Creating Geoprocessing Tools

Dave Wynne

Kimberly McCarty

Ghislain Prince

2024 ESRI DEVELOPER SUMMIT

Creating Geoprocessing Tools

Geoprocessing tools provide an integrated and familiar experience for analysts to perform data management and spatial analysis in ArcGIS. By creating a geoprocessing tool, others can reuse your methods and workflows with their own data without requiring them to learn or use code. This session will show you how to create and design geoprocessing tools, choose appropriate parameters, write custom validation routines to fine-tune the user experience, and author help so the tool can be used correctly by others. Approaches for debugging, error handling and messaging will also be covered.

Session Type: Technical Session

Access Type: In-Person, Recorded, Live stream

Topic: Spatial Analysis and Data Science

Session Level: Beginner

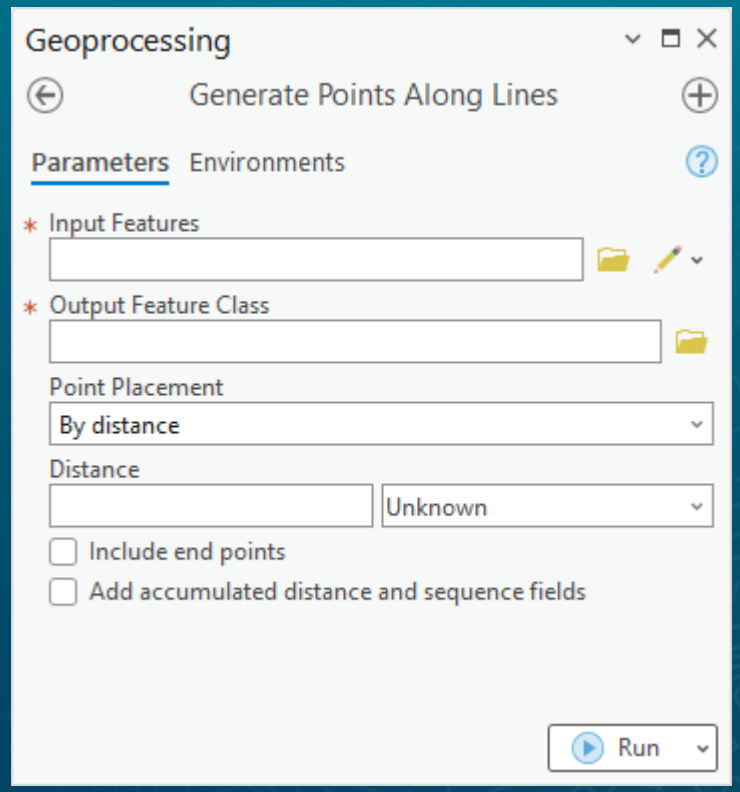
https://github.com/dWynne1/ds2024-creating-gp-tools

