Python:
Developing Geoprocessing Tools

David Wynne

Python: Developing Geoprocessing Tools

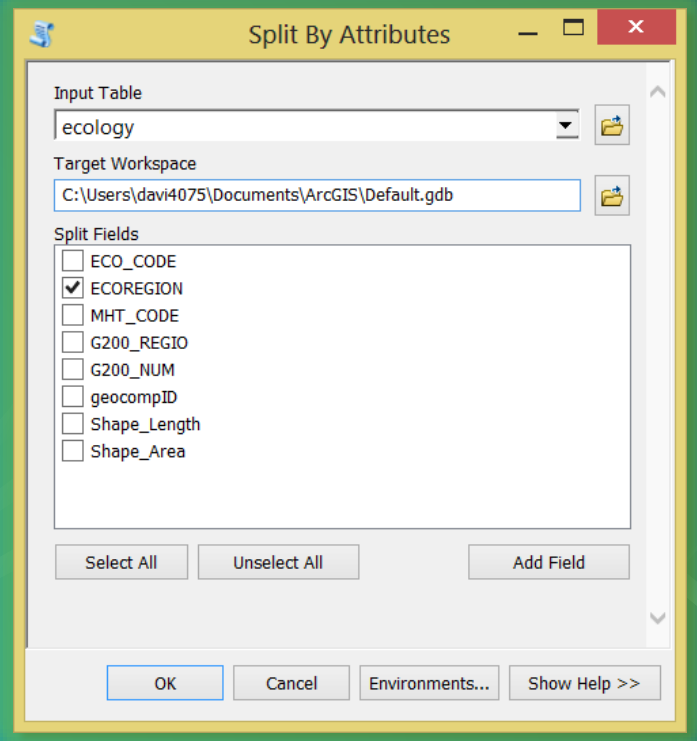
Join us as we step through the process of creating geoprocessing tools using Python. Using both script tools and Python toolboxes as examples, this workshop will highlight the important decisions in making fully functional geoprocessing tools.

Categories - Geoprocessing, Intermediate, Esri Technical Session

The geoprocessing tool

Validation

1. Update parameters
2. *Internal validation*
3. Update messages




```
'''
Python code

  Tool-specific behaviors

  1. Receive arguments
  2. Tool messages
  3. Progressor
  4. Cancellation behaviors
  5. Send arguments

'''

import arcpy

# More code
```

Geoprocessing Tool Commandments

Thou shall ...

… have unique parameter names within the tool

… keep the cost of validation to a minimum

… always have an output, even if it must be derived

… populate all output data elements within validation

… not test the validity of any derived value within validation

… have a filter list for every Boolean

… test the function from a script, a model, a dialog, and the command line

Script tools and Python toolboxes

- Using Python, we can build tools in two ways

- Script tools
  - Source is Python
  - Parameters defined through a wizard
  - Validation is Python (stored in the toolbox)
- Python toolboxes
  - Source, parameters and validation are all written in Python

- *Remember: A tool is a tool*

Script tools and Python toolbox validation

Script tools

```python
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')

    return
```

Python toolbox tools

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

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

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

- **Only access of parameters is different**

Parameters

- **Parameters define how we interact with a tool**

Key characteristics

Datatype	Feature layer, raster layer, table view, strings, floats, Booleans, …
Direction	Input, output
Parameter type	Required, optional, *derived

- **Derived are returned outputs that are determined within your code**
- ***Others: name (and label), multivalue, default, environment, filter, dependencies, symbology***

```python
def getParameterInfo(self):
    """Define parameter definitions"""
    param0 = arcpy.Parameter(displayName = "Input Features",
                             name = "in_features",
                             datatype = "GPFeatureLayer",
                             parameterType = "Required",
                             direction = "Input")
    param1 = arcpy.Parameter(displayName = "Statistics Field",
                             name = "stat_fields",
                             datatype = "GPValueTable",
                             parameterType = "Required",
                             direction = "Input")

    param1.parameterDependencies = [param0.name]
    param1.columns = [["Field", "Field"], ["String", "Statistics Type"]]
    param1.filters[0].list = ["Short", "Long", "Double", "Float"]
    param1.filters[1].type = "ValueList"
    param1.filters[1].list = ["SUM", "MIN", "MAX", "STDDEV", "MEAN"]

    param2 = arcpy.Parameter(displayName = "Output Table",
```

