



JANUARY 12, 2017

INPUTTING DATA

TOPIC 6

VINCENT A. DINOTO, R.
JEFFERSON COMMUNITY AND TECHNICAL COLLEGE
Vince.dinoto@kctcs.edu

Contents

Introduction	3
Why Create a Tool.....	3
Functions to be Covered	3
Examples	4
Example 1.....	4
Example 2.....	4
My Toolboxes.....	4
GetParameterAsText.....	5
Problem.....	6
Creating a Script.....	6
Adding the Script.....	8
Editing and Correcting Errors	10
Assignment 6.1.....	10

Figures

Figure 1: My Toolbox 5

Figure 2: Storage script 7

Figure 3: Script Wizard Screen 1 8

Figure 4: Script Wizard Screen 2 (note only part of the screen shown) 9

Figure 5: Data Types (note only part of the wizard is visible) 9

Introduction

The Toolbox is one of the main ways in which geoprocessing is created within Esri ArcMap Desktop. To this point, tools from existing toolboxes have been employed with Python scripts to do specific tasks. Multiple geoprocessing tools can be used in a single script, but the inputs were provided within the script not queried from the user, thus the script had to be modified each time new data was used. Typically, when a tool is used in geoprocessing in Esri ArcMap the user specifies the input, output and other parameters.

1. The format used: `arcpy.toolname_toolbox` (input, output, etc.)
2. The input, output, etc. could be a variable of direct information.

In this module, tools for specific tasks will be created based on existing geoprocessing tools and placed within a user toolbox. The script will not create a new geoprocessing function but instead will call one or more existing tools to be used for a particular purpose. For example, if you constantly clip and merge items for an operation, a tool could be created that does this process. The actual creation of a new geoprocessing tool is beyond the scope of this course.

Creating a tool for a personal toolbox will utilize a construction wizard. The process could be directly coded but this is generally beyond the context of this course.

Why Create a Tool

Any user of Esri ArcMap knows that it contains many tools and most are never used for typical operations, so why should the user create personalized tools?

- Many times a user will do similar operations as part of their daily routine, which involves multiple clicks of the mouse, this routine does not have any variation except potentially the input and output parameters.
- Many redundant processes could be automated, for example if the process is a set of clips after a merge, there is no reason to navigate manually through the tool menus to complete this process, a single tool could do the processes for the user. See the case study at the end of module 4.
- It is very important that proper file naming convention and storage locations be followed to allow for maximum automation.
- As with any Python, script the purpose is to make the technologist more productive and allow for informed decisions, and not be burdened by redundant processes. These processes are time consuming and requires little to no expertise in the process. To design the tool does take extensive thought.

Functions to be Covered

- ToolBox scripting wizard
- `GetParameterAsText`
- `ListFeatureClasses`
- `Arcpy.describe`
- `Copy Features`

Examples

Example 1:

At the National Geospatial Technology Center of Excellence (GeoTech Center), demographical research requires that specific datasets and geographical shapefiles be joined together for different communities.

- The datasets are always the same type of information, such as median income, educational attainment, race and gender. The feature files are census tract information, but the files represent different regions across the country. In the operation the following are manually done:
 - Creation of a new column (field) with a specific data type
 - The new field must be populated with data from a different field (this is done to change the format between the two columns, one was numerical and the other is text).
 - A tabular join is completed between the spatial file and the spreadsheet that contains demographic information.
- Each time this operation is done manually, the steps are identical.
- The input files and output files contain different information, but the same topics.

Example 2

The GeoTech Center creates regional geographies for two-year colleges.

- Shapefiles are merged together to create the regional boundary, this is usually done both for county boundaries and census tracts.
- The newly created boundaries are used to clip data from state level datasets like roads and rivers.
- The merging and clipping are the same each time, different counties and states are used, thus the input names and the output names vary.

My Toolboxes

As noted previously, new tools are not being created, but instead the combination of different tools are being created to meet the needs of specific repetitive processes. In previous modules, tools have been used to automate processes, but there were no parameters created for input of information by the user, thus the code had to be modified for each application.

The tool creation will be performed through the use of a wizard and not require extensive Python script composition and is referred to as a My Toolbox. An example of a tool that is a python script is the multiple ring buffer, which is a python tool that calls the buffer command numerous times and uses a looping function.

.

