Esri Developer Summit

March 8–11, 2016 | Palm Springs, CA

Python: Developing Geoprocessing Tools

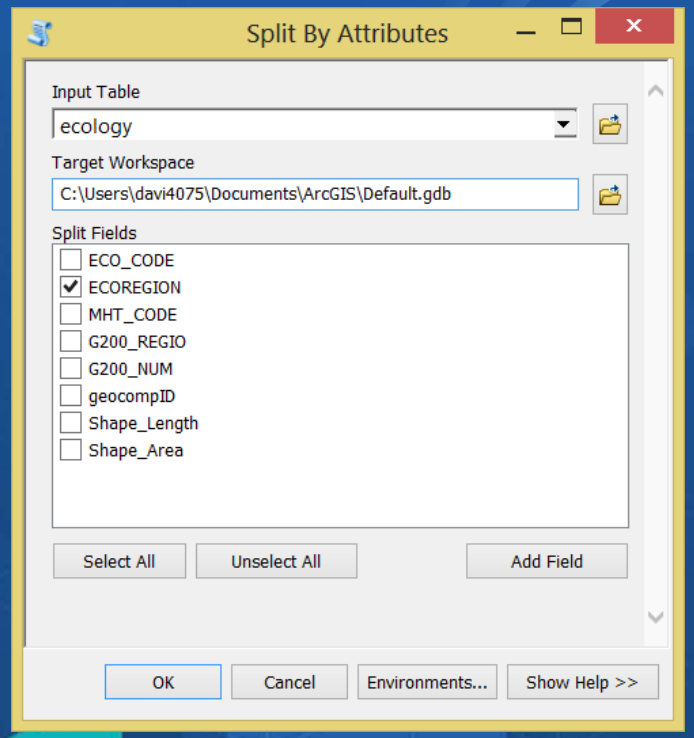
David Wynne

Jon Bodamer

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. Distributing your toolboxes and extending geoprocessing using Python modules will also be discussed.

Deconstructing a geoprocessing tool



Validation

1. **UpdateParameters**
2. **Internal validation**
3. **UpdateMessages**

```
'''
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 coded value domain for every Boolean

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

Script tools vs Python toolboxes

- **Using Python, we can build tools in two ways**

	Script tools	Python toolboxes
Origin	• 9.0	• 10.1
Coding	• Source is Python • Parameters through wizard • Validation is Python (but stored in the toolbox)	• Entirely in Python

- ***Remember: A tool is a tool***

Difference between Python toolboxes and script tools validation

```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 r

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

```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
```

Parameters and validation

Parameters

- **Parameters are how you interact with a tool**

Key characteristics

Datatype	Feature layers, raster layers, table view Strings, floats, Booleans
Direction	Input, output
Parameter type	Required, optional, *derived

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

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

```
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",
```

Python toolbox – Statistics tool

Jon Bodamer

Validation

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

Validation

- Responding to
  - value / valueAsText
  - altered
  - hasBeenValidated

```
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
```

```
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
```

updateMessages

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

```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
```

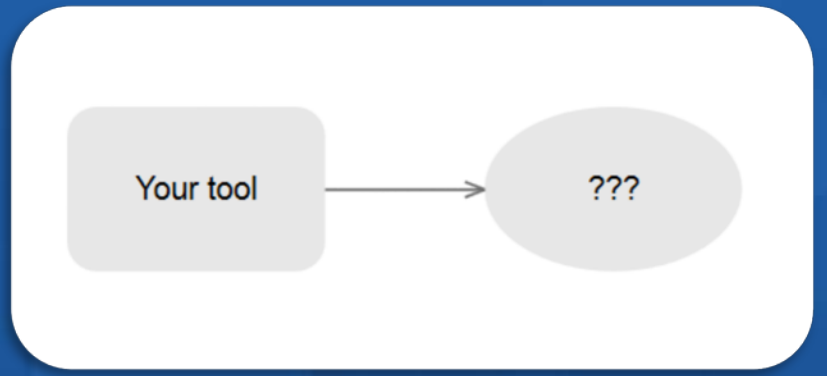
```python
def updateMessages(self, parameters):
    """Modify the messages created by internal validation for each tool
    parameter.  This method is called after internal validation."""

    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
```

Messaging

message	setErrorMessage()
clearMessage()	setIDMessage()
hasError()	setWarningMessage()
hasWarning()	

Validation: ModelBuilder

- Describe outputs for chaining in ModelBuilder
- Data variables in ModelBuilder include descriptions of data
- By updating the schema, 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]
```

```
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. """
```

Script tool – Upload to ArcGIS online

Jon Bodamer

Under the hood

Making your Python code work as a tool

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

2. arcpy internals
   - Messages – arcpy.AddMessage / AddWarning / AddError …
   - Progressor (step or ‘cylon eye’)
   - Control actions (or not) after a cancellation

3. Any derived values are pushed back to the tool

Cancellation behavior

- By default, when a tool is cancelled…
  - … code will be cancelled after the current line
- Using new environments, can respond to the cancellation
  - arcpy.env.autoCancelling
  - arcpy.env.isCancelled
- * added at 10.4, Pro 1.1