Getting started

Parameters – Validation

- As parameters are entered and modified, validation responds and interacts

- Tools validate parameters in two ways

1. Basic (“free”) validation, such as:
   - Does the input (or output) exist?
   - Is it the right data type?
   - Does this value match its filter?
   - Have all required parameters been supplied?
2. Additional rules and behavior you add

Validation

- Provides more control
  - Parameter interaction
  - Calculate defaults
  - Enable or disable parameters
- Setting errors and messages
- Defining output characteristics
  - Chain tools in ModelBuilder

Validation

- Validation is about responding to changes in:

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

```python
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 parameters[0].value:
        if not parameters[2].altered:
            extent = arcpy.Describe(parameters[0].value).extent
            if extent.width > extent.height:
                parameters[2].value = extent.width / 100.0
            else:
                parameters[2].value = extent.height / 100.0

    return
```

```python
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 parameters[0].value:
        p = feedparser.parse(parameters[0].valueAsText)
        if p['bozo'] == 0:  # Successful read
            entry = p.entries[0]
            f_names = entry.keys()
            f_names.remove('georss_point')
            f_names.remove('georss_elev')
            parameters[2].filter.list = f_names
```

Parameter tip!

- Describe accepts parameter objects directly
  - For layers, using `parameter.value` is expensive
  - Using parameter object is faster

```
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].value:
        shp_type = arcpy.Describe(parameters[0].value).shapeType
```

```
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

updateMessages

- Control over messages
  - Evaluate system messages, and relax or change
  - Add your own messages or errors based on your own criteria

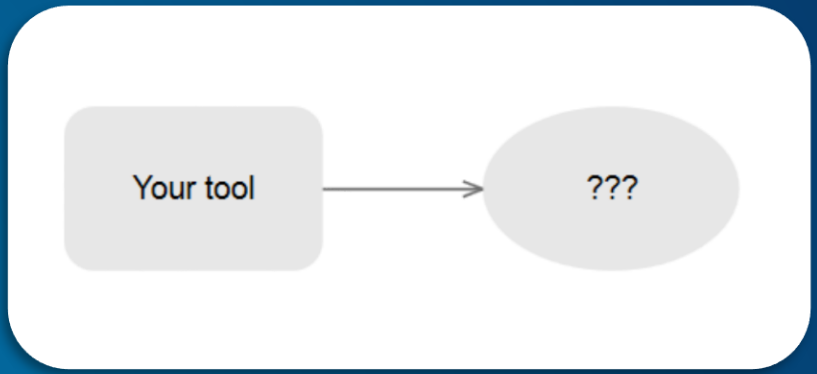
```
def updateMessages(self):
    """Modify the messages created by internal validation for each tool
    parameter.  This method is called after internal validation."""

    # 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')

    return
```

- `message`
- `clearMessage()`
- `hasError()`
- `hasWarning()`
- `setErrorMessage(message)`
- `setWarningMessage(message)`
- `setIDMessage(message_type, message_ID, {add_argument1}, {add_argument2})`

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

    p0 = parameters[0]

    # ERROR 000800: The value is not a member of ...
    if p0.hasError() and p0.message.find('000800') > -1:
        p0.setWarningMessage(p0.message)

    return
```

