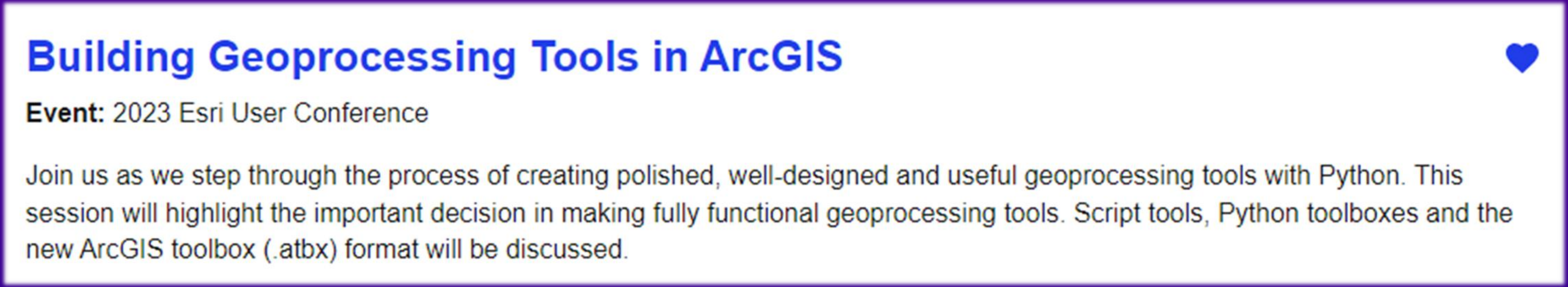
Building Geoprocessing Tools in ArcGIS

Dave Wynne

Sean Lim

2023 ESRI USER CONFERENCE



Building Geoprocessing Tools in ArcGIS

Event: 2023 Esri User Conference

Join us as we step through the process of creating polished, well-designed and useful geoprocessing tools with Python. This session will highlight the important decision in making fully functional geoprocessing tools. Script tools, Python toolboxes and the new ArcGIS toolbox (.atbx) format will be discussed.

Related geoprocessing concepts | Toolbox types | Parameters

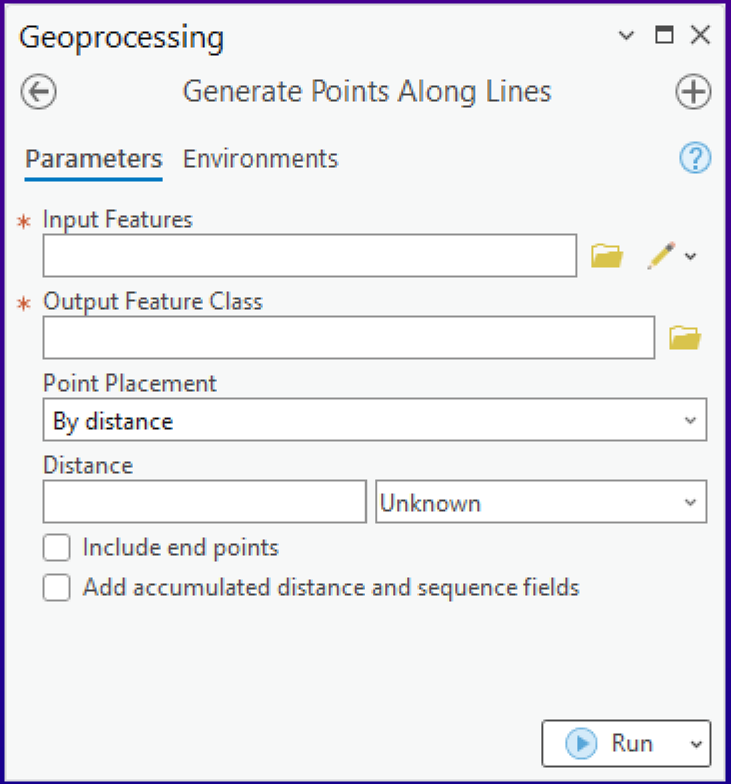
Communication | Validation | Tips | And more …

Why we build tools

- Extend Pro
- Organize functionality

- Your tool becomes part of the geoprocessing framework

- Use your tools in multiple ways
  - Geoprocessing pane
  - Python
  - ModelBuilder
  - Potentially as a geoprocessing service



Geoprocessing

Generate Points Along Lines

Parameters Environments

* Input Features

* Output Feature Class

Point Placement

By distance

Distance

Unknown

□ Include end points

□ Add accumulated distance and sequence fields

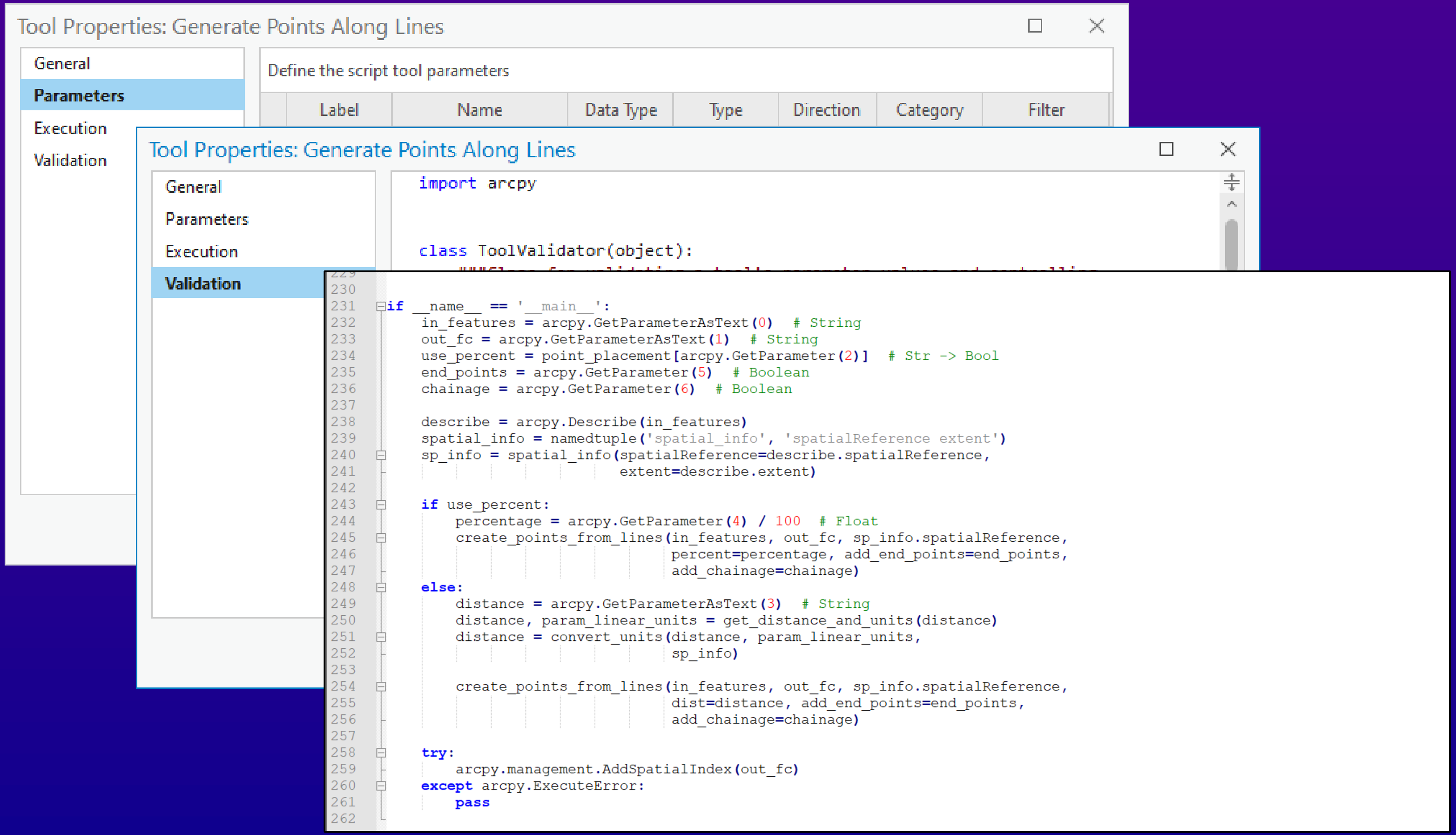
What makes a good geoprocessing tool?

