



Esri International User Conference | San Diego, CA
Technical Workshops |

Building tools with Python

Dale Honeycutt

Session description – Building Tools with Python

A geoprocessing tool does three pieces of work: it defines its parameters, validates its parameters, and executes some code that performs the actual work. This session will focus on the first two pieces—parameters and validation. At the end of this session, you will know how to define parameters using data types, direction, filters, dependencies, and multivalues. You will also gain understanding of how a tool validates its parameters and describes its output for use in ModelBuilder. Armed with this knowledge, you'll be able to design and build a tool that is robust and useful regardless of what programming language you use for implementation. All concepts will be demonstrated by implementing Python script tools.

Session Evaluations

- www.esri.com/sessionevals





Macros versus Tools

- **Geoprocessing = Computing with geographic data**
 - quickly and easily turn your ideas into repeatable workflows (software)
- **Two basic software types: macro and tool**
- **A macro is tied to a specific set of data**
 - A layer with a particular name (“Streets”), geometry type (lines), fields (“CFCC”, “Meters”)
 - In order to work on another set of data, either the macro code or data must be changed
- **A tool parameterizes data**
 - It is not hard-coded to a particular set of data
 - Handles incorrect data gracefully

Macros and tools in ArcGIS

- You can create macros with:
 - ModelBuilder
 - Python Window
- You can create tools with:
 - ModelBuilder
 - Python Scripts
 - ArcObjects
- Tools that you create are called *custom tools*
 - ...and are found in custom toolboxes that you create

Tool types and categories

Tool type	Description
	Built in tool. These tools are built using ArcObjects and a compiled programming language like .NET.
	Model tool. These tools are created using ModelBuilder.
	Script tool. These tools are created using the Script tool wizard and run a script file on disk, such as a Python file (.py), AML file (.aml), or executable (.exe or .bat).
	Specialized tool. These tools are rare—they are built by system developers and have their own unique user interface for using the tool. The ArcGIS Data Interoperability extension contains specialized tools.

Tool category	Description
System tools	System tools are those tools built and delivered by ESRI. They are installed by ArcGIS or any of its extension products. Almost all system tools are built-in tools, but you will also find system tools that are script or model tools. For example, the Spatial Statistics tools are all script tools, but since they are built and delivered by ESRI, they are considered system tools.
Custom tools	Custom tools are built by you. These are most often script or model tools, but they can be built-in tools as well. There are an infinite number of custom tools. You can download custom tools that other users have built by visiting the Model and Script tool gallery found on the Geoprocessing Resource Center. You can access the Geoprocessing Resource Center at http://resources.esri.com/ .

