", "double")
gp.Addfield_management("rest.shp", "WS_Z_RANGE", "double")
gp.Addfield_management("rest.shp", "WS_Z_MEAN", "double")
gp.Addfield_management("rest.shp", "WS_Z_MAX", "double")
gp.Addfield_management("rest.shp", "WS_Z_MIN", "double")
gp.Addfield_management("rest.shp", "WS_Z_MEDIA", "double")
gp.Addfield_management("rest.shp", "WS_Z_SD", "double")
gp.Addfield_management("rest.shp", "WS_A_KM2", "double")
gp.Addfield_management("rest.shp", "WS_LAT", "double")
gp.Addfield_management("rest.shp", "WS_LONG", "double")
gp.Addfield_management("rest.shp", "EcoregionI", "double")
gp.Addfield_management("rest.shp", "HydroID", "double")
gp.Addfield_management("rest.shp", "Name", "text")
```

Watershed. Watershed Script (*continued*)

```
gp.Addfield_management("rest.shp", "GlobalID", "text")
gp.Addfield_management("rest.shp", "Area", "double")
gp.Addfield_management("rest.shp", "Length", "double")
###Calculating Area and Length of Each object in the Shapefile
AreaExpression = "float(!SHAPE.AREA!)"
LengthExpression = "float(!SHAPE.LENGTH!)"
###Calculating the New Fields
gp.CalculateField_management("rest.shp", "WS_Z_RANGE", "[b_RANGE]", "VB")
gp.CalculateField_management("rest.shp", "WS_Z_MEAN", "[b_MEAN]", "VB")
gp.CalculateField_management("rest.shp", "WS_Z_MAX", "[b_MAX]", "VB")
gp.CalculateField_management("rest.shp", "WS_Z_MIN", "[b_MIN]", "VB")
gp.CalculateField_management("rest.shp", "WS_Z_MEDIA","[b_MEDIAN]", "VB")
gp.CalculateField_management("rest.shp", "WS_Z_SD", "[b_STD]", "VB")
gp.CalculateField_management("rest.shp", "WS_Z_CV", "[b_STD]/[b_MEAN]", "VB")
gp.CalculateField_management("rest.shp", "HydroID", "[test_Hydro]", "VB")
gp.CalculateField_management("rest.shp", "Name", "[test_Name]", "VB")
gp.CalculateField_management("rest.shp", "GlobalID", "[test_Globa]", "VB")
gp.CalculateField_management("rest.shp", "Area", AreaExpression, "Python")
gp.CalculateField_management("rest.shp", "Length", LengthExpression, "Python")
gp.CalculateField_management("rest.shp", "WS_PA", "[Length]/[Area]", "VB")
###Delete Unnecessary Fields
gp.Deletefield_management("rest.shp", "b_RANGE; b_MEAN; b_MAX; b_MIN;
b_Value;test_Name; b_MEDIAN; b_STD;test_OBJEC;test_Hydro;;test_Ecore;test_Globa;
test_Shape; test_SHA_1; b_Rowid; b_AREA")
##Reproject Shapefile into Geographic Coordinate System
gp.Project_management("rest.shp", "beringfinal.shp",
"GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,2
98.257223563]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]]",
"Pulkovo_1942_To_WGS_1984", "PROJCS['Kamchatka
Albers',GEOGCS['GCS_Pulkovo_1942',DATUM['D_Pulkovo_1942',SPHEROID['Krasovsky_1
940',6378245.0,298.3]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],PROJ
ECTION['Albers'],PARAMETER['False_Easting',0.0],PARAMETER['False_Northing',0.0],PA
RAMETER['Central_Meridian',160.0],PARAMETER['Standard_Parallel_1',52.5],PARAMETE
R['Standard_Parallel_2',67.5],PARAMETER['Latitude_Of_Origin',50.0],UNIT['Meter',1.0]]")
##Grab Latitude and Longitude of Centroid of Each Shapefile
wslongexpression = "float(!SHAPE.CENTROID!.split()[0])"
wslatexpression = "float(!SHAPE.CENTROID!.split()[1])"
gp.CalculateField_management("beringfinal.shp", "WS_LAT", wslatexpression, "Python")
gp.CalculateField_management("beringfinal.shp", "WS_LONG", wslongexpression, "Python")
###Reprojects Shapefile into orginial projection
```

Watershed. Watershed Script (*continued*)

```
gp.Makefeaturelayer_management("rest.shp", "r")
gp.Makefeaturelayer_management("beringfinal.shp", "b")
gp.Addjoin_management("r", "HydroID", "b", "HydroID")
gp.CalculateField_management("r", "rest.WS_LAT", "[beringfinal.WS_LAT]", "VB")
gp.CalculateField_management("r", "rest.WS_LONG", "[beringfinal.WS_LONG]", "VB")
gp.RemoveJoin_management("r", "beringfinal")
gp.Copyfeatures_management ("r", output)
gp.CalculateField_management (output, "WS_A_KM2", "[Area]/1000000", "VB")
gp.Delete_management ("beringfinal.shp")
gp.Delete_management ("test.shp")
gp.Delete_management ("rest.shp")
gp.Delete_management ("b")
```

Stats. Stats Script