- Performs an essential and elemental operation
- Simple
- Can be combined with other tools for larger processes

- Looks and behaves like other tools
  - For example:
    - A tool always has an output
    - Required parameters precede optional parameters

Tool structure

- Parameters
- Validation code
- Execution code



Tool Properties: Generate Points Along Lines

Define the script tool parameters

Label	Name	Data Type	Type	Direction	Category	Filter

Tool Properties: Generate Points Along Lines

```python
import arcpy


class ToolValidator(object):
```

```python
if __name__ == '__main__':
    in_features = arcpy.GetParameterAsText(0)  # String
    out_fc = arcpy.GetParameterAsText(1)  # String
    use_percent = point_placement[arcpy.GetParameter(2)]  # Str -> Bool
    end_points = arcpy.GetParameter(5)  # Boolean
    chainage = arcpy.GetParameter(6)  # Boolean

    describe = arcpy.Describe(in_features)
    spatial_info = namedtuple('spatial_info', 'spatialReference extent')
    sp_info = spatial_info(spatialReference=describe.spatialReference,
                           extent=describe.extent)

    if use_percent:
        percentage = arcpy.GetParameter(4) / 100  # Float
        create_points_from_lines(in_features, out_fc, sp_info.spatialReference,
                                 percent=percentage, add_end_points=end_points,
                                 add_chainage=chainage)
    else:
        distance = arcpy.GetParameterAsText(3)  # String
        distance, param_linear_units = get_distance_and_units(distance)
        distance = convert_units(distance, param_linear_units,
                                 sp_info)

        create_points_from_lines(in_features, out_fc, sp_info.spatialReference,
                                 dist=distance, add_end_points=end_points,
                                 add_chainage=chainage)

    try:
        arcpy.management.AddSpatialIndex(out_fc)
    except arcpy.ExecuteError:
        pass
```

Toolboxes

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

1. ArcGIS toolbox (.atbx) or legacy toolbox (.tbx)

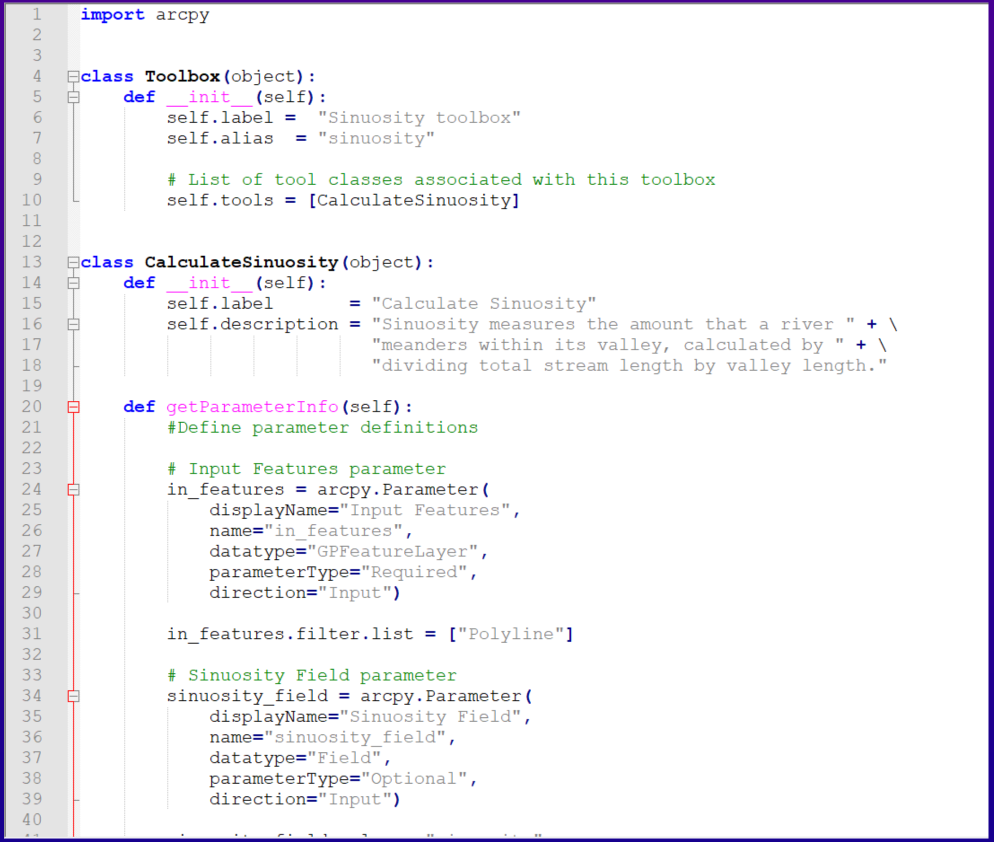
- Parameters defined through the Pro UI
- Validation is Python
- Execution is Python