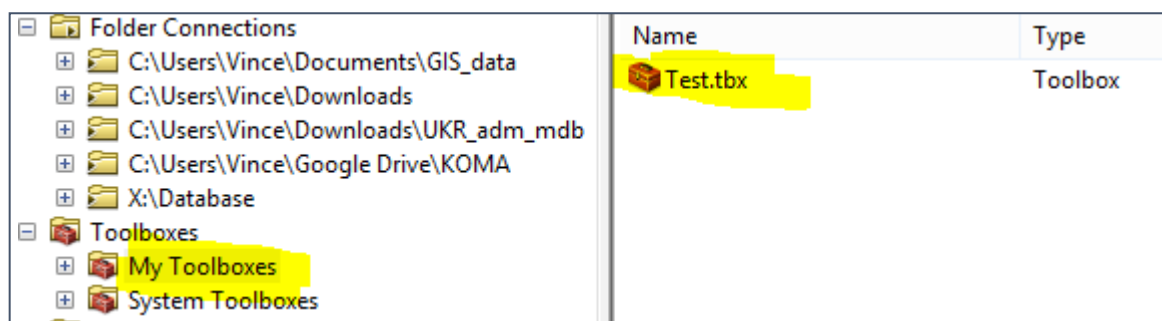


Figure 1: My Toolbox

- Open Esri ArcCatalog and locate the Toolboxes section, see Figure 1.
- Right click on My Toolboxes, select New then Toolbox, and provide an appropriate name. In our example, Test was the name chosen. Do not select the Python Toolbox, but just Toolbox.
 - My Toolboxes use an extension of .tbx
 - Python Toolboxes use an extensions of .pyt
- This process has created a blank toolbox, which contains no tools.

To add a script to the toolbox, the toolbox can initially have a blank script or a script can be composed before creating the toolbox. To create a blank script file:

- open your IDE with an empty file
- save the blank file
- this will create a blank script file

There are multiple ways to create your script file but it is recommended that you use an IDE, which will be the process demonstrated. Once the script file has been created in the IDE it will be added to the tool. The script file is added to the tool by using the add script wizard in ArcCatalog. Once the script is tested, edits can be made in the IDE or any other text editor such as Notepad. The tool will only run successfully within ArcMap or ArcCatalog and will **not** be fully functional in the IDE. Since the script will run inside ArcMap or ArcCatalog loading the arcpy module is not required.

GetParameterAsText

The **GetParameterAsText** is the arcpy command that will be used and it will create a dialog box that will allow the user to input information. The utilization of this tool is a two-step process:

1. Creating the code for the script
2. Setting the input parameters, done in the Add Script Wizard

The **GetParameterAsText** has a single attribute, which is an index; the index will be zero for the first input box. The index will be incremented with each additional use of the command, i.e. additional input boxes. The format of the function is:

Variable = arcpy.GetParameterAsText(index)

Problem

In earlier versions of ArcMap, most files were stored as individual shapefiles and used the Microsoft Windows Tree Structure to separate geographic regions; today it is strongly recommended that mapping data (feature class files) be stored within a geodatabase. This change in how information is stored has required organizations to move shapefiles into geodatabases. The script that will be composed will complete this task in an automated process. This discussion of geodatabases will **not** include relational databases but only file and personal geodatabases. It is assumed that the geodatabase has already been created prior to working on this lesson, if it has not been created, create an empty geodatabase. Make sure that none of the shapefiles to be placed in the geodatabase already exists in it; this will cause an error or will create a file with a different name. This occurs since there are no commands in the script provided for overwrite existing files.

- For example, while creating the script, it is run and thus the part of the scripting putting the feature class file into geodatabase is executed, therefore the geodatabase is no longer empty and will contain a file, when the script is run again a conflict will arise. The user needs to use ArcCatalog and manually delete the file from the geodatabase. If the file is not removed, the script will create a file with a generic name or a number after the provided name, thus each time the script is tested another file will be created which will clutter the geodatabase with useless information.
- For this example make sure all the shapefiles to be moved to the geodatabase are placed in the same file folder, remember shapefiles are composed of multiple individual files and all must be placed appropriately in the same folder. The script that will be created will move all files from a single folder.

In all examples of Python scripts created to this point, the user defined the environment, which stated the location of the data and the storage location of the information. The user had no control over these parameters when the script was executed. In toolbox construction, the user will input these parameters.

The user will specify the input folder that will contain the shapefile(s). The user will also specify the geodatabase that the geospatial files will be stored in. As noted previously it is assumed that the geodatabase has already been created before running the script.

