Building Geoprocessing Tools with Python: Beyond the Basics

Dave Wynne

2019 ESRI DEVELOPER SUMMIT
Palm Springs, CA

Building Geoprocessing Tools with Python: Beyond the Basics

Primrose B

This session will focus on creating polished, well-designed and useful geoprocessing tools with Python. We'll discuss validation in more depth, and cover debugging script tools. We'll also focus on strategies for organizing and distributing your toolbox code, and creating tools that behave well as geoprocessing services.

- Validation
- Code organization
- Debugging
- Packaging, services

- Materials at https://github.com/dWynne1/ds19-building-tools

Validation goals

- **Provides more control, enhanced experience**
  - **Parameter interaction**
  - **Calculate defaults**
  - **Enable or disable parameters**
  - **Setting errors and messages**

- **Defining output characteristics**
  - ***Chain tools in ModelBuilder***

Validation

- Mechanically, validation is about responding to changes in:

- `value` / `valueAsText`
  - Does a parameter have a value?
  - What is the value?
  - Properties of the data (`arcpy.Describe`)
- `altered`
  - Has the parameter been altered?
- `hasBeenValidated`
  - Has internal validation checked the parameter?

```
def updateParameters(self):
    """Modify the values and properties of parameters before internal
    validation is performed.  This method is called whenever a parameter
    has been changed."""

    if self.params[0].value:
        if not self.params[2].altered:
            extent = arcpy.Describe(self.params[0].value).extent
            if extent.width > extent.height:
                self.params[2].value = extend.width / 100.0
            else:
                self.params[2].value = extent.height / 100.0

    return
```

```
def updateMessages(self):
    """Modify the messages created by internal validation for each tool
    parameter.  This method is called after internal validation."""

    # Distance should never be negative
    if self.params[2].value <= 0.0:
        self.params[2].setErrorMessage(
            'Distance value cannot be a negative number')

    # If using percentages, distance must be less than 1.0
    elif self.params[3].value:
       if self.params[2].value > 1.0:
            self.params[2].setErrorMessage(
                'Percentages must be between 0.0 and 1.0')
```

Parameter tip

- On layers, use parameter.valueAsText over parameter.value
  - For layers, using `parameter.value` is expensive

- Describe accepts parameter objects directly
  - Using parameter object is faster (*especially in Pro*)

```
def updateParameters(self, parameters):
    """Modify the values and properties of parameters before internal
    validation is performed.  This method is called whenever a parameter
    has been changed."""

    if not parameters[0].altered and parameters[0].valueAsText:
        shp_type = arcpy.Describe(parameters[0]).shapeType
```

As of 10.4.1 and ArcGIS Pro 1.1

Script tool parameter properties tip

- Not all parameter properties are available when creating parameters in the UI
  - categories, enabled, controls (Pro), composite datatypes (10.x)

- But they can still be set in initializeParameters

Demo: Validation

Data from http://www.seanlahman.com/

Code organization – script tools

- Validation code lives inside the toolbox

- Hard to debug, edit, and manage
- Easier if code maintained outside the toolbox
  - Separate code into a Python file
  - Import back into the script tool
  - Use importlib.reload when actively working on validation code

Code organization – Python toolboxes

- A Python toolbox is recognized by it's file extension, .pyt
  - But we can't debug a .pyt directly
- Separate code into separate .py(s)
  - Functionally a Python toolbox is composed of a Toolbox class and tool classes
  - I maintain the Toolbox class in the .pyt, and separate the tools into separate modules
- Use importlib.reload when actively working/testing on imported code
  - And remove when finished

Demo: Toolbox/code organization

Debugging in ArcGIS Pro

- Debugging of Python-based tools has historically been difficult
- In particular, debugging validation required ‘creativity’:
  - Abusing parameter messaging
  - Temporary parameters to send information to
  - Logging
  - Message boxes

Debugging Python in Pro – prerequisites VS

New at Pro 2.1

- Visual Studio 2017 (Community version or up).
  - Visual Studio 2015 doesn't support debugging for Python 3.6, so it can't be used to attach ArcGIS Pro 2.1 which has been upgraded to Python 3.6.2.