Toolboxes

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

2. Python toolbox (.pyt)

- Parameters are Python
- Validation is Python
- Execution is Python

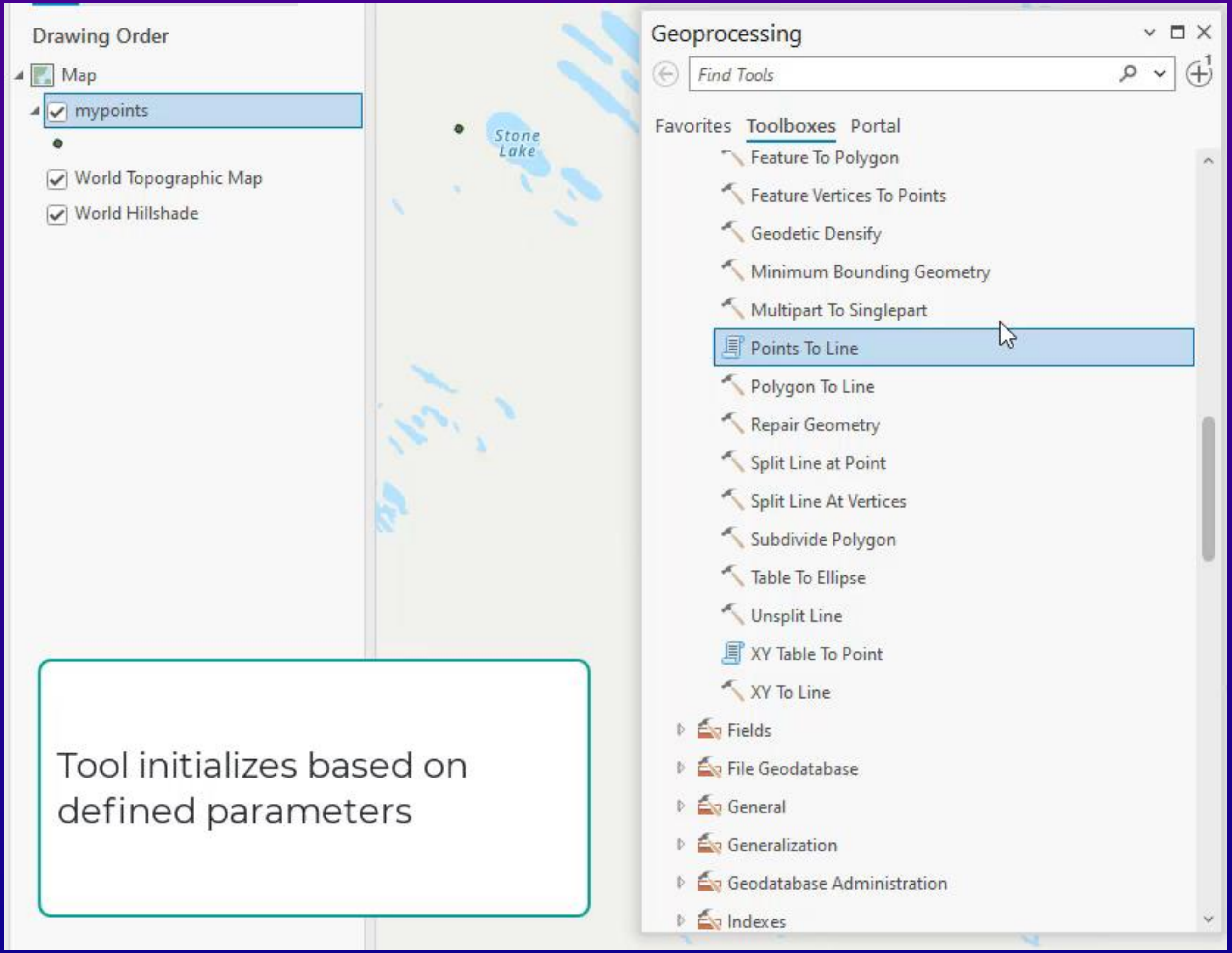


```python
import arcpy


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)

- Introduced at Pro 2.9
- Similar to the traditional toolbox (.tbx) format
- JSON-based with an open specification
- Stores tools, scripts, and models
- Better cross-release compatibility and persistence

How a tool works

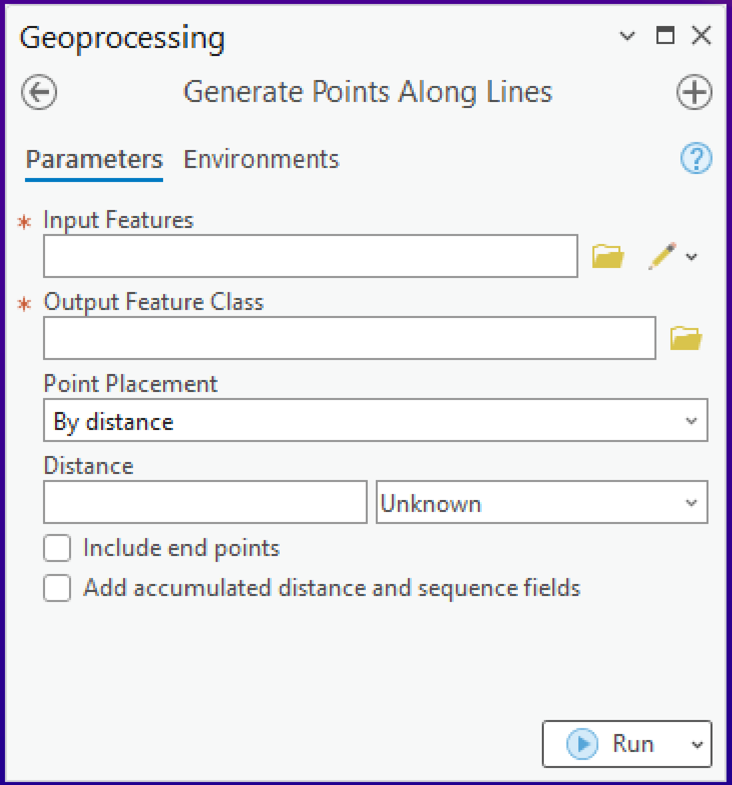


Tool initializes based on defined parameters

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?



Geoprocessing

Generate Points Along Lines

* Input Features

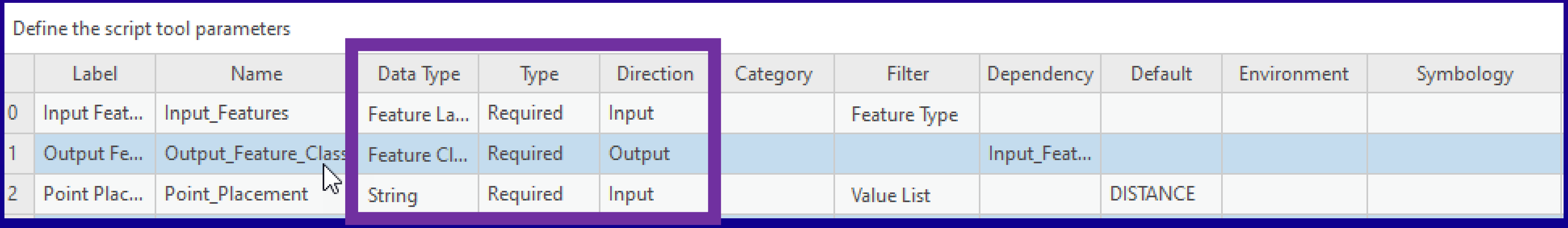
* Output Feature Class

Point Placement: By distance

Distance: Unknown

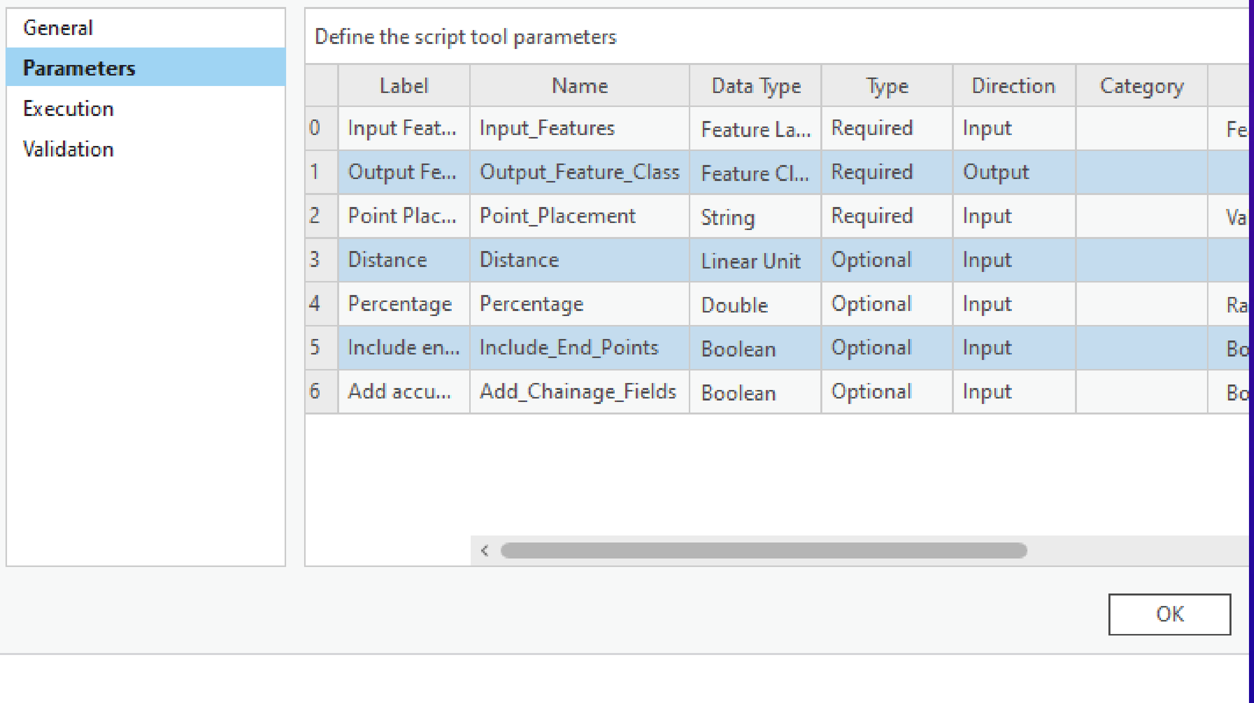
□ Include end points

□ Add accumulated distance and sequence fields



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		



Define the script tool parameters

	Label	Name	Data Type	Type	Direction	Category	
0	Input Feat...	Input_Features	Feature La...	Required	Input		Fe
1	Output Fe...	Output_Feature_Class	Feature Cl...	Required	Output		
2	Point Plac...	Point_Placement	String	Required	Input		Va
3	Distance	Distance	Linear Unit	Optional	Input		
4	Percentage	Percentage	Double	Optional	Input		Ra
5	Include en...	Include_End_Points	Boolean	Optional	Input		Bo
6	Add accu...	Add_Chainage_Fields	Boolean	Optional	Input		Bo

Parameters

Sean Lim

Accessing parameters in the code

- To access parameter values from arcpy, use:
  - GetParameterAsText – values returned as a string
  - GetParameter – values returned as a dynamic type*

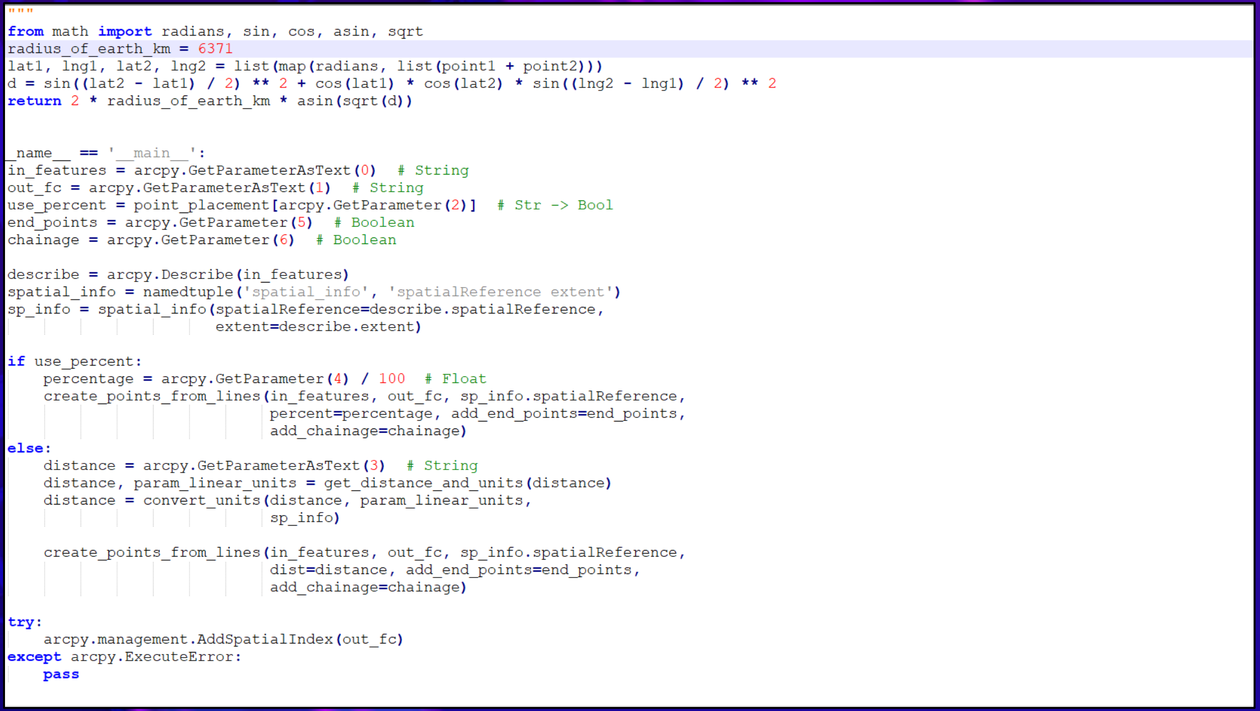
- *GetParameter is best for Boolean and numeric types

- For derived parameters, returns values back to the tool using:
  - SetParameterAsText or SetParameter

Communication within the tool (messages)

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

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



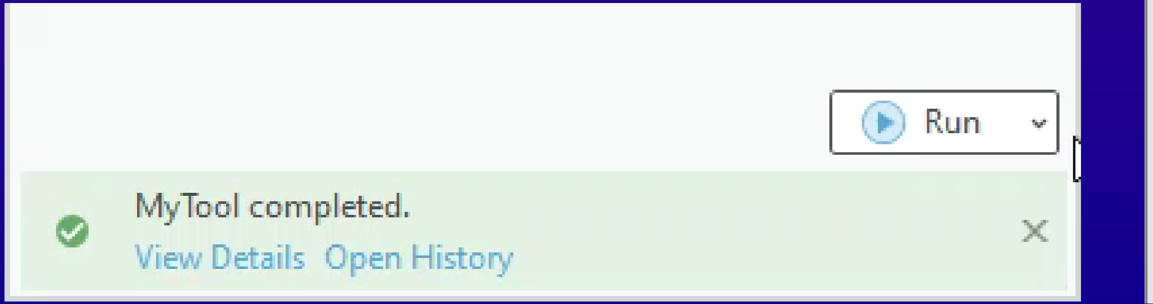
```
"""
from math import radians, sin, cos, asin, sqrt
radius_of_earth_km = 6371
lat1, lng1, lat2, lng2 = list(map(radians, list(point1 + point2)))
d = sin((lat2 - lat1) / 2) ** 2 + cos(lat1) * cos(lat2) * sin((lng2 - lng1) / 2) ** 2
return 2 * radius_of_earth_km * asin(sqrt(d))