Why we build tools

- Extend Pro
- Organize functionality

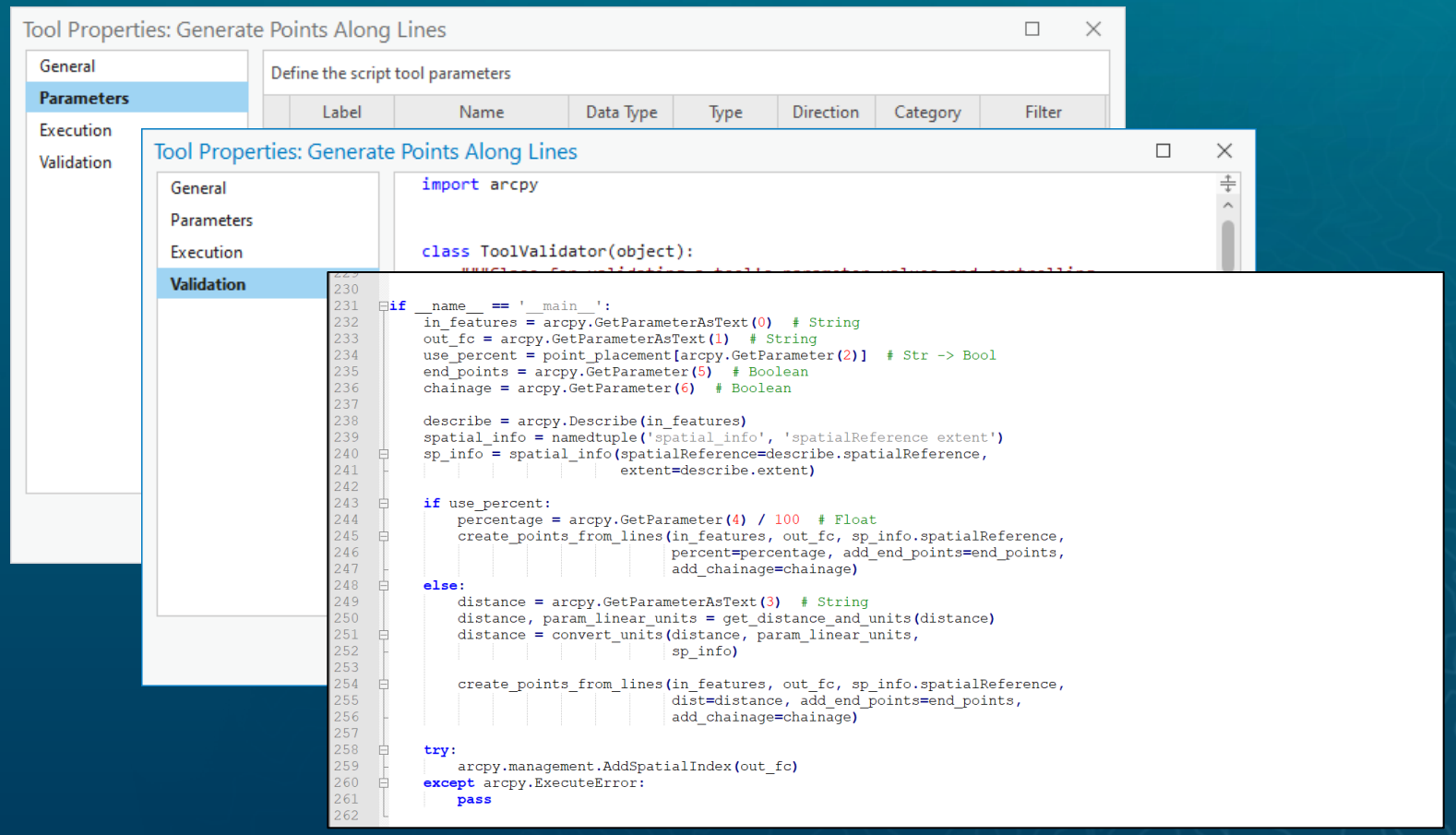
- Take advantage of Pro's geoprocessing framework

- Multiple ways to use a tool
  - Geoprocessing pane
  - Python
  - ArcGIS Pro SDK (.NET)
  - ModelBuilder
  - Share as a geoprocessing service



Tool structure

- Parameters
- Validation code
- Source code



Toolboxes

- Tools are organized in a toolbox
- We can build Python-based tools in two ways:

- Script tools – ArcGIS toolbox (.atbx) or Legacy toolbox (.tbx)
  - Parameters defined through the Pro UI
  - Validation is Python code
  - Source is Python code

Toolboxes

- Tools are organized in a toolbox
- We can build Python-based tools in two ways:

- Python toolbox (.pyt)
  - Parameters are Python code
  - Validation is Python code
  - Source is Python code

```python
import arcpy


class Toolbox(object):
    def __init__(self):
        self.label =  "Sinuosity toolbox"
        self.alias  = "sinuosity"

        # List of tool classes associated with this toolbox
        self.tools = [CalculateSinuosity]


class CalculateSinuosity(object):
    def __init__(self):
        self.label       = "Calculate Sinuosity"
        self.description = "Sinuosity measures the amount that a river " + \
                           "meanders within its valley, calculated by " + \
                           "dividing total stream length by valley length."

    def getParameterInfo(self):
        #Define parameter definitions

        # Input Features parameter
        in_features = arcpy.Parameter(
            displayName="Input Features",
            name="in_features",
            datatype="GPFeatureLayer",
            parameterType="Required",
            direction="Input")

        in_features.filter.list = ["Polyline"]

        # Sinuosity Field parameter
        sinuosity_field = arcpy.Parameter(
            displayName="Sinuosity Field",
            name="sinuosity_field",
            datatype="Field",
            parameterType="Optional",
            direction="Input")