Creating a Script

In Figure 2 the script is shown moving the shapefiles to the geodatabase; it was created in an IDE but not executed.

```

#load the arcpy and os
import arcpy
import os
#load the environment
from arcpy import env
#The first input window will define the workspace
env.workspace=arcpy.GetParameterAsText(0)
#The second input window will define the geodatabase that will be used
#note: geodatabase must already exist prior to running the script
output=arcpy.GetParameterAsText(1)
#get the feature class files that are contained in the defined workspace
shapefiles=arcpy.ListFeatureClasses()
#loop that writes the files to the geodatabase
#make sure the geodatabase does not contain the files to written
for shape in shapefiles:
    shapeDescription=arcpy.Describe(shape)
    arcpy.CopyFeatures_management(shape, os.path.join(output, shapeDescription.basename))

```

Figure 2: Storage Script

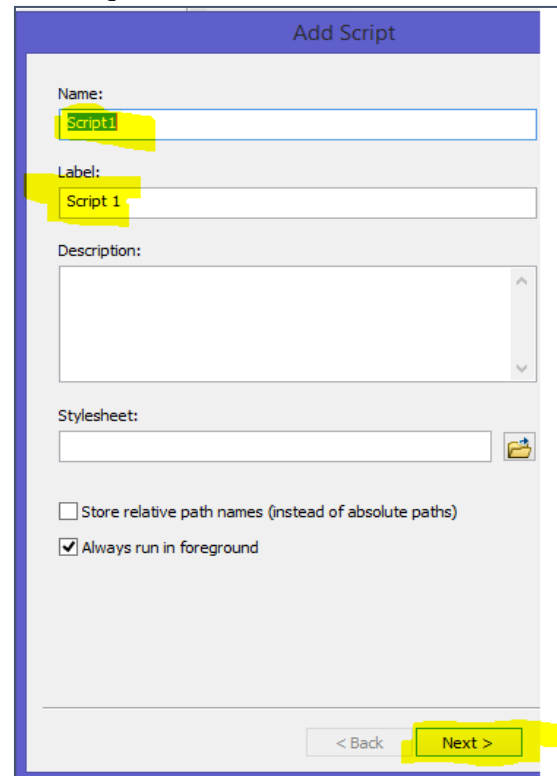
1. Loading arcpy (came from the author's starting template) is not required but will not damage the process.
2. The OS is the operating system and is required to make this script function properly.
3. The environment is loaded from arcpy. The environment will be used in context with the **GetParameterAsText** command.
4. In the next line of code, the workspace will be set to the value inputted by the user in the first **GetParameterAsText** with an index of 0, this is the location of the shapefile folder.
5. The next functional line again uses the **GetParameterAsText** with a 1 index, which will be the user supplied output location; this is the location of the geodatabase.
6. The **ListFeatureClasses** will copy all the shapefiles that are saved in the specified workspace. No parameters are required with this command, so the variable *shapefile* contains all the content from a specified file folder.
7. A **for** loop is used to write each shapefile to the geodatabase. Each iteration of the loop will write one shapefile and the associated information to the geodatabase. The variable *shape* will contain the information to be written. The command **Describe** and **CopyFeatures** are used. The **Describe** command returns the properties of the file contained in *shape*. The **CopyFeatures** command has multiple parameters. The first parameter is the input feature, which is contained in the file, named shape, the second parameter is the output location and the last parameter is the name and properties of the file.

Once the script is written the next step will be to use the script in the tool wizard.

Adding the Script

Normally when a script is constructed, it is run in the IDE and edited until the script runs properly without errors and that is the end of the process. For this case running the script is possible within the IDE, but may not be functional since it requires the use of the toolbox wizard. If instead the script was the creation of a Python toolbox, which requires more programming, then it might fully execute. That type of script requires more Python scripting knowledge than is covered in this course. The next step in the construction process will be using the tool wizard to load the script properly, see Figure 3.

1. The first step is opening ArcCatalog and right clicking on your Toolbox.
2. Click Add and the Script Wizard should open.
3. In the Name field select an appropriate name with no spaces or other special characters.
4. The Label field is the visible name of the script and can be more descriptive by using spaces and special characters. This name will be the name seen in the toolbox.
5. Click on the Next button to continue the wizard.



The screenshot shows the 'Add Script' wizard interface. It has a title bar 'Add Script'. Below it are four input fields: 'Name:' with 'Script1', 'Label:' with 'Script 1', 'Description:' (empty), and 'Stylesheet:' (empty). Below the 'Stylesheet' field are two checkboxes: 'Store relative path names (instead of absolute paths)' (unchecked) and 'Always run in foreground' (checked). At the bottom right are two buttons: '< Back' and 'Next >'. The 'Next >' button is highlighted with a yellow box.