_name__ == '__main__':
in_features = arcpy.GetParameterAsText(0)  # String
out_fc = arcpy.GetParameterAsText(1)  # String
use_percent = point_placement[arcpy.GetParameter(2)]  # Str -> Bool
end_points = arcpy.GetParameter(5)  # Boolean
chainage = arcpy.GetParameter(6)  # Boolean

describe = arcpy.Describe(in_features)
spatial_info = namedtuple('spatial_info', 'spatialReference extent')
sp_info = spatial_info(spatialReference=describe.spatialReference,
                       extent=describe.extent)

if use_percent:
    percentage = arcpy.GetParameter(4) / 100  # Float
    create_points_from_lines(in_features, out_fc, sp_info.spatialReference,
                             percent=percentage, add_end_points=end_points,
                             add_chainage=chainage)
else:
    distance = arcpy.GetParameterAsText(3)  # String
    distance, param_linear_units = get_distance_and_units(distance)
    distance = convert_units(distance, param_linear_units,
                             sp_info)

    create_points_from_lines(in_features, out_fc, sp_info.spatialReference,
                             dist=distance, add_end_points=end_points,
                             add_chainage=chainage)

try:
    arcpy.management.AddSpatialIndex(out_fc)
except arcpy.ExecuteError:
    pass
```

Tool source code

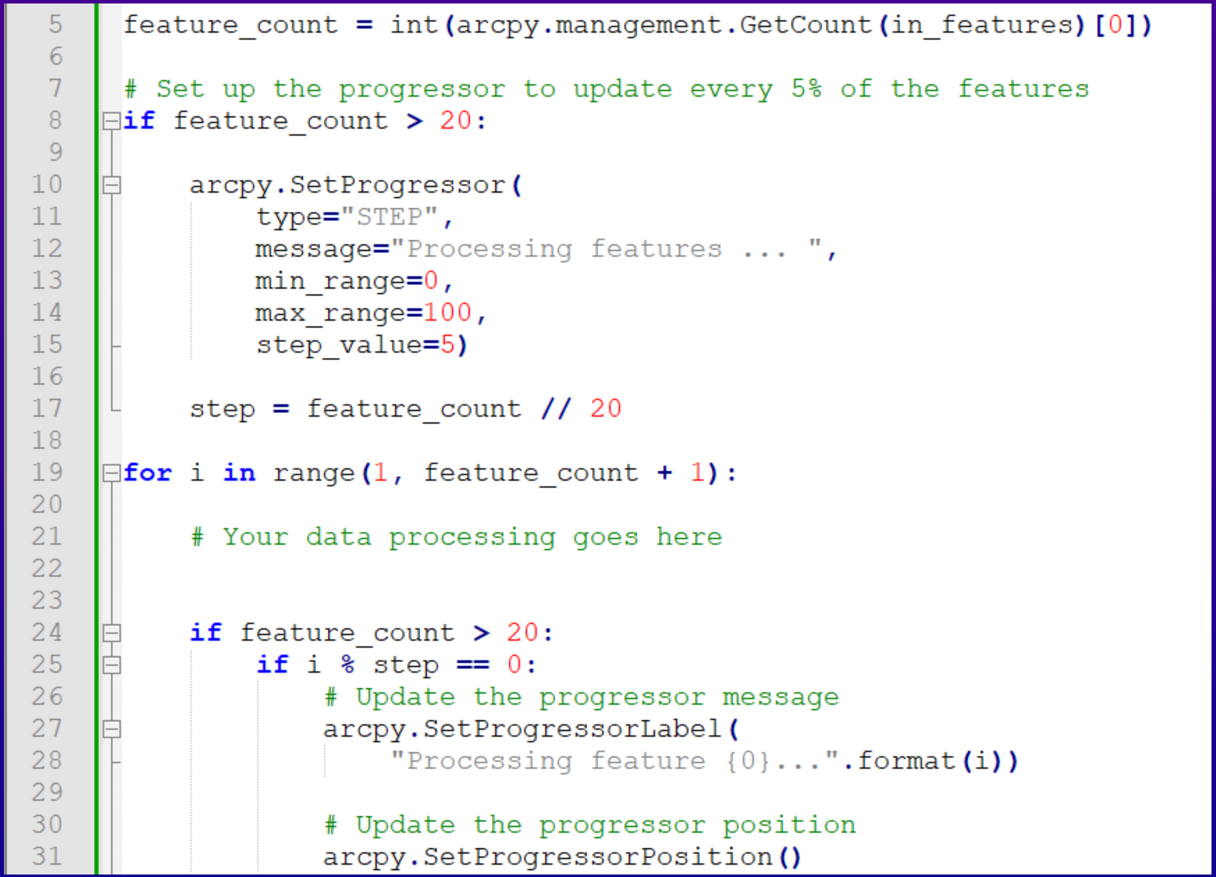
Sean Lim

Communication within the tool (progressor)

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



MyTool completed.



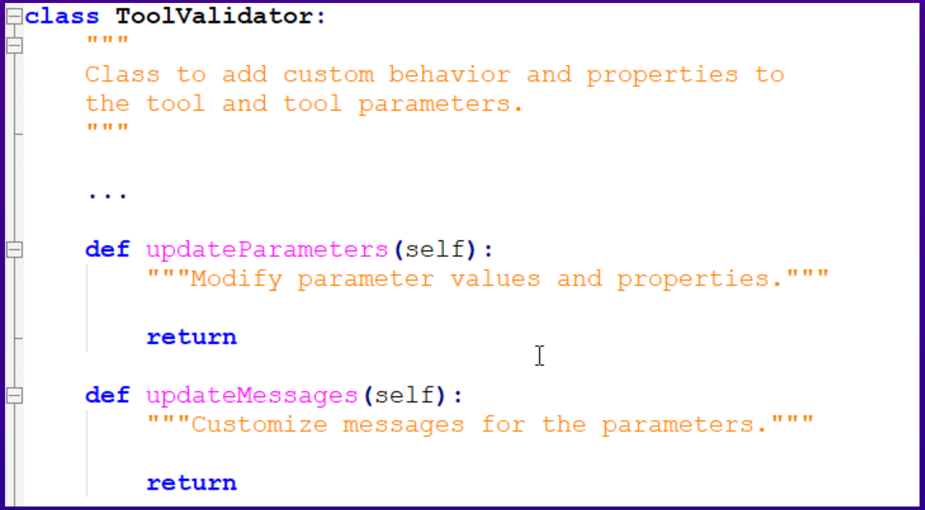
```
feature_count = int(arcpy.management.GetCount(in_features)[0])

