Charming the Snake

An introduction to Python and the ArcPy site-package

By Thad Tilton, Geographic Information Services, Inc.

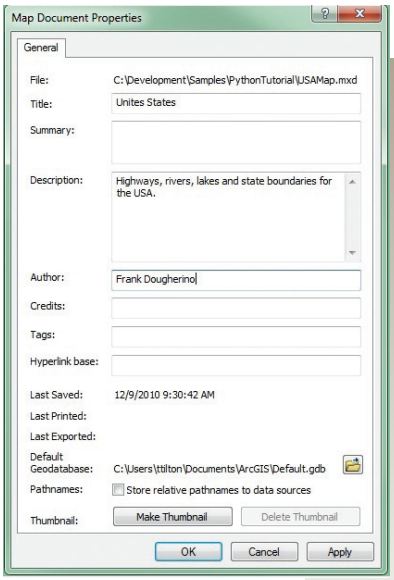
While fearing a real python might be justified, you shouldn't be intimidated by the Python language. Esri has embraced Python as the language that fulfills the needs of its user community.

Programming with Python using the new ArcPy site-package in ArcGIS 10 makes GIS programming accessible for the average GIS professional. For longtime GIS users who yearn for the simpler days of Avenue (the scripting language used with ArcView 3.x), Python has some of the same attractive features: a manageable number of well-documented classes and functions, scripts that can be easily loaded into a document and executed, code shared as simple text files, and (perhaps best of all) no need to know what QueryInterface means. Python is a widely used, nonproprietary language, so learning Python can be beneficial for non-GIS applications.

To help you get started with Python, this tutorial will describe how to create a script that produces a simple report for the current map document. The generated report will contain information about all data frames and layers in the map. Optionally, it can list any layers that have broken data links and create a map package (*.mpk) for the document.

While the script itself may not be particularly exciting, the tutorial's primary goal is illustrating techniques commonly used when writing Python scripts. Specifically, the tutorial will show how to

- Reference additional libraries (i.e., import functionality) for use in a script.
- Accept user-provided arguments.
- Show messages that report the script's progress.
- Get a reference to the current map document.
- Execute a geoprocessing tool.
- Access objects in a map document (data frames, layers, tables).
- Write branching and looping statements.
- Create, open, and write to files.



Make sure the map document has the Title, Description, and Author fields filled out.

What You Will Need

- ArcGIS Desktop 10 (ArcView, ArcEditor, or ArcInfo license)
- An ArcMap document containing at least two data frames with several data layers in each frame
- The script for this exercise, MxdReport.py, downloaded from esri.com/arcuser

Getting Started

Download MxdReport.py from the *ArcUser Online* Web site (esri.com/arcuser) so you can reference it as you do the exercise.

There are several ways to run Python code in ArcGIS. The Python Window, new with ArcGIS 10, can be used to run single lines of Python code or load an entire script into a command line-type dialog. However, for authoring full scripts, such as the one described in this tutorial, a Python integrated development environment (IDE) is the best option. Several IDEs are available for Python. The Python IDLE editor, which is included with ArcGIS 10, will be used in this exercise.

1. Open ArcMap. Choose a map document for this exercise that contains several data frames. Each data frame should contain several layers.
2. Make sure the map document has the Title, Description, and Author fields filled out. With the map document open, go to File > Map Document Properties to verify this. Do not omit this step. `(Add this info.)`
3. In ArcMap, display the ArcToolbox window by choosing Geoprocessing menu > ArcToolbox.
4. Open ArcCatalog. Right-click the geodatabase referenced by the map document. Choose New > Toolbox. Name the new toolbox GetStart.

Exercise3.gdb

5. Right-click ArcToolbox, choose Add Toolbox, and select the GetStart toolbox to add it. `Rename the new Toolbox to GetStart`
6. Use Windows Notepad or another text editor to create a new text file called MxdReport.py.
7. Return to ArcMap and create a new script inside the new GetStart toolbox by right-clicking the toolbox and choosing Add > Script.
8. In the Script Wizard, name the script MxdReport, add the label Create MXD Report, and click Next.
9. For the script file, navigate to the MxdReport.py text file created earlier and add it. Click Next.
10. Add three parameters (Output Path, Package Map, and Track Broken Links, in this order) and choose the values shown in Table 1 for each parameter. Click Finish.
11. Open the MxdReport.py script in the IDLE editor by right-clicking on the script and choosing Edit. The IDLE editor will open with two windows. One will contain the MxdReport script file. The other, called the Python shell, can be used for executing Python statements interactively, but it won't be used here, so close it.