```

ArcGIS toolbox format (.atbx)

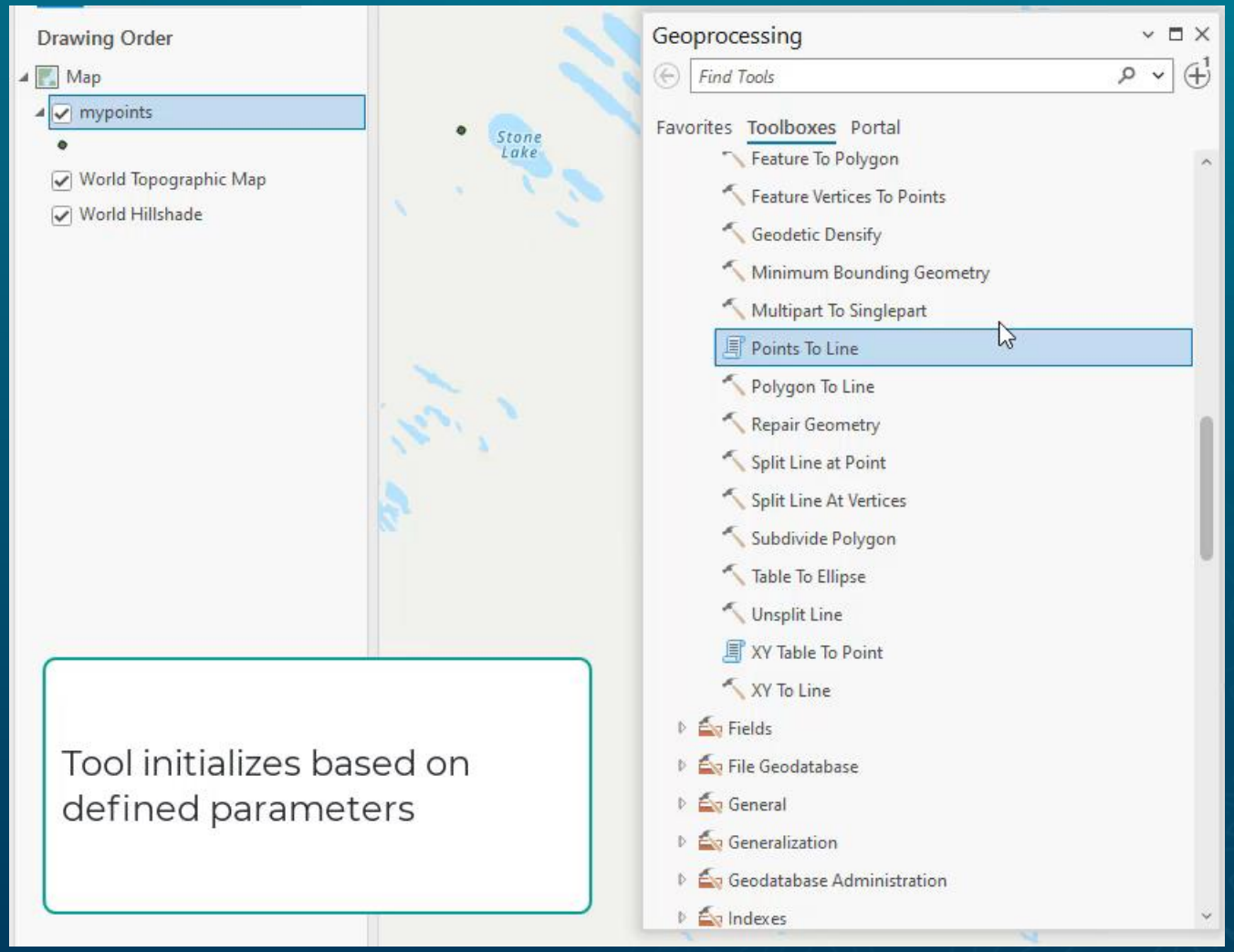
- JSON-based format with an open specification
- Stores tools, scripts, and models