# Set up the progressor to update every 5% of the features
if feature_count > 20:

    arcpy.SetProgressor(
        type="STEP",
        message="Processing features ... ",
        min_range=0,
        max_range=100,
        step_value=5)

    step = feature_count // 20

for i in range(1, feature_count + 1):

    # Your data processing goes here


    if feature_count > 20:
        if i % step == 0:
            # Update the progressor message
            arcpy.SetProgressorLabel(
                "Processing feature {0}...".format(i))

            # Update the progressor position
            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)

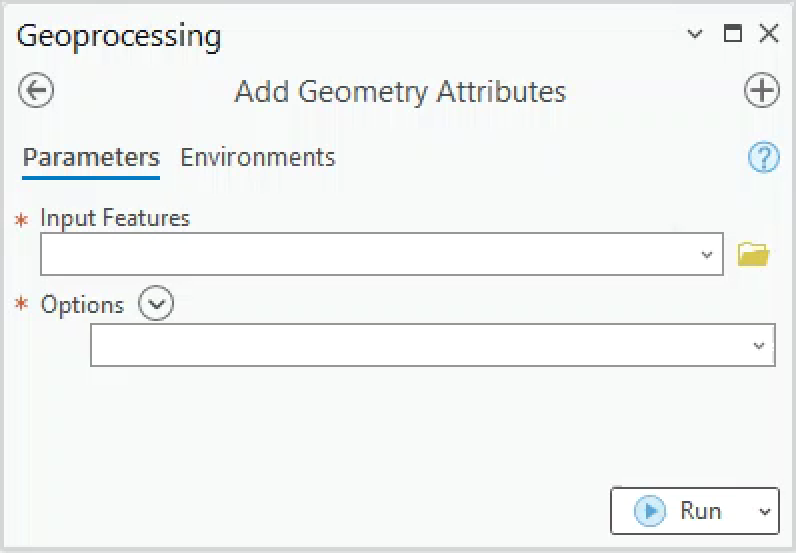
- Validation runs every time a parameter is modified

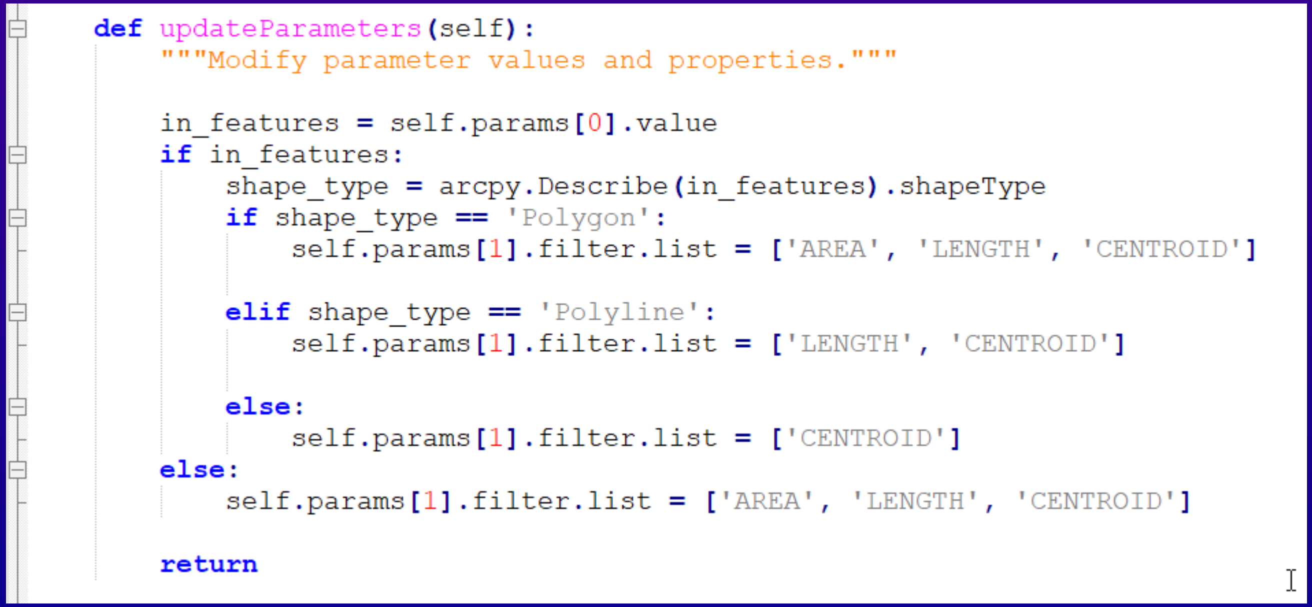


```
class ToolValidator:
    """
    Class to add custom behavior and properties to
    the tool and tool parameters.
    """


    ...

    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 certain parameter characteristics
  - Values
  - Filters
  - Enabled
  - Etc.

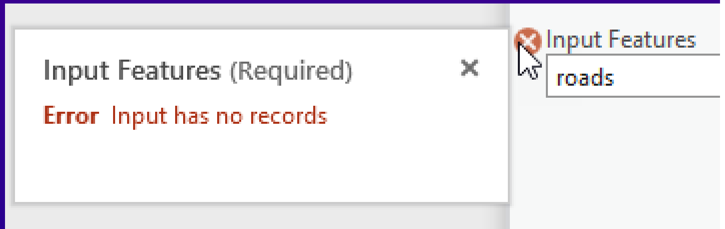




```python
    def updateParameters(self):
        """Modify parameter values and properties."""