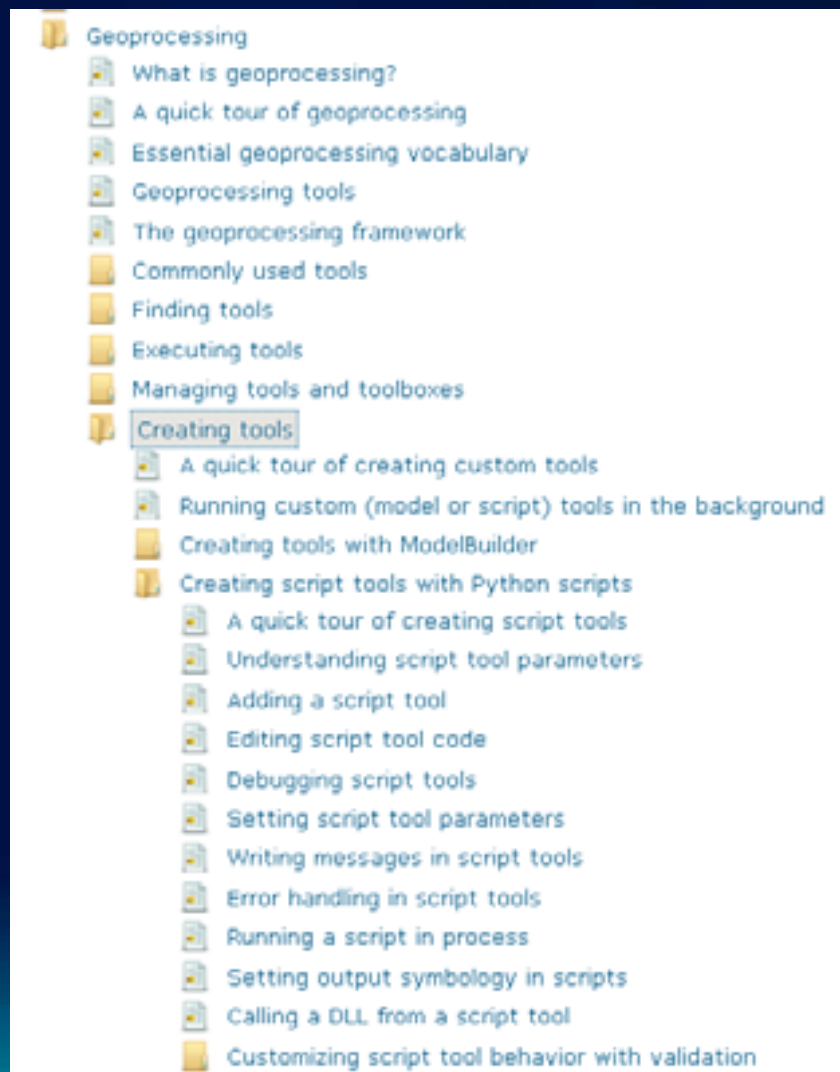
Getting help

Help system topics

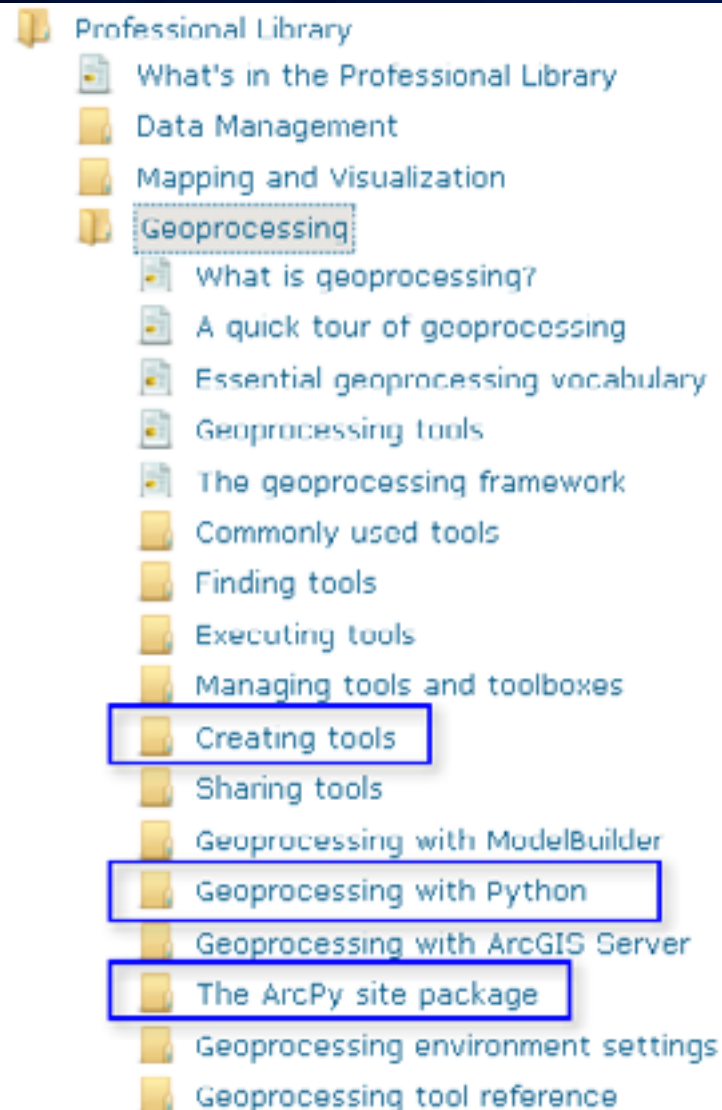


Everything shown today is discussed in the doc

- This illustration is the 10.0 doc
- Applies to 9.3 as well
- See 9.3 doc for use of arcgisscripting versus arcpy



Documentation



- **10.0 doc again – these books will help you when building script tools**

Demo: Script tool basics

Adding a script tool



Demo review – Table to html

- **GetParameterAsText(index)**
 - How parameters are received by the script
- **Add Script Tool Wizard**
 - Properties can be updated later using Properties dialog
- **Parameters**
 - Label
 - Data Type
 - Direction (input, output, derived)
 - Required vs. Optional
 - Filters

Demo – Table to html with field choices

- Field data type – setting **Obtained From**
- **MultiValues** are passed to script as a semi-colon delimited string
 - “aaa;bbb;ccc;ddd”
- Transform to python list with:
 - `mylist = string.split(“;”)`
 - `Fields = arcpy.GetParameterAsText(1).split(“;”)`