```
import arcgisscripting
gp = arcgisscripting.create()
#Workspace Location
gp.Workspace = "C:/R&D"
#Overwrites Output
gp.Overwriteoutput = 1
# Add three temporary fields
gp.Addfield_management("export_output.shp", "temp1", "DOUBLE")
gp.Addfield_management("export_output.shp", "temp2", "DOUBLE")
gp.Addfield_management("export_output.shp", "temp3", "DOUBLE")
# Divide Each attribute by its area
gp.Calculatefield_management("export_output.shp", "temp1",
"[WS_Z_RANGE]/[WS_A_KM2]","VB")
gp.Calculatefield_management("export_output.shp", "temp2",
"[WS_Z_MEAN]/[WS_A_KM2]","VB")
gp.Calculatefield_management("export_output.shp", "temp3",
"[WS_Z_MAX]/[WS_A_KM2]","VB")
# Frequency Analysis done to rank the data
gp.Frequency_analysis("export_output.shp", "table1", "temp1","")
gp.Frequency_analysis("export_output.shp", "table2", "temp2","")
gp.Frequency_analysis("export_output.shp", "table3", "temp3","")
gp.Addfield_management("table1","WOW", "SHORT")
gp.Addfield_management("table2","WOW", "SHORT")
gp.Addfield_management("table3","WOW", "SHORT")
# Get the number of rows in the feature class
time = gp.GetCount_management("table1")
times = gp.GetCount_management("table2")
timer = gp.GetCount_management("table3")
# Shuffles the data and assigns them a ranking 1 through n
gp.Calculatefield_management("table1", "WOW",str(time)+"-[Rowid]", "VB")
gp.Calculatefield_management("table2", "WOW",str(times)+"-[Rowid]", "VB")
gp.Calculatefield_management("table3", "WOW",str(timer)+"-[Rowid]", "VB")
gp.Addfield_management("table1","RANK", "SHORT")
gp.Addfield_management("table2","RANK", "SHORT")
gp.Addfield_management("table3","RANK", "SHORT")
gp.Calculatefield_management("table1", "RANK", "1+[WOW]", "VB")
gp.Calculatefield_management("table2", "RANK", "1+[WOW]", "VB")
gp.Calculatefield_management("table3", "RANK", "1+[WOW]", "VB")
gp.Deletefield_management("table1", "FID; FREQUENCY; WOW")
gp.Deletefield_management("table2", "FID; FREQUENCY; WOW")
gp.Deletefield_management("table3", "FID; FREQUENCY; WOW")
gp.Addfield_management("export_output.shp", "rZRange", "SHORT")
gp.Addfield_management("export_output.shp", "rZMean", "SHORT")
```

Stats. Stats Script (*continued*)

```
gp.Addfield_management("export_output.shp", "rZMax", "SHORT")
# Joins the stats to the orginial shapefile and calculates fields for each
# individual ranking
gp.MakeFeatureLayer_management("export_output.shp", "ranks")
gp.MakeTableView_management("table1", "rtable")
gp.MakeTableView_management("table2", "rtable1")
gp.MakeTableView_management("table3", "rtable2")
gp.Addjoin_management("ranks", "TEMP1", "rtable", "TEMP1")
gp.Calculatefield_management("ranks", "Export_Output.RZRANGE", "[table1:RANK]", "VB")
gp.Removejoin_management("ranks", "table1")
gp.Addjoin_management("ranks", "TEMP2", "rtable1", "TEMP2")
gp.Calculatefield_management("ranks", "Export_Output.RZMEAN", "[table2:RANK]", "VB")
gp.Removejoin_management("ranks", "table2")
gp.Addjoin_management("ranks", "TEMP3", "rtable2", "TEMP3")
gp.Calculatefield_management("ranks", "Export_Output.RZMAX", "[table3:RANK]", "VB")
gp.Removejoin_management("ranks", "table3")
gp.CopyFeatures_management("ranks", "newranks.shp")
gp.Deletefield_management("newranks.shp", "temp1;temp2;temp3")
gp.Delete_management ("table1", "ArcInfoTable")
gp.Delete_management ("table2", "ArcInfoTable")
gp.Delete_management ("table3", "ArcInfoTable")
gp.Addfield_management("newranks.shp", "rOver", "DOUBLE")
gp.Addfield_management("newranks.shp", "rOverall", "SHORT")
gp.Calculatefield_management("newranks.shp", "rOver", "([rZRange] + [rZMean] +
[rZMax])/3","VB")
gp.Frequency_analysis("newranks.shp", "table4", "rOver","")
gp.Addfield_management("table4","RANK", "SHORT")
gp.Calculatefield_management("table4", "RANK", "[Rowid]", "VB")
gp.Deletefield_management("table4", "FID; FREQUENCY")
# Calculates Overall Ranking
gp.MakeFeatureLayer_management("newranks.shp", "r")
gp.MakeTableView_management("table4", "rtable7")
gp.Addjoin_management("r", "rOver", "rtable7", "ROVER")
gp.Calculatefield_management("r", "newranks.rOverall", "[table4:RANK]", "VB")
gp.Removejoin_management("r", "table4")
gp.CopyFeatures_management("r", "finalstats.shp")
gp.Deletefield_management("finalstats.shp", "ROVER")
gp.Delete_management("newranks.shp")
gp.Delete_management ("table4", "ArcInfoTable")
gp.Deletefield_management("export_output.shp","temp1;temp2;temp3;rZRange;rZMean;RZMa
x")
del gp
```

Watershed Query. Watershed Query Script