Validation: ModelBuilder

- Describe outputs for chaining in ModelBuilder
- By updating `schema` of outputs, subsequent tools can see pending changes prior to execution




```
parameters[1].parameterDependencies = [parameters[0].name]
parameters[1].schema.clone = True
parameters[1].schema.geometryTypeRule = 'AsSpecified'
parameters[1].schema.geometryType = 'Point'
parameters[1].schema.fieldsRule = 'FirstDependencyFIDs'

id_field = arcpy.Field()
id_field.name = 'FID_1'
id_field.type = 'Integer'

parameters[1].schema.additionalFields = [id_field]
```

```python
class AgolTasks(object):
    """ Class to handle ArcGIS Online tasks."""

    def __init__(self, username, password, portal_url):
        self.username = username
        self.password = password
        self.portal_url = portal_url
        self.rest_url = "%s/sharing/rest" % self.portal_url
        self.token = self.get_token()

    @staticmethod
    def rest_response(request):
        """ Sends the request to REST and returns the REST response as json."""
        with urlopen(request) as response:
            json_data = response.read().decode("utf-8")
            json_data = json.loads(json_data, encoding="utf-8")
        if json_data:
            return json_data

    def get_token(self):
        """ Returns a token for use in ArcGIS Online. """
```

Validation

Making your Python code work as a tool

1. Parameters are received
   - Script tool – `GetParameter` or `GetParameterAsText`
   - Python toolbox – `parameter.value` or `.valueAsText`

2. arcpy internals
   - Messages – `arcpy.AddMessage`, `AddWarning`, `AddError`
   - Progressor
   - Optionally, control actions after a cancellation

3. Derived values, if any, are pushed back to the tool

```python
'''
Python code

  Tool-specific behaviors

  1. Receive arguments
  2. Tool messages
  3. Progressor
  4. Cancellation behaviors
  5. Send arguments

'''

import arcpy

# More code
```

Cancellation behavior

- By default, when a tool is cancelled…
  - … code will be cancelled after the current line
- Can respond to the cancellation using environments:
  - `arcpy.env.autoCancelling`
  - `arcpy.env.isCancelled`

* As of 10.4.1 and ArcGIS Pro 1.1

```python
arcpy.env.autoCancelling = False


class CustomCancelException(Exception):
    """Custom exception for cancellations"""
    pass


def my_function(tables, output):
    temp_tables = []
    try:
        for table in tables:
            temp_tables.append(arcpy.CopyRows_management(table, '#')[0])

            # If isCancelled is True this means that the cancel button
            # has been pressed
            if arcpy.env.isCancelled:
                raise CustomCancelException('Tool has been cancelled')

        arcpy.Merge_management(tables, output)

    except CustomCancelException as err:
        arcpy.AddError(err)
    finally:
        # Clean up intermediate data
        if temp_tables:
            for temp_table in temp_tables:
                arcpy.Delete_management(temp_table)
```

```python
def my_function(tables, output):
    temp_tables = []
    try:
        for table in tables:
            temp_tables.append(arcpy.CopyRows_management(table, '#')[0])

            # If isCancelled is True this means that the cancel button
            # has been pressed
            if arcpy.env.isCancelled:
                raise CustomCancelException('Tool has been cancelled')

        arcpy.Merge_management(tables, output)

    except CustomCancelException as err:
        arcpy.AddError(err)
    finally:
        # Clean up intermediate data
        if temp_tables:
            for temp_table in temp_tables:
                arcpy.Delete_management(temp_table)
```

Making it all work together

Parameters, cancellations, messages, progressor, encryption

Distributing your toolboxes in Python modules

- 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://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/extending-geoprocessing-through-python-modules.htm

- "Deploying Your Geoprocessing Tools as Python Modules"
  - Wednesday, Demo Theater 1, 4:00-4:30 pm

```python
class AgolTasks(object):
    """ Class to handle ArcGIS Online tasks."""

    def __init__(self, username, password, portal_url):
        self.username = username
        self.password = password
        self.portal_url = portal_url
        self.rest_url = "%s/sharing/rest" % self.portal_url
        self.token = self.get_token()

    @staticmethod
    def rest_response(request):
        """ Sends the request to REST and returns the REST response as json."""
        with urlopen(request) as response:
            json_data = response.read().decode("utf-8")
            json_data = json.loads(json_data, encoding="utf-8")
        if json_data:
            return json_data

    def get_token(self):
        """ Returns a token for use in ArcGIS Online. """
```