Figure 3: Script Wizard Screen 1

6. The next step is providing the pathway to the script file which was written in the IDE. See Figure 4

7. Use the browse folder button to locate the script file.

8. Click on next at the bottom of the wizard screen.

9. In the next window (Figure 5) of the script wizard, there will be a need for an alignment of the **GetParameterAsText** with the data input from the user. This is the critical point in the process and must be done carefully to insure that the proper parameters are inputted into the tool.

10. Each row must have a one to one correspondence with each input that is designed in the script.

11. The display name in Figure 5 is the name, which will appear in the tool, the designer must select an appropriate name. The author has chosen two names to be used in the tool.

12. The data type must be set properly to insure accurate functionality. For the Input Data, Disk Connection was selected from a pull down menu. This data type gives the user the ability to specify a folder that will contain the shapefiles. The Output GDB has a Data Type of Workspace or Feature Dataset. This data type was selected to allow for browse button usage.

The user will be able to input the information in a specified geodatabase.

13. When a row is highlighted the Parameter Properties will be shown for that specific row and other parameters that can be controlled. In both cases the default values were used.

14. The number of rows must be in agreement with the number of **GetParameterAsText** in the script. The index value will correspond to each input so the zero parameter is the Input Data and the one parameter is the Output GDB.

15. Click next to continue.

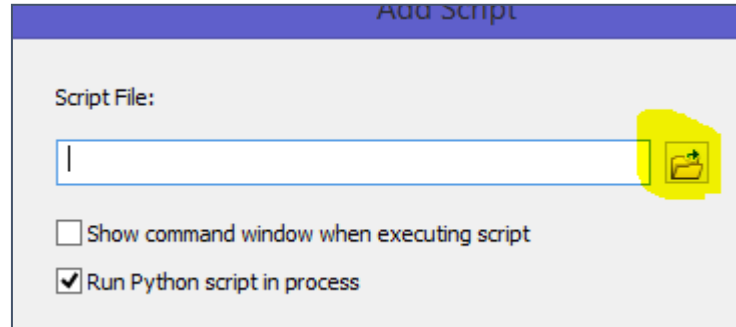


Figure 4: Script Wizard Screen 2 (note only part of the screen shown)

Display Name	Data Type
Input Data	Disk Connection
Output GDB	Workspace or Feature Dataset

Click any parameter above to see its properties below.

Parameter Properties	
Property	Value
Type	Required
Direction	Input
MultiValue	No
Default	
Environment	
Filter	None
Obtained from	

Figure 5: Data Types (note only part of the wizard is visible)

Editing and Correcting Errors

Once the script has been created in the wizard, it can be edited. To edit the script right click on it and select edit. This will open the script in a text editor, generally Notepad. The script can also be edited using an IDE, but not directly. If the learner right clicks on the script and selects properties, those properties implemented in the wizard can be edited. Once the editing has been completed, it is important to make sure that no files were created. If files were created, they must be manually deleted, since they can cause potential errors when the script is run again.

Once the editing is completed and any files created, the script can be run again. When the script tool is run, it should provide an input box, in which the user will locate the input folder; this is done through the use of the browse button. Next the storage location should be specified and again can be located using the browse button. When the script has completed the operation, the new shapefiles should be located in the geodatabase. There should be a one to one correspondence to the old older shapefiles. It is important that none of the shapefiles used in the process be opened in ArcMap when the script is run, this can create a lock on the shapefile and thus the operation will not function properly.

Assignment 6.1

A geodatabase containing five counties from the same state of the same type, i.e. such as census tract boundaries. These are the only files contained in the geodatabase. In this operation the five counties will be merged together to form a single file regional geography. In addition, the created regional geography will be used to clip a river and a railroad files at a statewide geography. The resulting merged counties and the clipped files will be saved back to the same geodatabase.



The river and railroad files should be in the same file folder and contain no additional information. Make sure the files use appropriate naming convention such as: KY_Region_Louisville_Railroads.

Create a single tool that produces the clipping and merging. The tool will have two or three different inputs. One for the geodatabase, one for the river and railroad file folder and potential the output location. The required information for your instructor should include the script, the screen image of the input window, the contents of the geodatabase and output maps of the results.



JCTC_CIT 299_Module 2_Topic 6_Inputting_Data by Vincent A. DiNoto, Jr. is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to, accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership. This is an equal opportunity program. Assistive technologies are available upon request and include Voice/TTY (771 or 800-947-6644).