```
import arcgisscripting,sys,os,math,shutil
gp = arcgisscripting.create()
#Workspace Location
name = sys.argv[1]
outputs = sys.argv[2]
gp.Overwriteoutput = 1
gp.Makefeaturelayer_management("c:/geoshare/overall/toolbox/VR dc
NEQA.sde/neqa.DL.Watershed", "wtshd")
gp.SelectLayerByAttribute_management("wtshd", "NEW_SELECTION", "\"Name\" =
'"+str(name)+"'")
gp.Copyfeatures_management("wtshd", "c:/geoshare/overall/toolbox/newwatershedfinal.shp")
rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatershedfinal.shp")
row = rows.next()
Hydro = row.getvalue("HydroID")
area = row.getvalue("WS_A_KM2")
fpn = row.getvalue("FP_NUM")
ndn = row.getvalue("ND_NUM")
wtbn = row.getvalue("WTB_NUM")
del rows, row
newarea = area * 1000000
fpd = fpn/newarea
ndd = ndn/newarea
wtbd = wtbn/newarea
print newarea
print fpd
print ndd
print wtbd
gp.Deletefield_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp","FP_A_DENS
I; ND_A_DENSI;WTB_DENSIT")
gp.Addfield_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp",
"FP_A_DENSI", "Double")
gp.Addfield_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp",
"ND_A_DENSI", "Double")
gp.Addfield_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp",
"WTB_DENSIT", "Double")
gp.CalculateField("c:/geoshare/overall/toolbox/newwatershedfinal.shp", "FP_A_DENSI",
str(fpd), "Python")
gp.CalculateField("c:/geoshare/overall/toolbox/newwatershedfinal.shp", "ND_A_DENSI",
str(ndd), "Python")
gp.CalculateField("c:/geoshare/overall/toolbox/newwatershedfinal.shp", "WTB_DENSIT",
str(wtbd), "Python")
gp.Addfield_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp", "SHAPEFILE",
"TEXT")
```

Watershed Query. Watershed Query Script (*continued*)

```
gp.Addfield_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp", "TEXTFILE",
"TEXT")
rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatershedfinal.shp")
row = rows.next()
Area = row.getvalue("WS_LAT")
del rows, row
print Area
web = "<a href=http://typology.ntsg.umt.edu/data/overall/text/q"+str(Area)+"t.zip
target=_blank>Download</a>"
web2 = "<a href=http://typology.ntsg.umt.edu/data/overall/shapefile/q"+str(Area)+"s.zip
target=_blank>Download</a>"
cur = gp.UpdateCursor ("c:/geoshare/overall/toolbox/newwatershedfinal.shp")
row = cur.Next()
while row:
   row.SetValue("SHAPEFILE",str(web2))
   cur.UpdateRow(row)
   row = cur.Next()
del cur
cur = gp.UpdateCursor ("c:/geoshare/overall/toolbox/newwatershedfinal.shp")
row = cur.Next()
while row:
   row.SetValue("TEXTFILE",str(web))
   cur.UpdateRow(row)
   row = cur.Next()
del cur
gp.copyrows_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp",
"c:/geoshare/overall/toolbox/zipquerytable/"+str(name)+"_table.dbf")
shutil.copy("c:/geoshare/overall/toolbox/zipquerytable/"+str(name)+"_table.dbf","c:/geoshare/ov
erall/toolbox/zipquerytable/"+str(name)+"_table.csv")
gp.delete_management("c:/geoshare/overall/toolbox/zipquerytable/"+str(name)+"_table.dbf")
gp.copyfeatures_management("c:/geoshare/overall/toolbox/newwatershedfinal.shp","c:/geoshare
/overall/toolbox/zipqueryshapefile/"+str(name)+".shp")
gp.AddToolbox("c:/geoshare/overall/toolbox/ZipandShip.tbx")
gp.ZipData_zip("c:/geoshare/overall/toolbox/zipquerytable",
"c:/geoshare/overall/toolbox/qwatershedt.zip")
gp.AddToolbox("c:/geoshare/overall/toolbox/ZipandShip.tbx")
gp.ZipData_zip("c:/geoshare/overall/toolbox/zipqueryshapefile",
"c:/geoshare/overall/toolbox/qwatersheds.zip")
gp.copy_management("c:/geoshare/overall/toolbox/qwatershedt.zip","c:/Inetpub/wwwroot/Data/
overall/text/q"+str(Area)+"t.zip")
gp.copy_management("c:/geoshare/overall/toolbox/qwatersheds.zip","c:/Inetpub/wwwroot/Data/
overall/Shapefile/q"+str(Area)+"s.zip")
os.remove("c:/geoshare/overall/toolbox/zipquerytable/"+str(name)+"_table.csv")
```

Watershed Query. Watershed Query Script (*continued*)