- Introduced at Pro 2.9

- Same look and feel as the traditional Legacy toolbox (.tbx) format

- Better cross-release compatibility and persistence

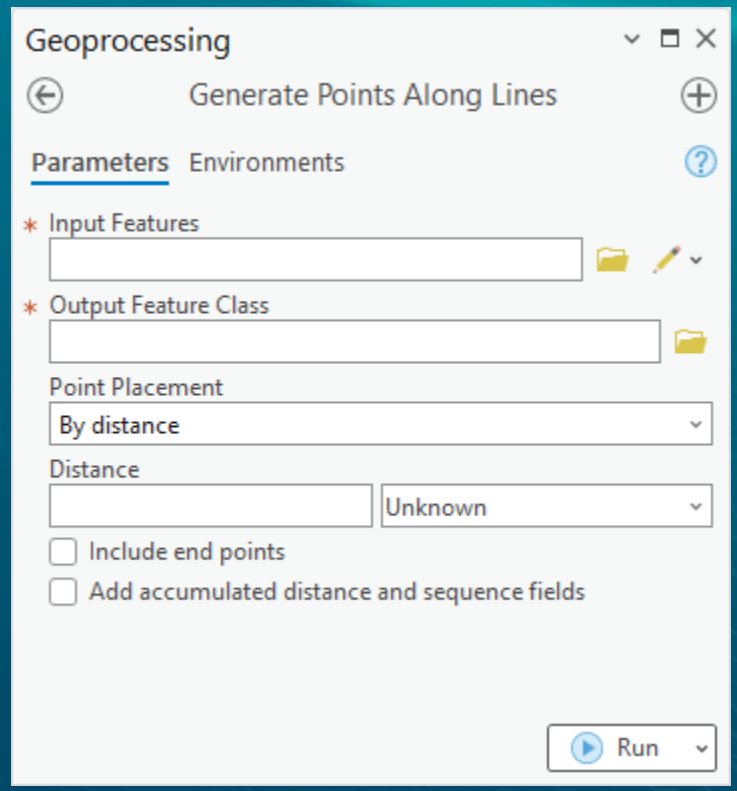
How a tool works



Parameters

- Parameters are how you interact with a tool

- Parameters provide simple rules
  - Does an input exist?
  - Is the input the right type?
  - Is this value an expected keyword?



Define the script tool parameters

	Label	Name	Data Type	Type	Direction	Category	Filter	Dependency	Default	Environment	Symbology
0	Input Feat...	Input_Features	Feature La...	Required	Input		Feature Type				
1	Output Fe...	Output_Feature_Class	Feature Cl...	Required	Output			Input_Feat...			
2	Point Plac...	Point_Placement	String	Required	Input		Value List		DISTANCE		

	Label	Name	Data Type	Type	Direction	Category	Filter
0	Input Features	in_features	Feature La...	Required	Input		
1	Output Rotated Features	out_rotated_features	Feature Cl...	Required	Output		
2	Rotation Angle	rotation_angle	Double	Required	Input		
3	Feature Set or Point	feature_set_or_point	String	Optional	Input		Value List
4	Rotate Feature Set	rotate_feature_set	Feature Set	Optional	Input		Feature Type
5	Rotate Point	rotate_point	Point	Optional	Input		
*			String	Required	Input		

Parameters

Accessing parameters in the code

- To access parameter values from arcpy, use:
  - GetParameterAsText – Value returned as a string
  - GetParameter – Value returned as a Python or ArcPy type (best for Boolean and numeric types)

- Use these functions with a 0-based index number:
  - `arcpy.GetParameterAsText(6)`
- Or starting at 3.2, you can also use a parameter name:
  - `arcpy.GetParameterAsText("dissolve_field")`

Communication within the tool (messages)

- Relay information using arcpy message functions
  - AddMessage
  - AddWarning
  - AddError
  - AddIDMessage – Supports Esri ID codes