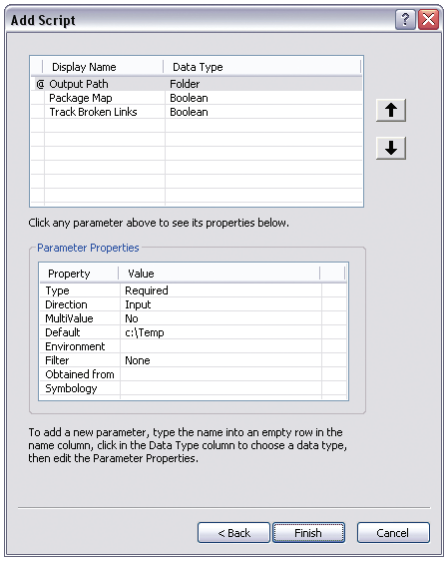
Continued on page 54

Charming the Snake

Continued from page 53

Parameter Name	Data Type	Default Value
Output Path	Folder	C:\Temp
Package Map	Boolean	False
Track Broken Links	Boolean	True

Table 1: MxdReport.py script parameters



Define the properties for the script by adding three parameters (Output Path, Package Map, and Track Broken Links) using the values shown in Table 1.

Importing Additional Modules

Python modules are files used to organize sets of additional functionality so your script can access it. The import statement is one way to bring a module into a script and make the functionality it defines available. At the top of the script window, type the following import statements:

```
import os
import sys
import arcpy
import arcpy.mapping as mapping
```

The os and sys modules contain functions the script will need for working with files on disk. The ArcPy site-package will allow access to geoprocessing tools. The ArcPy mapping module provides functionality for working with the map document and the objects it contains.

Reading User Inputs

Commonly, a script will need some input from the user to run. When creating a script, parameters can be defined to specify the information to be entered by the user. ArcGIS uses parameter information to create the necessary input controls. The order of parameters defined for the script is important, because parameter values are read from an ordered list of inputs. Add the following code to read user inputs:

```
outDir = arcpy.GetParameterAsText(0)
packageMap = arcpy.GetParameter(1)
checkBrokenLayers = arcpy.GetParameter(2)
```

GetParameterAsText reads the user input for the output file location (for example) as a text string. GetParameter reads user input and returns an object (e.g., Boolean).

Getting a Reference to the Current Map

When scripting in any environment, the current document is often the key object. Once a reference to the current document is obtained, the script can drill down into the additional objects it contains. A reference to the map document will provide access to the data frames, layers, and tables the map contains. Add the code below to get a reference to the current map document and store it in a variable called mxd.

```
mxd = mapping.MapDocument('Current')
```

The required parameter for the MapDocument function is either a path to a map document on disk or the keyword Current to get the map in which the script is executing.

Opening a File on Disk

The report generated by this script will be output as a simple text file to a location specified by the user. Add the code below to build the output file name for the report:

```
reportPath = outDir + '\\' + mxd.title + '.txt'
```

The + operator is used to concatenate strings in Python. The output report file will have the same name as the map, but with a .txt extension (e.g., C:\\Temp\\USA.mxd.txt).

Use the open function to open the file:

```
reportFile = open(reportPath, 'w')
```

The open function takes two arguments: the file name and the file mode. Valid file modes are *w* for write, *r* for read, and *a* for append. If the file exists, it will be overwritten. If it does not exist, it will be created. Add the line of code below to report the script's status to the user. This message will appear in the script's output window as it executes.

```
arcpy.AddMessage('Writing report to ' + reportPath)
```

Getting Information about the Map

The MapDocument object has several properties for getting basic information about the map. Create a variable to hold the report text and begin writing the report.

```
reportText = 'Title: ' + mxd.title + '\n'
reportText += 'Author: ' + mxd.author + '\n'
reportText += 'Description: ' + mxd.description + '\n'
reportText += '~~~~~~~~~~~~~~~~~~~~~~~~' + '\n'
```

The \n is an escape sequence used to generate a carriage return. The += combination is a shortcut for appending text to the end of the variable (e.g., reportText = reportText + "new text").