```
gp.delete_management("c:/geoshare/overall/toolbox/zipqueryshapefile/"+str(name)+".shp")
gp.Copyfeatures_management ("c:/geoshare/overall/toolbox/newwatershedfinal.shp", outputs)
```

Variable Query. Variable Query Script

```
# Import system modules
import sys, string, os, arcgisscripting
# Create the Geoprocessor object
gp = arcgisscripting.create()
gp.Overwriteoutput = 1

# Local variables...
vtype = sys.argv[1]
vmin = sys.argv[2]
vmax = sys.argv[3]
output = sys.argv[4]

hope = int(vmin)
forever = int(vmax)
# Process: Select...
gp.Select_analysis("c:/geoshare/overall/toolbox/VR dc NEQA.sde/neqa.DL.Watershed", output,
"\""+str(vtype)+"_NUM\" >"+str(hope)+" AND \""+str(vtype)+"_NUM\" <"+str(forever))
```

Similarity Query. Similarity Query Script

```
import sys, string, os, arcgisscripting

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

# Local variables...
vname = sys.argv[1]
vtype = sys.argv[2]
output = sys.argv[3]

gp.Overwriteoutput = 1

# Process: Select...
gp.Select_analysis("c:/geoshare/overall/toolbox/VR dc NEQA.sde/neqa.DL.Watershed",
"c:/geoshare/overall/toolbox/newwatershed.shp", "\"Name\" = '"+str(vname)+"'")
rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatershed.shp")
row = rows.next()
grid = row.getvalue(str(vtype))
del rows, row
print grid
import operator
x = 1
hope = operator.add(int(grid),int(x))
print hope
forever = operator.sub(int(grid),int(x))
print forever
gp.Select_analysis("c:/geoshare/overall/toolbox/VR dc NEQA.sde/neqa.DL.Watershed", output,
"\""+str(vtype)+"\" ="+str(hope)+" OR \""+str(vtype)+"\" ="+str(forever))
```

Rank Query. Rank Query Script

```
import sys, string, os, arcgisscripting

# Create the Geoprocessor object
gp = arcgisscripting.create()
gp.Overwriteoutput = 1


# Local variables...
vtype = sys.argv[1]
vmin = sys.argv[2]
vmax = sys.argv[3]
output = sys.argv[4]

hope = int(vmin)
forever = int(vmax)
# Process: Select...
gp.Select_analysis("c:/geoshare/overall/toolbox/VR dc NEQA.sde/neqa.DL.Watershed", output,
"\""+str(vtype)+"\" >="+str(hope)+" AND \""+str(vtype)+"\" <="+str(forever))
```

SaRON Query. SaRON Query Script

```
import arcgisscripting,sys,os,math
gp = arcgisscripting.create()
gp.Overwriteoutput = 1
inputs = sys.argv[1]
outputs = sys.argv[2]
gp.CopyFeatures_management(inputs, "c:/geoshare/overall/toolbox/saron.gdb/saron/extent")
gp.CopyFeatures_management("c:/geoshare/overall/toolbox/saron.gdb/saron/extent","c:/geoshar
e/overall/toolbox/extent.shp")
gp.Addfield_management("c:/geoshare/overall/toolbox/extent.shp", "Left", "double")
gp.Addfield_management("c:/geoshare/overall/toolbox/extent.shp", "Bottom", "double")
gp.Addfield_management("c:/geoshare/overall/toolbox/extent.shp", "Right", "double")
gp.Addfield_management("c:/geoshare/overall/toolbox/extent.shp", "Top", "double")
gp.Addfield_management("c:/geoshare/overall/toolbox/extent.shp", "SaRONDATA", "TEXT")
left = "float(!SHAPE.EXTENT!.split()[0])"
bottom = "float(!SHAPE.EXTENT!.split()[1])"
right = "float(!SHAPE.EXTENT!.split()[2])"
top = "float(!SHAPE.EXTENT!.split()[3])"
gp.CalculateField_management("c:/geoshare/overall/toolbox/extent.shp", "Left", left, "Python")
gp.CalculateField_management("c:/geoshare/overall/toolbox/extent.shp", "Bottom", bottom,
"Python")
gp.CalculateField_management("c:/geoshare/overall/toolbox/extent.shp", "Right", right,
"Python")
gp.CalculateField_management("c:/geoshare/overall/toolbox/extent.shp", "Top", top, "Python")
rows = gp.searchcursor("c:/geoshare/overall/toolbox/extent.shp")
row = rows.next()
l = row.getvalue("Left")
b = row.getvalue("Bottom")
r = row.getvalue("Right")
t = row.getvalue("Top")
del rows, row
web = "<a
href=https://web2.flbs.umt.edu/SaRONResearchWeb/MapDrillDown/SummaryData.aspx?ULlat
="+str(t)+"&ULlon="+str(l)+"&LRlat="+str(b)+"&LRlon="+str(r)+" target=_blank>View</a>"
cur = gp.UpdateCursor ("c:/geoshare/overall/toolbox/extent.shp")
row = cur.Next()
while row:
   row.SetValue("SaRONDATA",str(web))
   cur.UpdateRow(row)
   row = cur.Next()
```

SaRON Query. SaRON Query Script (*continued*)