```
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)
```

```
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)
```

Cancellation behavior

David Wynne

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

Distributing tools

Distributing your tools in Python modules

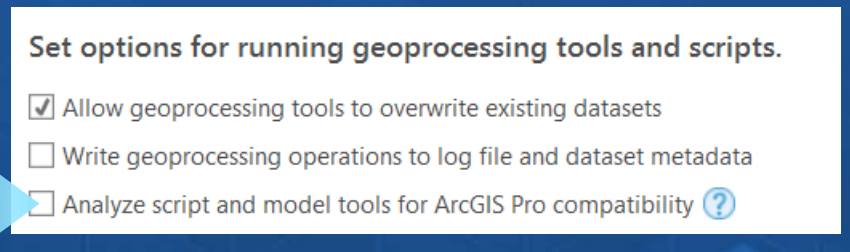
- Create Python modules that play nice in ArcGIS

- Easily distributable
- Toolboxes appear as system toolboxes
- Toolboxes are incorporated into arcpy
- Supports dialog side-panel help and localized messages

- Requires a specific structure

Other considerations

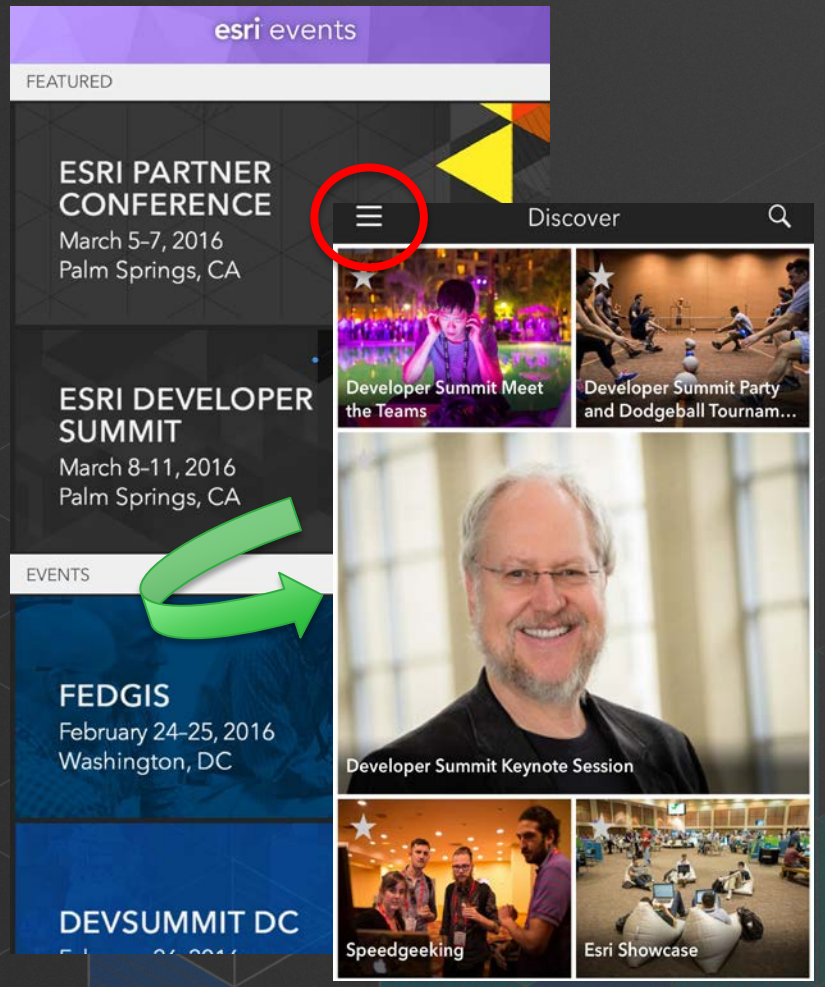