        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
```

Validation – updateMessages

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

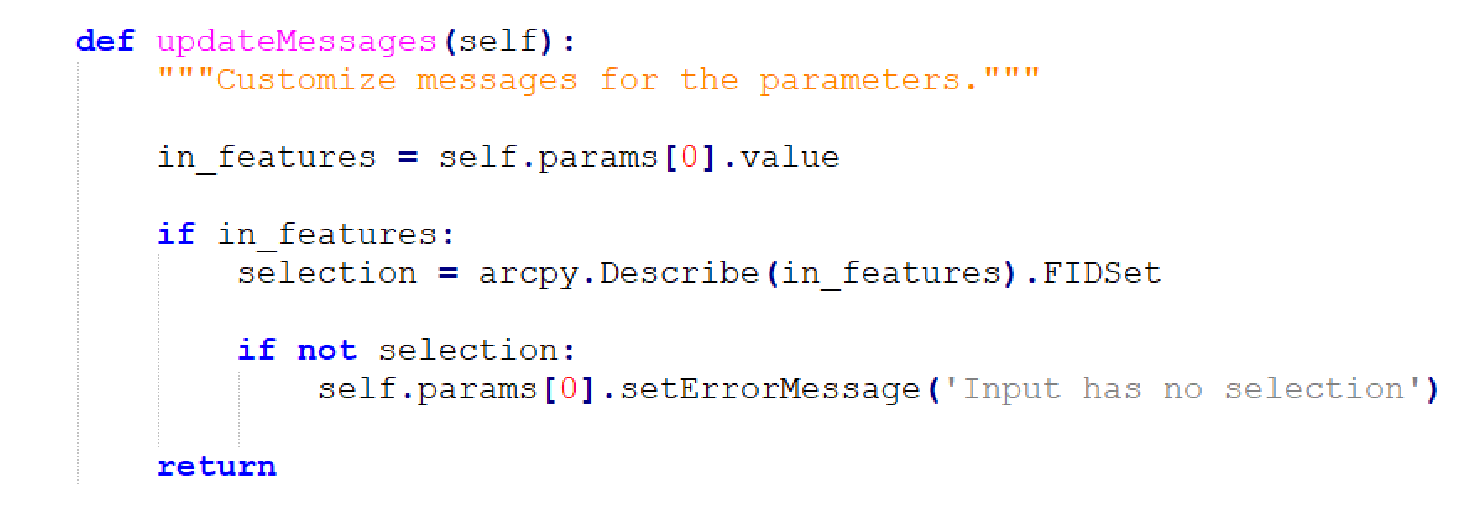


Input Features (Required)

Error Input has no records

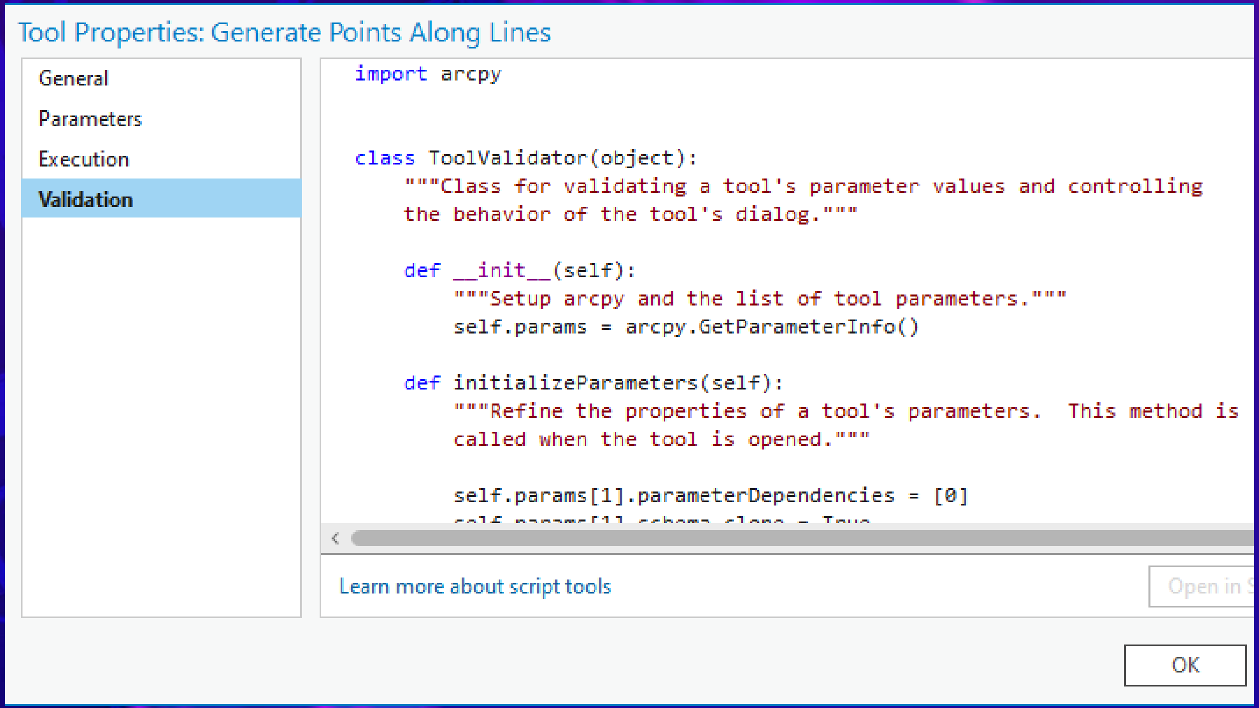
Input Features

roads



```python
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


Tool Properties: Generate Points Along Lines
General
Parameters
Execution
Validation
Learn more about script tools
Open in S
OK

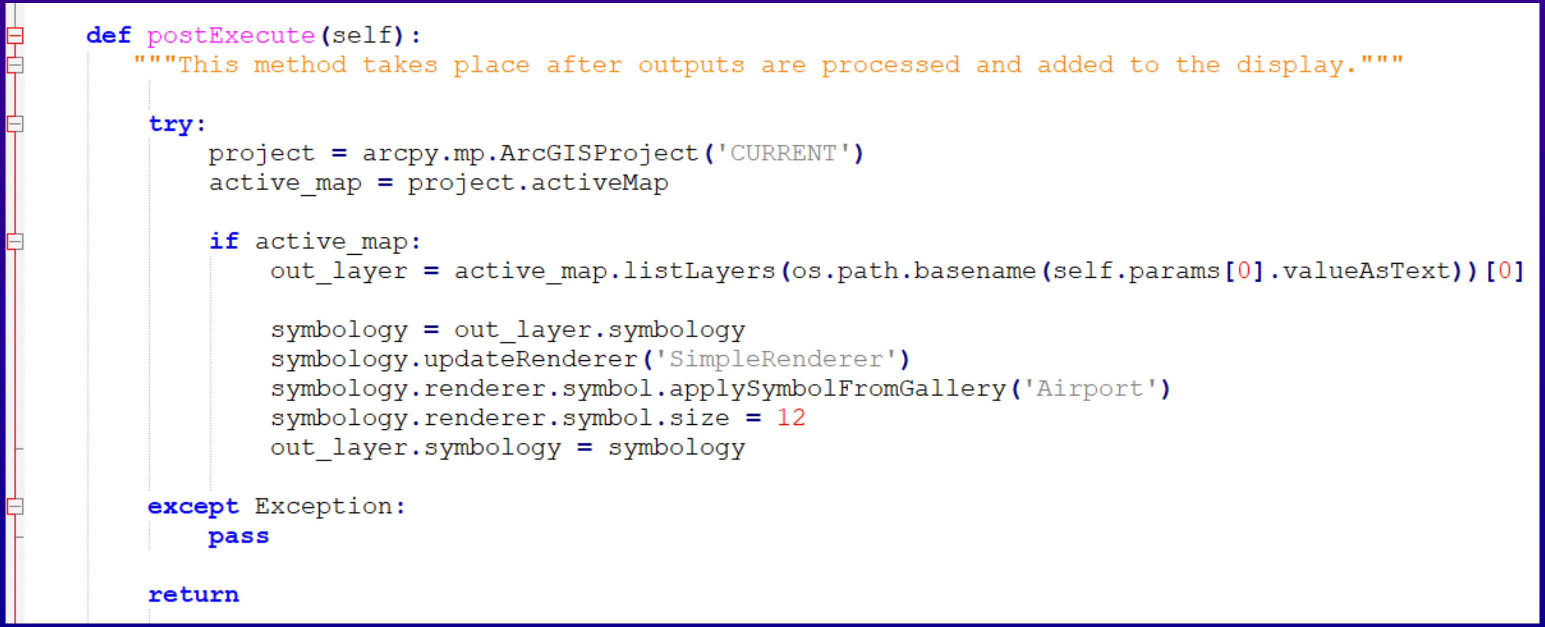
```
import arcpy


class ToolValidator(object):
    """Class for validating a tool's parameter values and controlling
    the behavior of the tool's dialog."""

    def __init__(self):
        """Setup arcpy and the list of tool parameters."""
        self.params = arcpy.GetParameterInfo()

    def initializeParameters(self):
        """Refine the properties of a tool's parameters.  This method is
        called when the tool is opened."""

        self.params[1].parameterDependencies = [0]
        self.params[1].schema.clone = True
```