```
del cur
gp.DeleteField_management("c:/geoshare/overall/toolbox/extent.shp",
"Shape_Leng;Shape_Area;Left;Bottom;Right;Top")
```

RAP Table. RAP Table Script

```
import sys, string, os, arcgisscripting,shutil
gp = arcgisscripting.create()
in_Table = "c:/geoshare/overall/toolbox/watershedall.shp"
in_name = sys.argv[1]
in_Field0 = sys.argv[2]
in_Field1 = sys.argv[3]
in_Field2 = sys.argv[4]
in_Field3 = sys.argv[5]
in_Field4 = sys.argv[6]
in_Field5 = sys.argv[7]
in_Field6 = sys.argv[8]
in_Field7 = sys.argv[9]
in_Field8 = sys.argv[10]
in_Field9 = sys.argv[11]
output = sys.argv[12]
gp.Overwriteoutput = 1
gp.Makefeaturelayer_management(in_Table, "wtshd")
gp.SelectLayerByAttribute_management("wtshd", "NEW_SELECTION", "\"Name\" =
'"+str(in_name)+"'")
gp.Copyfeatures_management("wtshd", "c:/geoshare/overall/toolbox/newwatersheds.shp")
rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
row = rows.next()
lat = row.getvalue("WS_LAT")
longt = row.getvalue("WS_LONG")
del rows, row
gp.CreateFeatureclass("c:/geoshare/overall/toolbox/", "watershed.shp", "POINT")
rows = gp.InsertCursor("c:/geoshare/overall/toolbox/watershed.shp")
newPoint = gp.CreateObject("Point")
newPoint.ID = 1
newPoint.X = str(longt)
newPoint.Y = str(lat)
newRow = rows.NewRow()
newRow.shape = newPoint
rows.InsertRow(newRow)
del newRow
del newPoint
del rows
newtable = "c:/geoshare/overall/toolbox/watershed.shp"
rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
row = rows.next()
latt = row.getvalue("WS_LAT")
longt = row.getvalue("WS_LONG")
Name = row.getvalue("Name")
```

Rap Table. Rap Table Script (*continued*)

```
gp.addfield_management(newtable,"Name","TEXT")
gp.addfield_management(newtable,"WS_LAT","double")
gp.addfield_management(newtable,"WS_LONG","double")
cur = gp.UpdateCursor (newtable)
row = cur.Next()
while row:
   row.SetValue("Name",str(Name))
   row.SetValue("WS_LAT",str(latt))
   row.SetValue("WS_LONG",str(longt))
   cur.UpdateRow(row)
   row = cur.Next()
del cur
try:
   if not gp.Exists(in_Table):
      raise Exception, "Input table does not exist"
   fields = gp.ListFields(in_Table, in_Field0)
   field_found = fields.Next()
   if field_found:
      rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
      row = rows.next()
      f = row.getvalue(str(in_Field0))
      gp.addfield_management(newtable,str(in_Field0),"double")
      gp.calculatefield_management(newtable,str(in_Field0),str(f),"Python")
      del rows, row
   else:
      gp.AddMessage("Field %s not found in %s" % (in_Field0, in_Table))
      gp.SetParameterAsText(2, "False")
      gp.SetParameterAsText(3, "True")
except Exception, errMsg:
   if gp.GetMessages(2):
      gp.AddError(gp.GetMessages(2))
   else:
      gp.AddError(str(errMsg))
try:
   if not gp.Exists(in_Table):
      raise Exception, "Input table does not exist"
   fields = gp.ListFields(in_Table, in_Field1)
   field_found = fields.Next()
   if field_found:
      rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
      row = rows.next()
      f1= row.getvalue(str(in_Field1))
      gp.addfield_management(newtable,str(in_Field1),"double")
```

Rap Table. Rap Table Script…continued

```
        gp.calculatefield_management(newtable,str(in_Field1),str(f1),"Python")
        del rows, row
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field1, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field2)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f2= row.getvalue(str(in_Field2))
        del rows, row
        gp.addfield_management(newtable,str(in_Field2),"double")
        gp.calculatefield_management(newtable,str(in_Field2),str(f2),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field2, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field3)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f3= row.getvalue(str(in_Field3))
        del rows, row
        gp.addfield_management(newtable,str(in_Field3),"double")
        gp.calculatefield_management(newtable,str(in_Field3),str(f3),"Python")
```

Rap Table. Rap Table Script (*continued*)