- **ArcGIS Pro**
  - **Use Analyze Tools For Pro tool to aid in migration**
  - **Turn off the tool analyzer setting. Extra overhead analyzing your tools**



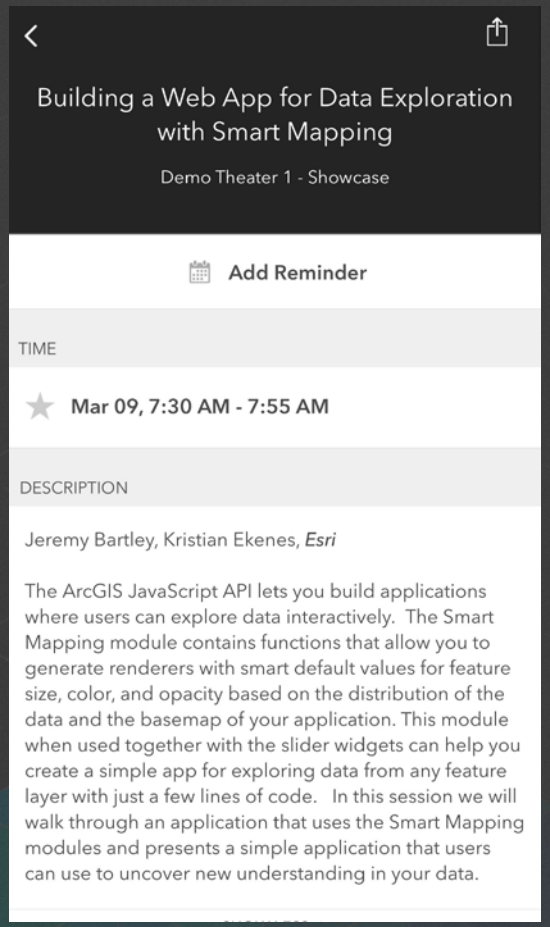
- **Tools as services**
  - **Best practices → Today, Demo Theater 1, 2:30-3:30 pm**

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

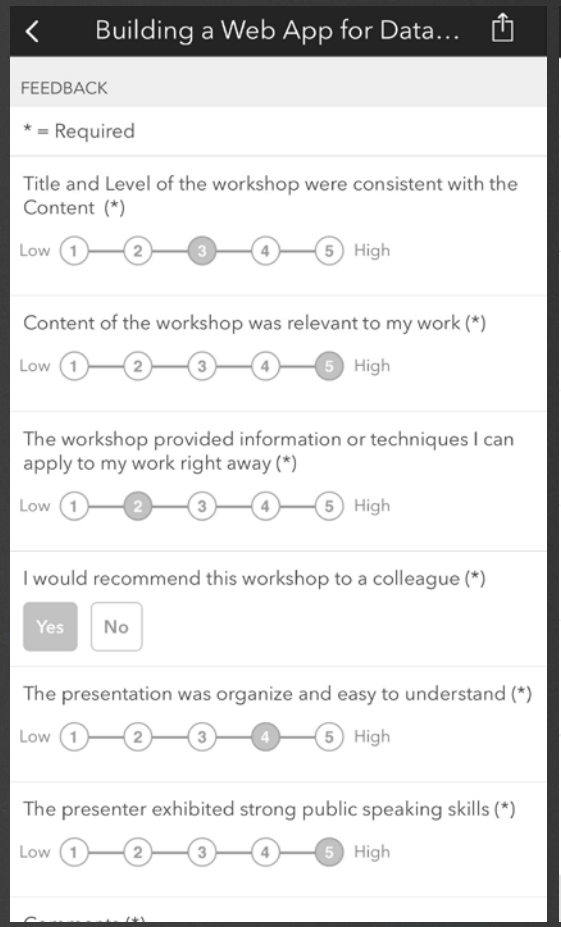
Download the Esri Events app and find your event



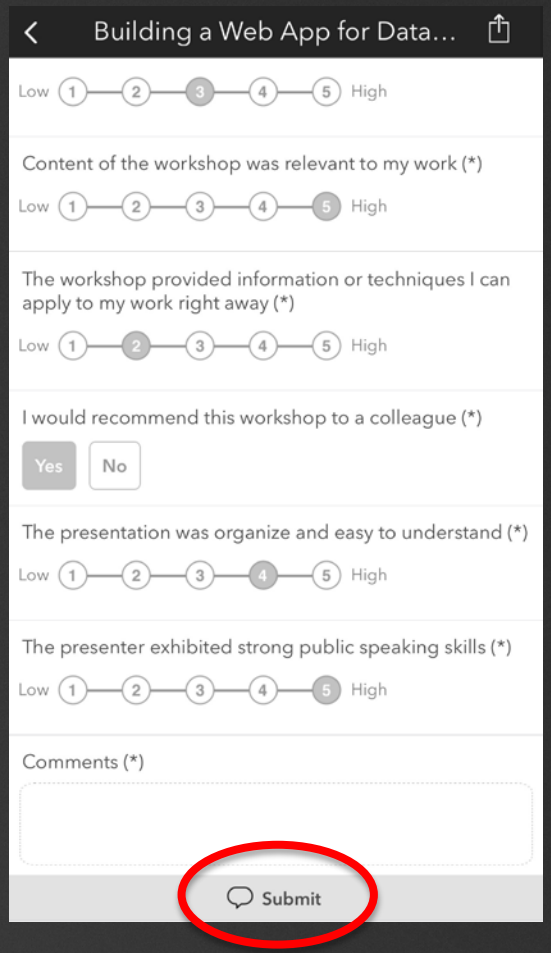
Select the session you attended



Scroll down to the "Feedback" section



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




esri®