Distributing your toolbox

Additional considerations

- ArcGIS Pro
  - Use Analyze Tools For Pro geoprocessing tool to aid in migration
  - Turn off the Pro's tool analyzer setting—extra overhead analyzing your tools

- Tools as services
  - "Creating Geoprocessing Services With Python Script Tools"
    - Wednesday, Demo Theater 3, 5:30-6:00 pm

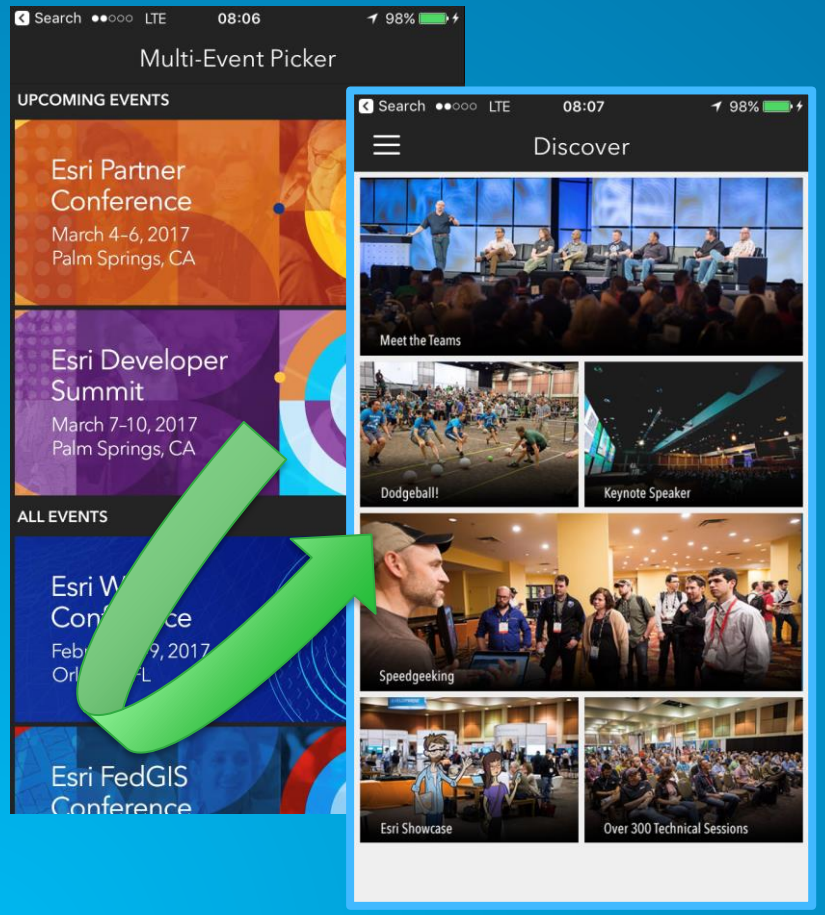
ctypes

- Supports access to other languages (C, C++)
  - In Python's standard library
- Use `ctypes` to access/make use of a dll
- Provides wrappers around C libraries, functions, variables
- Advantages
  - Familiarity, performance
  - Do what you need through `ctypes`, do the rest through Python
  - Development time
  - Protect your intellectual property
  - DLL/Typelib/Assembly registration