```
        gp.AddMessage("Field %s not found in %s" % (in_Field3, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field4)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f4= row.getvalue(str(in_Field4))
        del rows, row
        gp.addfield_management(newtable,str(in_Field4),"double")
        gp.calculatefield_management(newtable,str(in_Field4),str(f4),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field4, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field5)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f5= row.getvalue(str(in_Field5))
        del rows, row
        gp.addfield_management(newtable,str(in_Field5),"double")
        gp.calculatefield_management(newtable,str(in_Field5),str(f5),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field5, in_Table))
        gp.SetParameterAsText(2, "False")
```

Rap Table. Rap Table Script (*continued*)

```
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field6)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f6= row.getvalue(str(in_Field6))
        del rows, row
        gp.addfield_management(newtable,str(in_Field6),"double")
        gp.calculatefield_management(newtable,str(in_Field6),str(f6),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field6, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field7)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f7= row.getvalue(str(in_Field7))
        del rows, row
        gp.addfield_management(newtable,str(in_Field7),"double")
        gp.calculatefield_management(newtable,str(in_Field7),str(f7),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field7, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
```

Rap Table. Rap Table Script (*continued*)

```
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field8)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f8= row.getvalue(str(in_Field8))
        del rows, row
        gp.addfield_management(newtable,str(in_Field8),"double")
        gp.calculatefield_management(newtable,str(in_Field8),str(f8),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field8, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
    else:
        gp.AddError(str(errMsg))
try:
    if not gp.Exists(in_Table):
        raise Exception, "Input table does not exist"
    fields = gp.ListFields(in_Table, in_Field9)
    field_found = fields.Next()
    if field_found:
        rows = gp.searchcursor("c:/geoshare/overall/toolbox/newwatersheds.shp")
        row = rows.next()
        f9= row.getvalue(str(in_Field9))
        del rows, row
        gp.addfield_management(newtable,str(in_Field9),"double")
        gp.calculatefield_management(newtable,str(in_Field9),str(f9),"Python")
    else:
        gp.AddMessage("Field %s not found in %s" % (in_Field9, in_Table))
        gp.SetParameterAsText(2, "False")
        gp.SetParameterAsText(3, "True")
except Exception, errMsg:
    if gp.GetMessages(2):
        gp.AddError(gp.GetMessages(2))
```

Rap Table. Rap Table Script (*continued*)

```
    else:
        gp.AddError(str(errMsg))
gp.CopyRows_management("c:/geoshare/overall/toolbox/watershed.shp",
"c:/geoshare/overall/toolbox/new")
gp.copyfeatures_management("c:/geoshare/overall/toolbox/watershed.shp","c:/geoshare/overall/t
oolbox/new.shp")
gp.Addfield_management("c:/geoshare/overall/toolbox/new.shp", "Download", "TEXT")
web = "<a href=http://typology-ags/data/text/q"+str(lat)+"t.zip target=_blank>Download Text
File</a>"
cur = gp.UpdateCursor ("c:/geoshare/overall/toolbox/new.shp")
row = cur.Next()
while row:
    row.SetValue("Download",str(web))
    cur.UpdateRow(row)
    row = cur.Next()
del cur
gp.copyrows_management("c:/geoshare/overall/toolbox/new",
"c:/geoshare/overall/toolbox/zipquerytable/"+str(Name)+"_table.dbf")
shutil.copy("c:/geoshare/overall/toolbox/zipquerytable/"+str(Name)+"_table.dbf","c:/geoshare/o
verall/toolbox/zipquerytable/"+str(Name)+"_table.csv")
gp.delete_management("c:/geoshare/overall/toolbox/zipquerytable/"+str(Name)+"_table.dbf")
gp.AddToolbox("c:/geoshare/overall/toolbox/ZipandShip.tbx")
gp.ZipData_zip("c:/geoshare/overall/toolbox/zipquerytable",
"c:/geoshare/overall/toolbox/qwatershedt.zip")
gp.copy_management("c:/geoshare/overall/toolbox/qwatershedt.zip","c:/Inetpub/wwwroot/Data/t
ext/q"+str(lat)+"t.zip")
os.remove("c:/geoshare/overall/toolbox/zipquerytable/"+str(Name)+"_table.csv")
gp.copyfeatures_management("c:/geoshare/overall/toolbox/new.shp","c:/geoshare/overall/toolbo
x/saron.gdb/Saron/point")
gp.copyfeatures_management("c:/geoshare/overall/toolbox/saron.gdb/Saron/point",output)
```

Subwatershed. Subwatershed Script