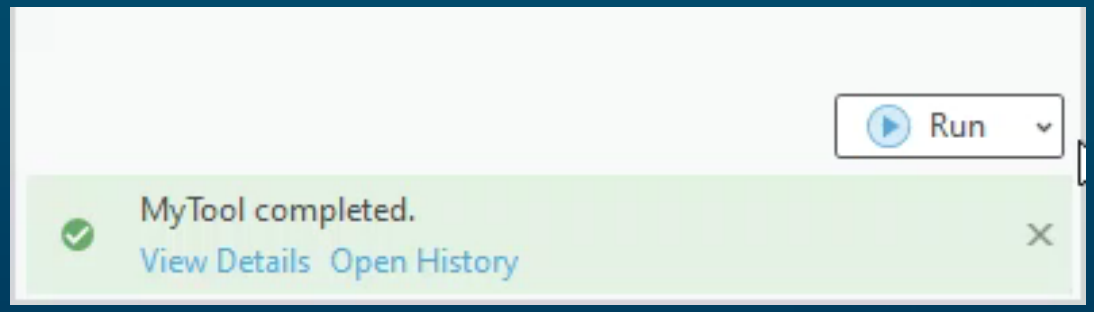
- Note: Error messages are only messages, they will not end the script
  - Best to exit your code soon after, such as Python's sys.exit()

```
if __name__ == '__main__':

    in_features = arcpy.GetParameterAsText(0)
    out_features = arcpy.GetParameterAsText(1)
    angle = radians(arcpy.GetParameter(2))
    option = arcpy.GetParameterAsText(3)
    feature_set = arcpy.GetParameter(4)
    point = arcpy.GetParameterAsText(5)

    create_feature_class(in_features, out_features)

    if option == 'POINT':
        # Create a geometry from the x,y-coordinates
        rotation_point = [float(i) for i in point.split(' ')]

    elif option == 'FEATURESET':
        # Extract the point from the feature set
        with arcpy.da.SearchCursor(feature_set, 'SHAPE@XY') as cursor:
            for row in cursor:
                rotation_point = row[0]
                # Use the first point, skip any others
                break
```

Tool source code

Communication within the tool (progressor)

- Relay simple information to the Geoprocessing pane
- Can provide messages and step increments
  - SetProgressor
  - SetProgressorPosition
  - SetProgressorLabel
  - ResetProgressor




```
5   feature_count = int(arcpy.management.GetCount(in_features)[0])
6
7   # Set up the progressor to update every 5% of the features
8   if feature_count > 20:
9
10      arcpy.SetProgressor(
11          type="STEP",
12          message="Processing features ... ",
13          min_range=0,
14          max_range=100,
15          step_value=5)
16
17      step = feature_count // 20
18
19  for i in range(1, feature_count + 1):
20
21      # Your data processing goes here
22
23
24      if feature_count > 20:
25          if i % step == 0:
26              # Update the progressor message
27              arcpy.SetProgressorLabel(
28                  "Processing feature {0}...".format(i))
29
30              # Update the progressor position
31              arcpy.SetProgressorPosition()
```

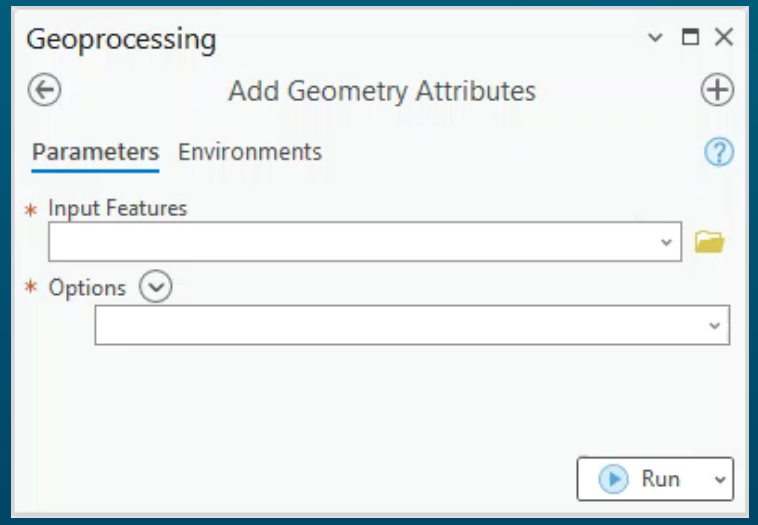
Parameter validation

- Parameters provide some simple ‘free’ validation
- Refine your tool’s behavior with additional validation
  - Parameter interaction
  - Calculate defaults
  - Enable or disable parameters
  - Set parameter errors and messages
  - Define characteristics of your output (for ModelBuilder)
- Validation runs every time a parameter is modified

```
class ToolValidator:
    """
    Class to add custom behavior and properties to
    the tool and tool parameters.
    """

    ...

    def updateParameters(self):
        """Modify parameter values and properties."""

        return

    def updateMessages(self):
        """Customize messages for the parameters."""

        return
```

Validation – updateParameters

- updateParameters allows you to change specific parameter characteristics
  - Including values, filters, enabled, etc.