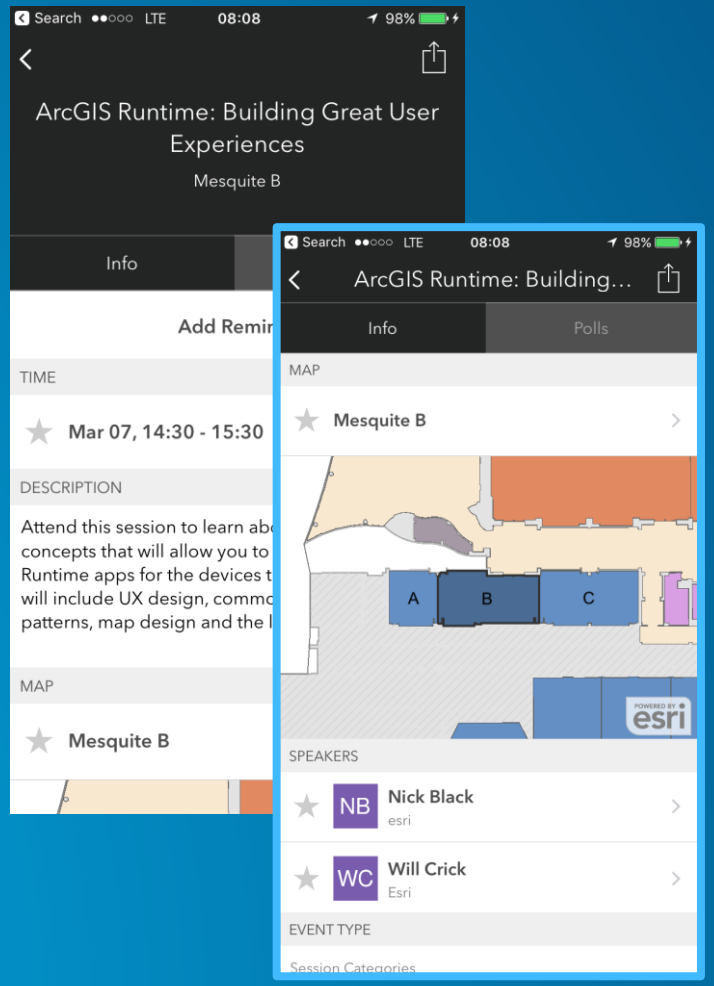
- https://github.com/ghisprince/arcpy-cpp-interop

Please Take Our Survey! – No more memorizing Session ID numbers!! ☺

Download the Esri Events app and find your event

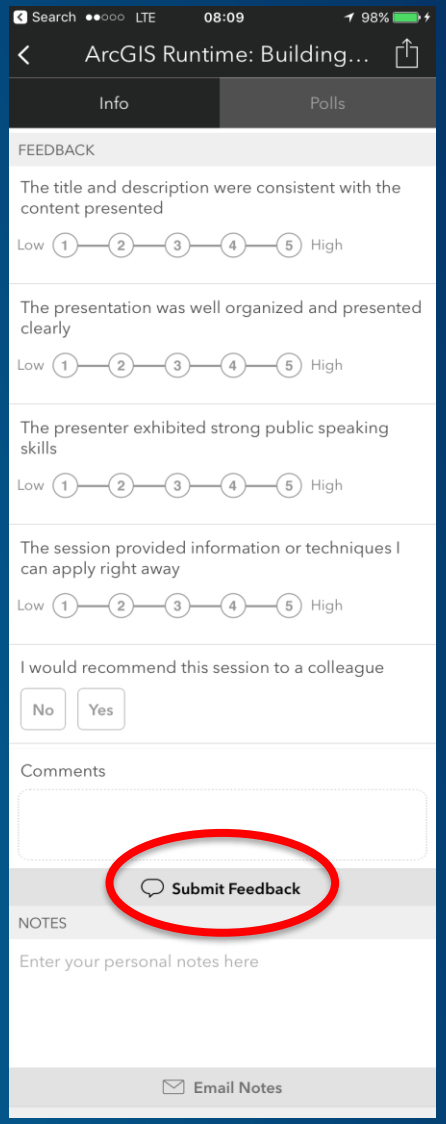


Select the session you attended



Scroll down to the "Feedback" section

Complete Answers, add a Comment, and Select "Submit"



esri

THE SCIENCE OF WHERE