```
import arcgisscripting,sys,os,math,shutil
gp = arcgisscripting.create()
#Workspace Location
#Overwrites Output
pourpoint = sys.argv[1]
outputs = sys.argv[2]
gp.Overwriteoutput = 1
gp.CheckOutExtension("spatial")
gp.CopyFeatures_management(pourpoint, "c:/geoshare/world/toolbox/pours.shp")
gp.MakeFeatureLayer_management("c:/geoshare/world/toolbox/pours.shp", "pour")
gp.MakeFeatureLayer_management("c:/geoshare/world/toolbox/VRdcNEQA.sde/neqa.DL.Wate
rshed","watershed")
gp.SelectLayerbyLocation_management("watershed", "INTERSECT", "pour","",
"NEW_SELECTION")
gp.Copyfeatures_management("watershed","c:/geoshare/world/toolbox/selectwtshd.shp")
rows = gp.searchcursor("c:/geoshare/world/toolbox/selectwtshd.shp")
row = rows.next()
database = row.getvalue("DBName")
del rows, row
print database
rename = str(database)[:3]
print rename
#stream binary raster produced in archydro
gp.Snappourpoint_sa("c:/geoshare/world/toolbox/pours.shp", "e:/grids/"+str(rename)+"str",
"c:/geoshare/world/toolbox/west", "200")
gp.Watershed_sa("e:/grids/"+str(rename)+"fdr", "c:/geoshare/world/toolbox/west",
"c:/geoshare/world/script/warast")
gp.Rastertopolygon_conversion("c:/geoshare/world/script/warast","c:/geoshare/world/toolbox/ne
wwatershed.shp","NO_SIMPLIFY")
gp.Delete_management("c:/geoshare/world/script/warast")
# Creates geodatabase setting and adds necessary catchment fields
gp.createfileGDB_management("c:/geoshare/world/toolbox", "water")
gp.CopyFeatures_management("c:/geoshare/world/toolbox/newwatershed.shp",
"c:/geoshare/world/toolbox/water.gdb/newwatershed")
gp.Addfield_management("c:/geoshare/world/toolbox/water.gdb/newwatershed","Name","TEXT
")
gp.Addfield_management("c:/geoshare/world/toolbox/water.gdb/newwatershed","HydroID","LO
NG")
gp.Addfield_management("c:/geoshare/world/toolbox/water.gdb/newwatershed","EcoregionID",
"SHORT")
gp.CopyFeatures_management("c:/geoshare/world/toolbox/water.gdb/newwatershed",
"c:/geoshare/world/toolbox/wtshd.shp")
gp.ZonalStatisticsAsTable_sa("c:/geoshare/world/toolbox/wtshd.shp", "HydroID",
"e:/grids/"+str(rename)+"dem", "c:/geoshare/world/script/b", "DATA")
```

Subwatershed. Subwatershed Script (*continued*)

```
#Makes Input Shapefile a Feautre Layer
gp.Copyfeatures_management ("c:/geoshare/world/toolbox/wtshd.shp",
"c:/geoshare/world/toolbox/test.shp")
gp.Makefeaturelayer_management("c:/geoshare/world/toolbox/test.shp", "new")
gp.Addjoin_management("new", "HydroID", "c:/geoshare/world/script/b", "VALUE")
gp.Copyfeatures_management ("new", "c:/geoshare/world/toolbox/rest.shp")
gp.Deletefield_management("c:/geoshare/world/toolbox/rest.shp", "b_SUM; b_VARIETY;
b_MAJORITY; b_MINORITY; b_COUNT")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_PA", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_CV", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_RANGE", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MEAN", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MAX", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MIN", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MEDIA", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_SD", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_A_KM2", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_LAT", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "WS_LONG", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "Name", "text")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "Area", "double")
gp.Addfield_management("c:/geoshare/world/toolbox/rest.shp", "Length", "double")
#####Calculating Area and Length of Each object in the Shapefile
AreaExpression = "float(!SHAPE.AREA!)"
LengthExpression = "float(!SHAPE.LENGTH!)"
#####Calculating the New Fields
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_RANGE",
"!b_RANGE!", "PYTHON")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MEAN",
"!b_MEAN!", "PYTHON")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MAX",
"!b_MAX!", "PYTHON")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_MIN",
"!b_MIN!", "PYTHON")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp",
"WS_Z_MEDIA","!b_MEDIAN!", "PYTHON")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_SD", "!b_STD!",
"PYTHON")
```

Subwatershed. Subwatershed Script (*continued*)