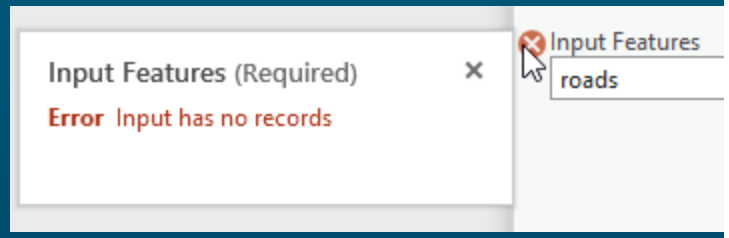



```
def updateParameters(self):
    """Modify parameter values and properties."""

    in_features = self.params[0].value
    if in_features:
        shape_type = arcpy.Describe(in_features).shapeType
        if shape_type == 'Polygon':
            self.params[1].filter.list = ['AREA', 'LENGTH', 'CENTROID']

        elif shape_type == 'Polyline':
            self.params[1].filter.list = ['LENGTH', 'CENTROID']

        else:
            self.params[1].filter.list = ['CENTROID']
    else:
        self.params[1].filter.list = ['AREA', 'LENGTH', 'CENTROID']

    return
```

- Parameters in validation are accessed by a 0-based index.
- Or starting at 3.2 for script tools, the parameter name

Validation – updateMessages

- updateMessages allows you provide warnings or errors before running the tool
- Provides information in Geoprocessing pane in real time




```
def updateMessages(self):
    """Customize messages for the parameters."""

    in_features = self.params[0].value

    if in_features:
        selection = arcpy.Describe(in_features).FIDSet

        if not selection:
            self.params[0].setErrorMessage('Input has no selection')

    return
```

- Note: only use message methods in updateMessages

```python
class ToolValidator:
    """Class to add custom behavior and properties to the tool and tool parameters."""

    def __init__(self):
        """Set self.params for use in other functions"""
        self.params = arcpy.GetParameterInfo()

    def initializeParameters(self):
        """Customize parameter properties.
        This gets called when the tool is opened."""

        return

    def updateParameters(self):
        """Modify parameter values and properties.
        This gets called each time a parameter is modified, before
        standard validation."""

        if self.params[3].valueAsText == 'POINT':
            self.params[4].enabled = False
            self.params[4].value = None

            self.params[5].enabled = True
```

Validation (and metadata)

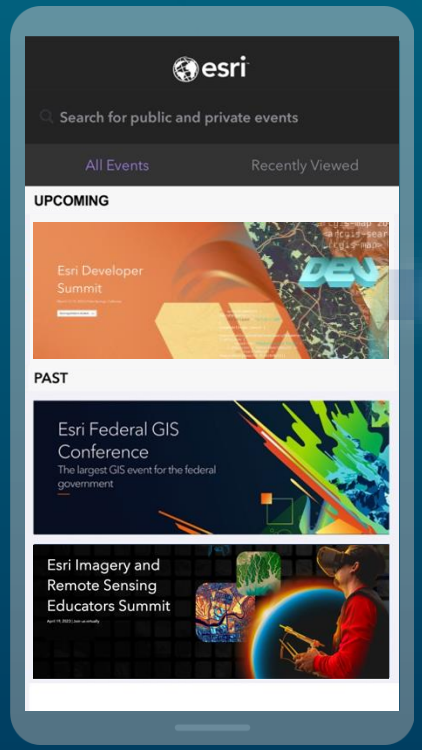
Symbology

- Use a layer file to set a parameter's symbology property
- Or, use the postExecute validation method *(new at 3.0)*
  - Runs when a tool completes
  - Use the arcpy.mp module