Validation (and metadata)

Sean Lim

Symbology

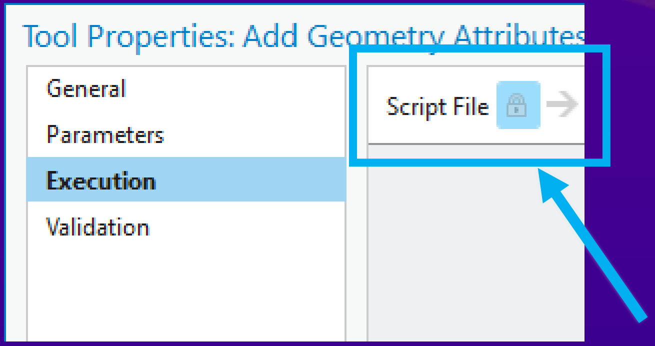
- Use a layer file to set a parameter’s symbology property
- Or, use the postExecute validation method *(new at 3.0)*
  - Is called when a tool completes
  - Use the arcpy.mp module

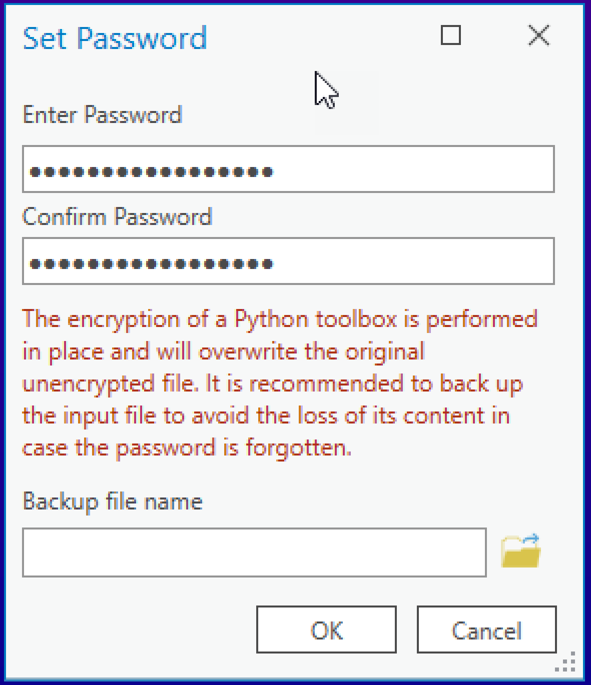


```
    def postExecute(self):
        """This method takes place after outputs are processed and added to the display."""

        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
```

Embedding and encryption

- You can embed code in .atbx and .tbx toolboxes
  - One less file to manage
- Once embedded the code can be encrypted
- Python toolboxes also support encryption
  - The entire .pyt file is encrypted

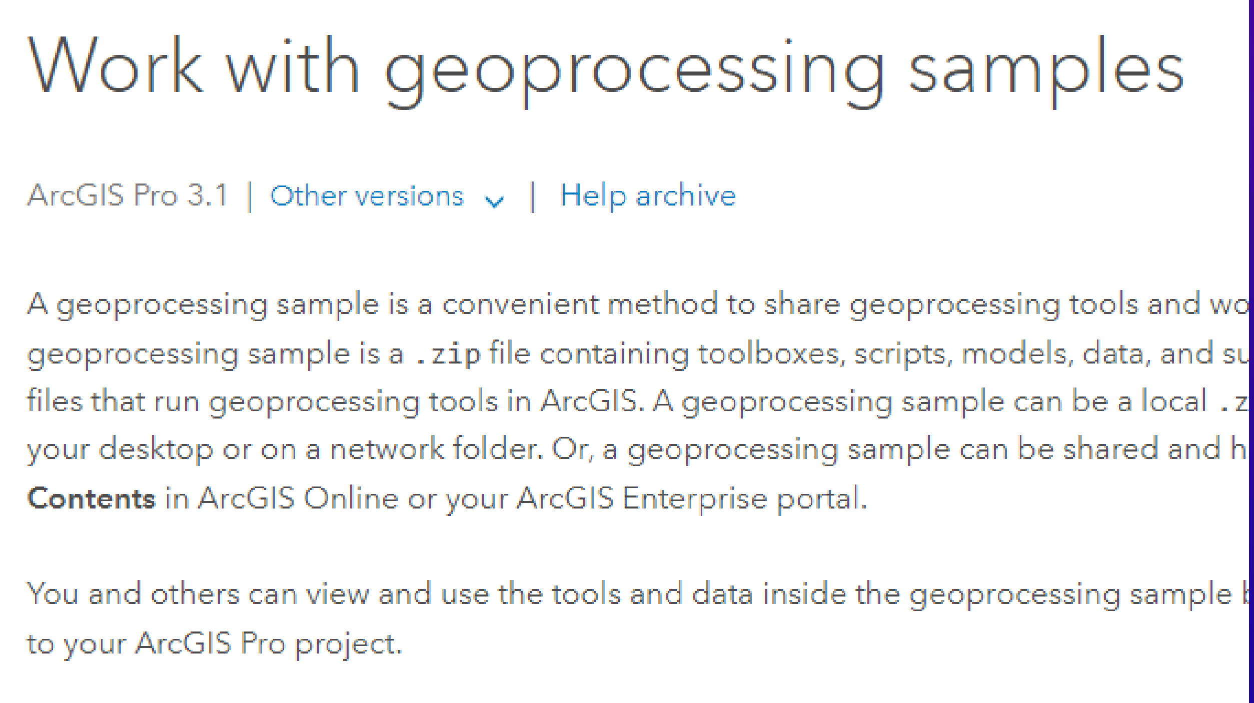




The encryption of a Python toolbox is performed in place and will overwrite the original unencrypted file. It is recommended to back up the input file to avoid the loss of its content in case the password is forgotten.

Geoprocessing samples

Sean Lim



Work with geoprocessing samples

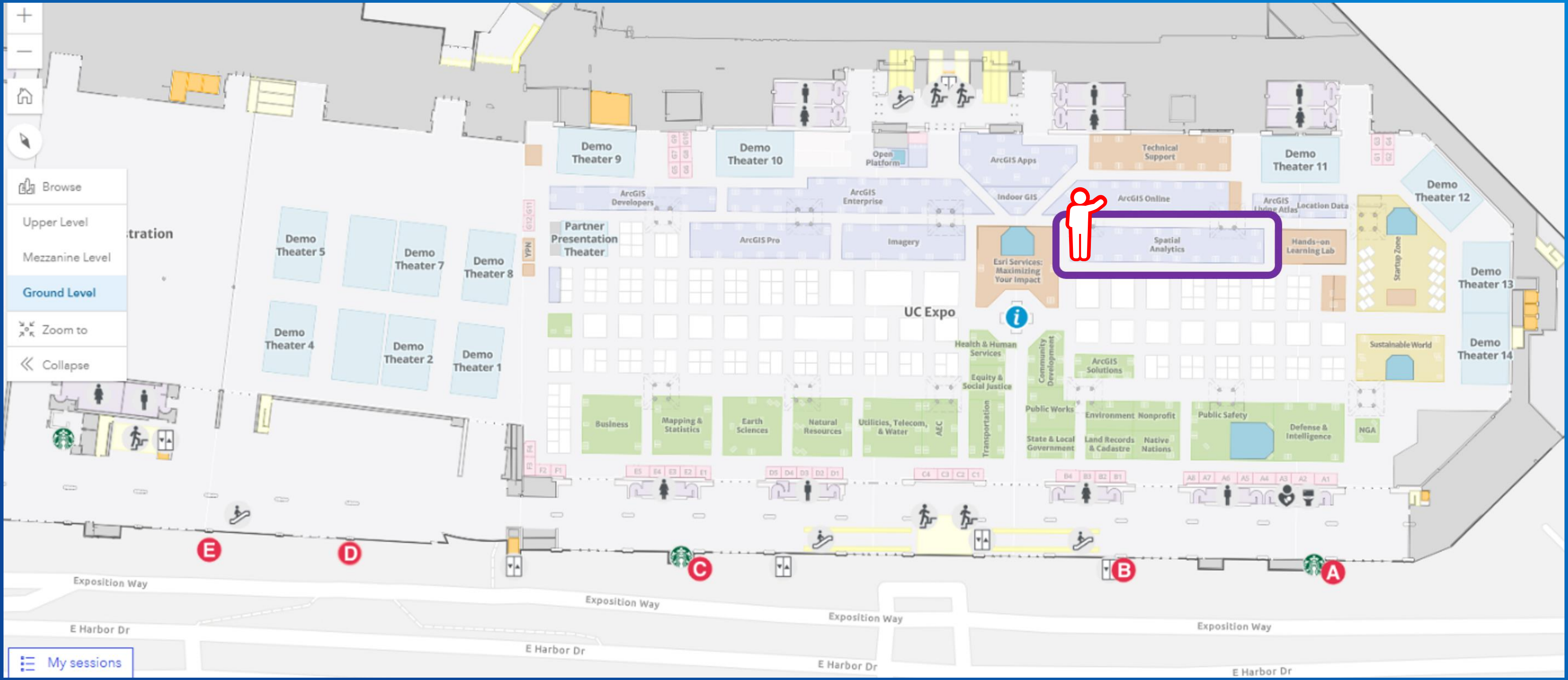
ArcGIS Pro 3.1 | Other versions ⌄ | Help archive