- Python Development Tools package: https://github.com/Microsoft/PTVS
- Python code you want to debug must be in a .py file

http://esriurl.com/13927

Debugging Python in Pro – Getting started

- Ensure Python is initialized in Pro
  - If not already. If needed, open a Python-based tool or Python window
- In Visual Studio, under *Debug* menu, click *Attach to Process…*
  - Select Python as code type
  - In process list, click *Attach*
  - Once the status bar is orange, you're ready to go

Demo: Debugging tools

Organizing toolboxes as packages

- Create Python modules that play nice in ArcGIS
  - Easily distributable
  - Toolboxes appear as system toolboxes
  - Toolboxes are incorporated into arcpy
  - Supports dialog side-panel help and localized messages

http://esriurl.com/13926

Demo: Toolboxes and Python packages

Publishing GP services

Preparation – workflows

- **Service won't publish if the tool won't run**
  - **Check the Results window in Pro or Desktop for any errors**

- **Opt for Asynchronous over Synchronous for medium to large tools**
  - **This option is available in the Publishing Wizard**
  - **“Async” tools need to be queried, while “Sync” will run to completion**

Publishing GP services

Preparation – data

- Avoid hard coded data paths and dependencies
  - Favor input parameters which a user can customize
  - my_param = arcpy.GetParameterAsText()
- If you have to use a “hard coded” data path, build it dynamically
  - Utilize os.path.join(“root directory”, “file_name”) so the tool works on any OS
  - Use sys.path.append(“server’s path to the module”) if needed
- Write intermediate/temporary output data to memory

Publishing Tips and Tricks

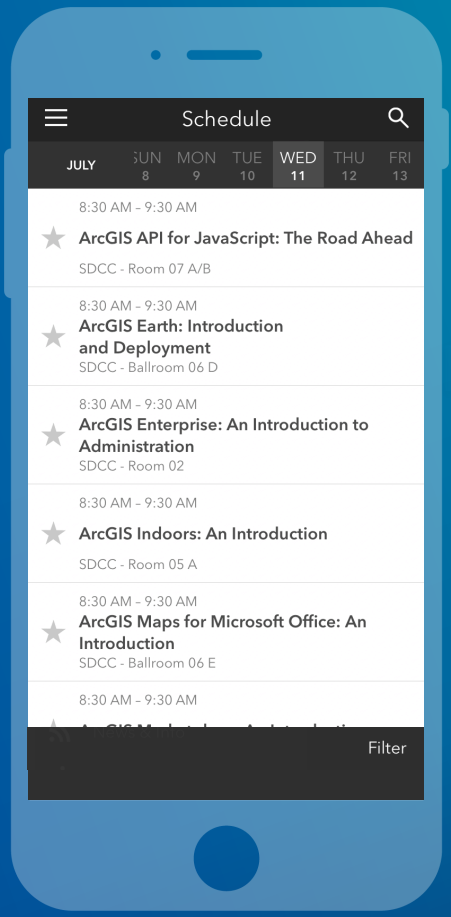
- Project Data is consolidated; aka copied to the Data Store
  - Data is found in any directory in the script and Table of Contents
  - Data referenced from the server (e.g. Data Store) is not consolidated
  - Python recursively searches directories for data to consolidate
- Consider using a Geoprocessing Package to store Project Data
  - Great for offline debugging and development
- Tool validation will be published, and executed by SubmitJob()
  - The validation occurs server-side, and should act the same, but test it!


esri
THE
SCIENCE
OF
WHERE

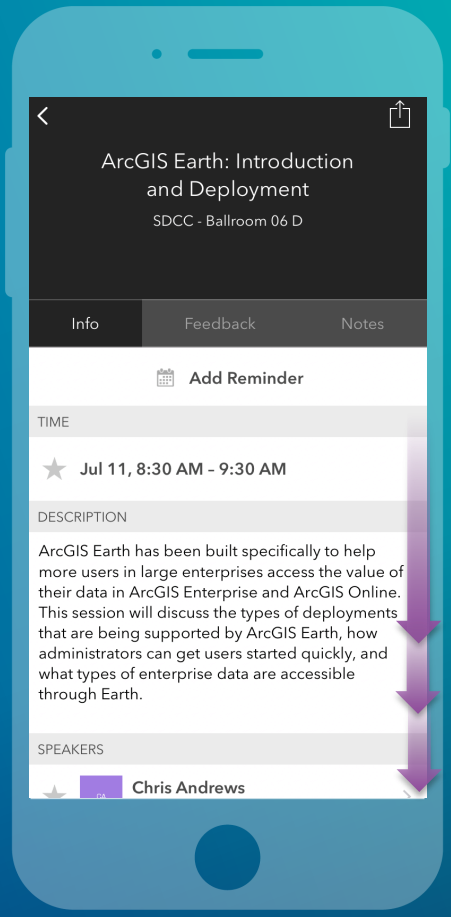
Please Take Our Survey on the App

Download the Esri Events app and find your event

Select the session you attended



Scroll down to find the feedback section



Complete answers and select “Submit”