Writing a Simple Branching Statement

In all programming languages, the term *branching* refers to the ability to make decisions in the code. The most basic example is an IF/THEN statement, which will perform an action only if a specified condition is true. Define the following branching statement that checks if the user wants to create a map package:

```
if packageMap:
```

Indentation is often used in scripts to make code more readable. In Python, indentation has a functional importance. Code that is indented under a branching statement indicates that the code should run in response to the condition being true. To check indentations, refer to MxdReport.py, downloaded from *ArcUser Online*. Add code to build the output path for the map package.

```
packagePath = outDir + '\\' + mxd.title.replace('.', '_') + '.mpk'
```

Build another branching statement that checks whether the output map package exists. If it does, add a message that informs the user the package will not be re-created:

```
if (os.path.exists(packagePath)):
   arcpy.AddMessage('Map package already exists
                    (' + packagePath + ')')
```

Executing a Geoprocessing Tool

Any geoprocessing tool that is available from ArcToolbox may be executed programmatically using ArcPy. To create a map package, you will execute the PackageMap geoprocessing tool and pass in the required arguments. Add an ELSE section to the branching statement started above. Add a message to indicate that the map package is being created:

```
else:
   arcpy.AddMessage'(Creating map package
                    ' + packagePath + ')' )
```

Use ArcPy to call the PackageMap geoprocessing tool.

```
arcpy.PackageMap_management(mxd.filePath,
                            packagePath)
```

PackageMap_management requires the path to the map document and the path to the output package file.

Loop through All Data Frames in the Map

In Python, you may wish to loop based on a condition (a WHILE loop) or over a collection of items (FOR loop). Use mapping.ListDataFrames to get a list of data frames from the map. Make sure to begin your code at the left edge of the editor (i.e., without indentation) to ensure it runs outside of the previous branching statement:

```
dataFrames = mapping.ListDataFrames(mxd, '')
```

The ListDataFrames function takes two arguments: a map document (or a path to one) and a wildcard string for filtering data frames based on name (e.g., La*). The function returns a Python list of data frames. Start a FOR loop to iterate over the data frames in the list:

```
for frame in dataFrames:
```

Add information about the data frame to reportText string. Like a branching statement, the code to execute inside a loop must be indented beneath the FOR statement.

```
reportText += '\nData Frame: ' + frame.name + '\n'
reportText += ('Spatial Reference: ' + frame.spatial
               Reference.name + '\n')
```

Loop Through All Layers in Each Data Frame

Inside the loop that iterates over data frame objects, a nested loop will iterate over the collection of layers inside each data frame. Add the code below to complete the nested loop of layer objects. Don't forget to indent this code beneath the parent loop.

```
layers = mapping.ListLayers(mxd, '', frame)
i = 0
for lyr in layers:
   reportText += '\tLayer ' + (str(i) + ': ' + lyr.
                              name + '\n')
   i += 1
```

Similar to the ListDataFrames function, ListLayers returns a Python list object containing layer objects. The str function converts a value (e.g., integer) to a string.

Find Layers That Have a Broken Data Source

Using Python to locate layers in the map that have missing data sources is easy. Rather than checking each layer individually, a single function—ListBrokenDataSources—can be used to return a Python list of broken layers. If the user has requested this information in the report, use ListBrokenDataSources to get a list of layers. The LEN function returns the number of items in a list.

```
if checkBrokenLayers:
   arcpy.AddMessage('Checking for missing data
                    sources')
   brokenList = mapping.ListBrokenDataSources(mxd)
   reportText += ('\nFound ' + str(len(brokenList))
                  + ' layers with missing data.')
   for broken in brokenList:
        reportText += '\t- ' + broken.name + '\n'
```

Write Text File to Disk

At this point, all the text for the report is contained in the reportText variable. All that's needed now is to write this string to the output file that was created earlier. After writing the text stored in the reportText variable to the file, close it and delete the mxd variable using this code.

```
reportFile.write(reportText)
reportFile.close()
del mxd
```

Test the Script

The script is now ready to run.

1. Save changes to the script in the IDLE editor.
2. Double-click the script in ArcToolbox to launch the input dialog. The controls were built from the settings you chose.
3. Specify an output folder for the report, check all check boxes, and run the script.
4. As the script executes, watch the output window for the messages you added. Once execution is complete, check the output text file and map package to verify that they were generated correctly.

Conclusion

With the ArcPy site-package, Esri has integrated Python into ArcGIS 10. Python's relative simplicity and power make it the scripting language of the future for ArcGIS. For more information, contact Thad Tilton at ttilton@gisinc.com.

About the Author

Thad Tilton is a senior GIS developer for Geographic Information Services, Inc.

Python's relative simplicity and power make it the scripting language of the future for ArcGIS.