A geoprocessing sample is a convenient method to share geoprocessing tools and wo
geoprocessing sample is a `.zip` file containing toolboxes, scripts, models, data, and su
files that run geoprocessing tools in ArcGIS. A geoprocessing sample can be a local `.z`
your desktop or on a network folder. Or, a geoprocessing sample can be shared and h
Contents in ArcGIS Online or your ArcGIS Enterprise portal.

You and others can view and use the tools and data inside the geoprocessing sample b
to your ArcGIS Pro project.

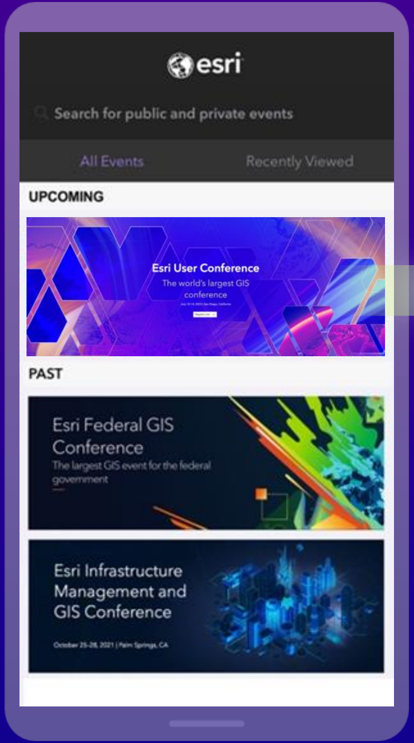
https://pro.arcgis.com/en/pro-app/latest/help/analysis/geoprocessing/share-analysis/geoprocessing-samples.htm

Come say hi


Browse
Upper Level
Mezzanine Level
Ground Level
Zoom to
Collapse
My sessions
tration
Demo Theater 9
Demo Theater 10
Open Platform
ArcGIS Apps
Technical Support
Demo Theater 11
Demo Theater 12
ArcGIS Developers
ArcGIS Enterprise
Indoor GIS
ArcGIS Online
ArcGIS Living Atlas
Location Data
Partner Presentation Theater
ArcGIS Pro
Imagery
Esri Services: Maximizing Your Impact
Spatial Analytics
Hands-on Learning Lab
Startup Zone
Demo Theater 13
Demo Theater 14
Demo Theater 5
Demo Theater 7
Demo Theater 8
Demo Theater 4
Demo Theater 2
Demo Theater 1
UC Expo
Health & Human Services
Community Development
Equity & Social Justice
ArcGIS Solutions
Sustainable World
Business
Mapping & Statistics
Earth Sciences
Natural Resources
Utilities, Telecom, & Water
AEC
Transportation
Public Works
State & Local Government
Environment
Nonprofit
Land Records & Cadastre
Native Nations
Public Safety
Defense & Intelligence
NGA
E
D
C
B
A
Exposition Way
E Harbor Dr

Please Share Your Feedback in the App

Download the Esri Events app and find your event



esri

Search for public and private events

All Events | Recently Viewed

UPCOMING

Esri User Conference

The world's largest GIS conference

PAST

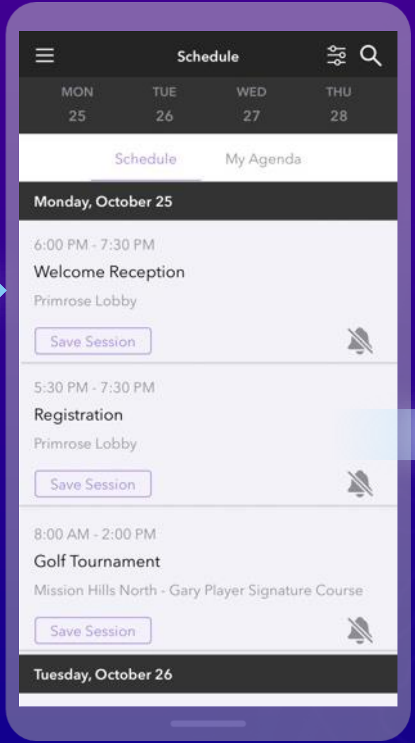
Esri Federal GIS Conference

The largest GIS event for the federal government

Esri Infrastructure Management and GIS Conference

October 25-28, 2021 | Palm Springs, CA

Select the session you attended



Schedule

MON	TUE	WED	THU
25	26	27	28

Schedule | My Agenda

Monday, October 25

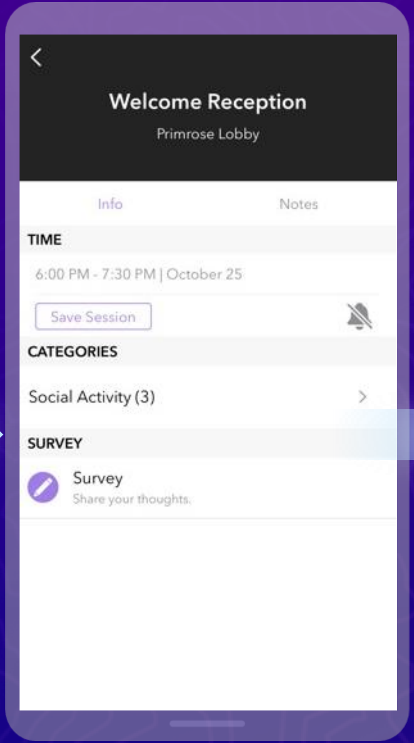
6:00 PM - 7:30 PM
Welcome Reception
Primrose Lobby
Save Session

5:30 PM - 7:30 PM
Registration
Primrose Lobby
Save Session

8:00 AM - 2:00 PM
Golf Tournament
Mission Hills North - Gary Player Signature Course
Save Session

Tuesday, October 26

Scroll down to “Survey”



Welcome Reception

Primrose Lobby

Info | Notes

TIME

6:00 PM - 7:30 PM | October 25

Save Session

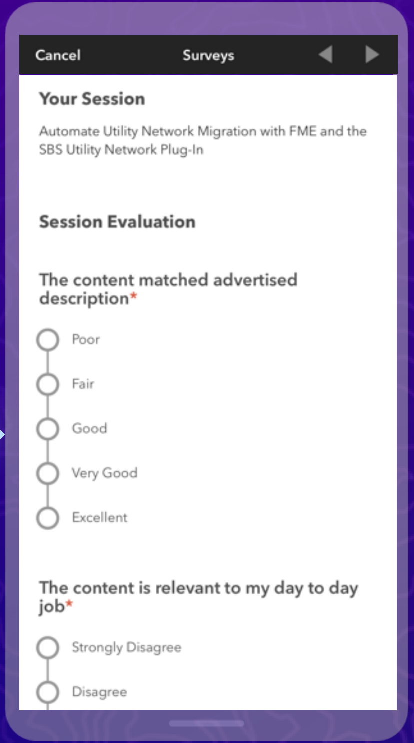
CATEGORIES

Social Activity (3)

SURVEY

Survey
Share your thoughts.

Log in to access the survey



Cancel | Surveys

Your Session

Automate Utility Network Migration with FME and the SBS Utility Network Plug-In

Session Evaluation

The content matched advertised description*

○ Poor
○ Fair
○ Good
○ Very Good
○ Excellent

The content is relevant to my day to day job*

○ Strongly Disagree
○ Disagree

esri®

THE
SCIENCE
OF
WHERE®