```
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_Z_CV",
"!b_STD!/!b_MEAN!", "PYTHON")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "Area", AreaExpression,
"Python")
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "Length",
LengthExpression, "Python")
rows = gp.searchcursor("c:/geoshare/world/toolbox/rest.shp")
row = rows.next()
grid = row.getvalue("Area")
grids = row.getvalue("Length")
del rows, row
print grid
print grids
gp.CalculateField_management("c:/geoshare/world/toolbox/rest.shp", "WS_PA",
str(grids)+"/"+str(grid), "PYTHON")
#####Delete Unnecessary Fields
gp.Deletefield_management("c:/geoshare/world/toolbox/rest.shp", "b_RANGE; b_MEAN;
b_MAX; b_MIN; b_Value;test_Name; b_MEDIAN; b_STD;test_Hydro;;test_Ecore; test_Shape;
test_SHA_1; b_Rowid; b_AREA")
#Reproject Shapefile into Geographic Coordinate System
gp.CreateFeatureDataset("c:/geoshare/world/toolbox/water.gdb","project","GEOGCS['GCS_WG
S_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],PRIM
EM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]]")
gp.Copyfeatures_management("c:/geoshare/world/toolbox/rest.shp","c:/geoshare/world/toolbox/
water.gdb/project/project")
gp.Copyfeatures_management("c:/geoshare/world/toolbox/water.gdb/project","c:/geoshare/world
/toolbox/beringfinal.shp")
wslongexpression = "float(!SHAPE.CENTROID!.split()[0])"
wslatexpression = "float(!SHAPE.CENTROID!.split()[1])"
gp.CalculateField_management("c:/geoshare/world/toolbox/beringfinal.shp", "WS_LAT",
wslatexpression, "Python")
gp.CalculateField_management("c:/geoshare/world/toolbox/beringfinal.shp", "WS_LONG",
wslongexpression, "Python")
gp.CalculateField_management("c:/geoshare/world/toolbox/beringfinal.shp", "WS_A_KM2",
"!Area!/1000000", "PYTHON")
#Reprojects Shapefile into orginial projection
gp.Dissolve_management("c:/geoshare/world/toolbox/beringfinal.shp",
"c:/geoshare/world/toolbox/fws.shp","FID;Name;HydroID;EcoregionI;WS_PA;WS_Z_CV;WS_
Z_RANGE;WS_Z_MEAN;WS_Z_MAX;WS_Z_MIN;WS_Z_MEDIA;WS_Z_SD;WS_A_KM2
;WS_LAT;WS_LONG")
nodes =
"c:/geoshare/world/toolbox/VRdc"+str(database)+".sde/"+str(database)+".DL.Hydrography/"+str
(database)+".DL.Nodes"
```

Subwatershed. Subwatershed Script (*continued*)

```
floodplains =
"c:/geoshare/world/toolbox/VRdc"+str(database)+".sde/"+str(database)+".DL.Hydrography/"+str
(database)+".DL.Floodplain"
lakes =
"c:/geoshare/world/toolbox/VRdc"+str(database)+".sde/"+str(database)+".DL.Hydrography/"+str
(database)+".DL.Waterbody"
dams =
"c:/geoshare/world/toolbox/VRdc"+str(database)+".sde/"+str(database)+".DL.Hydrography/"+str
(database)+".DL.Dam"
gp.MakeFeatureLayer_management(nodes, "no")
gp.MakeFeatureLayer_management(floodplains, "fp")
gp.MakeFeatureLayer_management(lakes, "lk")
gp.MakeFeatureLayer_management(dams, "dm")
gp.SelectLayerbyLocation_management("no", "INTERSECT",
"c:/geoshare/world/toolbox/fws.shp","", "NEW_SELECTION")
gp.SelectLayerbyLocation_management("fp", "INTERSECT",
"c:/geoshare/world/toolbox/fws.shp","", "NEW_SELECTION")
gp.SelectLayerbyLocation_management("lk", "INTERSECT",
"c:/geoshare/world/toolbox/fws.shp","", "NEW_SELECTION")
gp.SelectLayerbyLocation_management("dm", "INTERSECT",
"c:/geoshare/world/toolbox/fws.shp","", "NEW_SELECTION")
gp.Copyfeatures_management("no", "c:/geoshare/world/toolbox/no.shp")
gp.Copyfeatures_management("fp", "c:/geoshare/world/toolbox/fp.shp")
gp.Copyfeatures_management("lk", "c:/geoshare/world/toolbox/lk.shp")
gp.Copyfeatures_management("dm", "c:/geoshare/world/toolbox/dms.shp")
##gp.Statistics_analysis("c:/geoshare/world/toolbox/fp.shp","c:/geoshare/world/toolbox/fpstats.d
bf", "xydiff SUM; linelen SUM")
##rows = gp.searchcursor("c:/geoshare/world/toolbox/fpstats.dbf")
##row = rows.next()
##xy = row.getvalue("SUM_xydiff")
##line = row.getvalue("SUM_linele")
##del rows, row
##print xy
##print line
gp.Addfield_management("c:/geoshare/world/toolbox/fws.shp", "WTBNUM", "DOUBLE")
gp.Addfield_management("c:/geoshare/world/toolbox/fws.shp", "NDNUM", "DOUBLE")
gp.Addfield_management("c:/geoshare/world/toolbox/fws.shp", "FPNUM", "DOUBLE")
gp.Addfield_management("c:/geoshare/world/toolbox/fws.shp", "DAMNUM", "DOUBLE")
#gp.Addfield_management("c:/geoshare/world/toolbox/fws.shp", "FP_SIN", "DOUBLE")
```