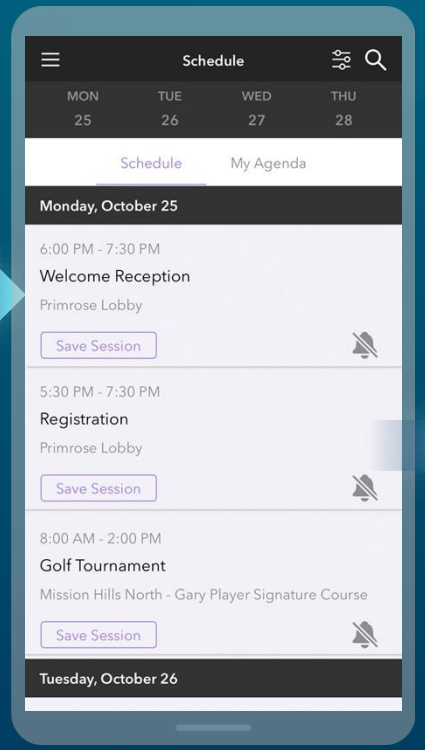
```
def postExecute(self):
    """This method takes place after outputs are processed and added to the display."""

    try:
        project = arcpy.mp.ArcGISProject('CURRENT')
        active_map = project.activeMap

        if active_map:
            out_layer = active_map.listLayers(os.path.basename(self.params[0].valueAsText))[0]

            symbology = out_layer.symbology
            symbology.updateRenderer('SimpleRenderer')
            symbology.renderer.symbol.applySymbolFromGallery('Airport')
            symbology.renderer.symbol.size = 12
            out_layer.symbology = symbology

    except Exception:
        pass

    return
```

Please Share Your Feedback in the App

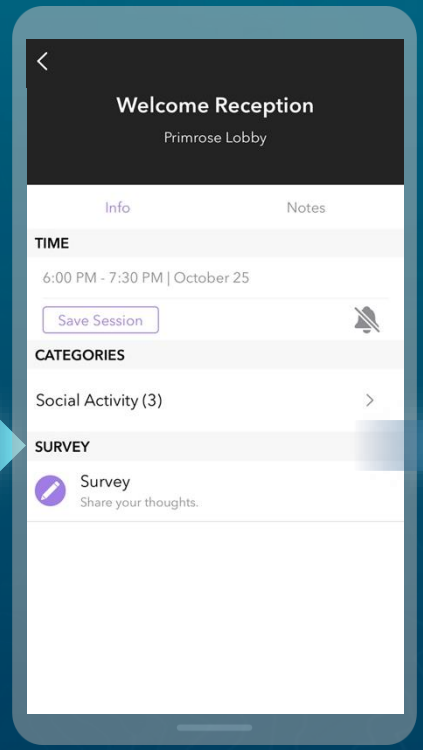
Download the Esri Events app and find your event



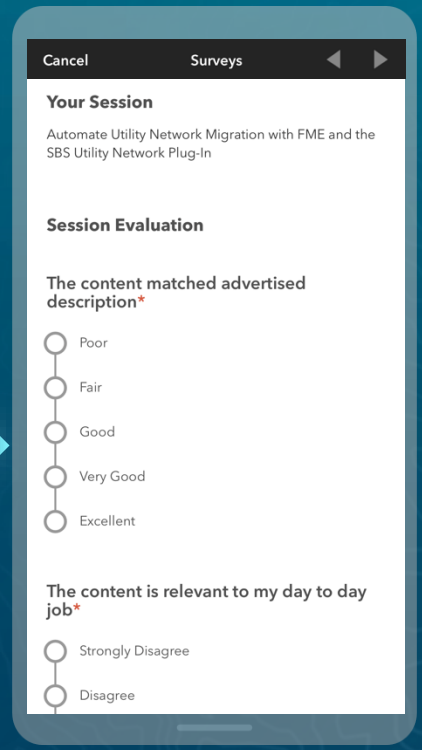
Select the session you attended



Scroll down to “Survey”



Log in to access the survey



Connect With Us On Social

And Join the Conversation Using #EsriDevSummit

- twitter.com/EsriDevs #esridevsummit2024
- twitter.com/EsriDevEvents
- youtube.com/c/EsriDevelopers
- links.esri.com/DevVideos
- github.com/Esri
- github.com/EsriDevEvents
- links.esri.com/EsriDevCommunity