Parameter filters

Value List	A list of string or numeric values. Used with String, Long, Double, and Boolean parameter data types.
Range	A minimum and maximum value. Used with Long and Double data types.
Feature Class	A list of allowable feature class types: Point, Multipoint, Polyline, Polygon, MultiPatch, Sphere, Annotation, Dimension. More than one value can be supplied to the filter.
File	A list of file suffixes. Example: "txt; e00; ditamap".
Field	A list of allowable field types: Short, Long, Single, Double, Text, Date, OID, Geometry, Blob, Raster, GUID, GlobalID, XML. More than one value can be supplied to the filter.
Workspace	A list of allowable workspace types: File System, Local Database, Remote Database. More than one value can be supplied.

Demo: Keywords and Booleans



Demo review

- **Keywords:**
 - A **Value List** filter on a string parameter
 - Keywords should be UPPER case, no spaces
- **Booleans:**
 - always have two keywords
 - The True keyword is listed first, False keyword second
 - In scripting, you can either pass
 - The keywords
 - Python True or False
 - Always received as string
- **Be sure to supply a default value**

Data types



Data types

- **Core concept of GP**
- **Every data type has:**
 - A UI control
 - Built-in validation logic
- **Two basic types**
 - Datasets
 - Scalar (numbers, strings, simple structures)
- **Datasets have corresponding data elements**
 - Properties accessed using Describe
 - Lightweight descriptions of the dataset
 - Fields, extent, spatial reference, etc.

Common data types

- **Layers (i.e. Feature Layer, Raster Layer, etc)**
 - Allows user to pick a layer from the TOC, a dataset on disk, or a .lyr file on disk
 - Cursors will only return selected features in the layer
- **Table View**
 - Anything represented as a table (Feature Layer, Joined tables, tables on disk, etc)
 - Cursors will only return selected rows (or features) in the table
- **Long, Double, String, etc.**
- **Workspace, folder, file, text file**

Data types continued

- All data types have a string representation
 - In ArcObjects, the GPDataType object has a GetAsText() and SetAsText() method
- Some data types have a horrid string representation
 - Spatial Reference, Field Map
 - Use arcpy objects to manipulate these
- What data type to use?
 - Find a tool similar to what you want to do
 - Tool reference page lists the data type
 - Experiment in ModelBuilder
 - See [Data types for geoprocessing tool parameters](#)

Demo: Exploring data types



Data types -- review

- **Use Create Variable in ModelBuilder to explore the UI of a data type**
- **Any data type can be made into a multiple value – doesn't always make sense.**
- **Composite data types ("field or value") can only be created in ArcObjects**
- **Feature Set (and Record Set) for interactive entry of features or table rows**

Derived output



All tools **MUST** have output

- **ModelBuilder** – tool's output is used as input to another tool
- **Two types of output:**
 - **Required** : user specifies the output dataset
 - **Derived:**
 - **No output dataset (a number)**
 - **Modification of input**
 - **Output dataset determined by the tool**

Derived data

- The output parameter is set to **Derived**
- Optionally, **Obtained From** is set
- Examples of derived output:
 - **Get Count** : output = a number
 - **Calculate Field** : output = exact copy of input
 - **Add Field** : output = exact copy of input + a new field
 - **Create Feature Class** : new output as defined by inputs (workspace, name, schema)

Demo: Derived output



Demo review: Scalar values

- All scalar values (numbers, text, etc) are derived
 - example: **Get Count**
- In script tool wizard, set output to **Derived**
- Must use **setParameterAsText()** or **setParameter()** in your executable

Demo review – Output same as input

- Example: **Calculate Field**
- In script tool wizard, set output to **Derived** and **Obtained From** to the input parameter
- **SetParameterAsText()** not needed
 - But good habit to do anyway

Demo review – Modify input schema

- Example: **Add Field**
- In script tool wizard, set output to **Derived** and **Obtained From** to the input parameter
- In ToolValidator (coming up next), you'll modify the output schema to contain the new field
- **SetParameterAsText()** not needed
 - But good habit to do anyway

Demo review – new output

- Example: **Create Feature Class**
- In script tool wizard, set output to **Derived**, **Obtained From** is blank (empty)
- Optionally, set the schema in ToolValidator (coming up next)
- Use **SetParameterAsText()** in your executable

Validation

Programming the ToolValidator class



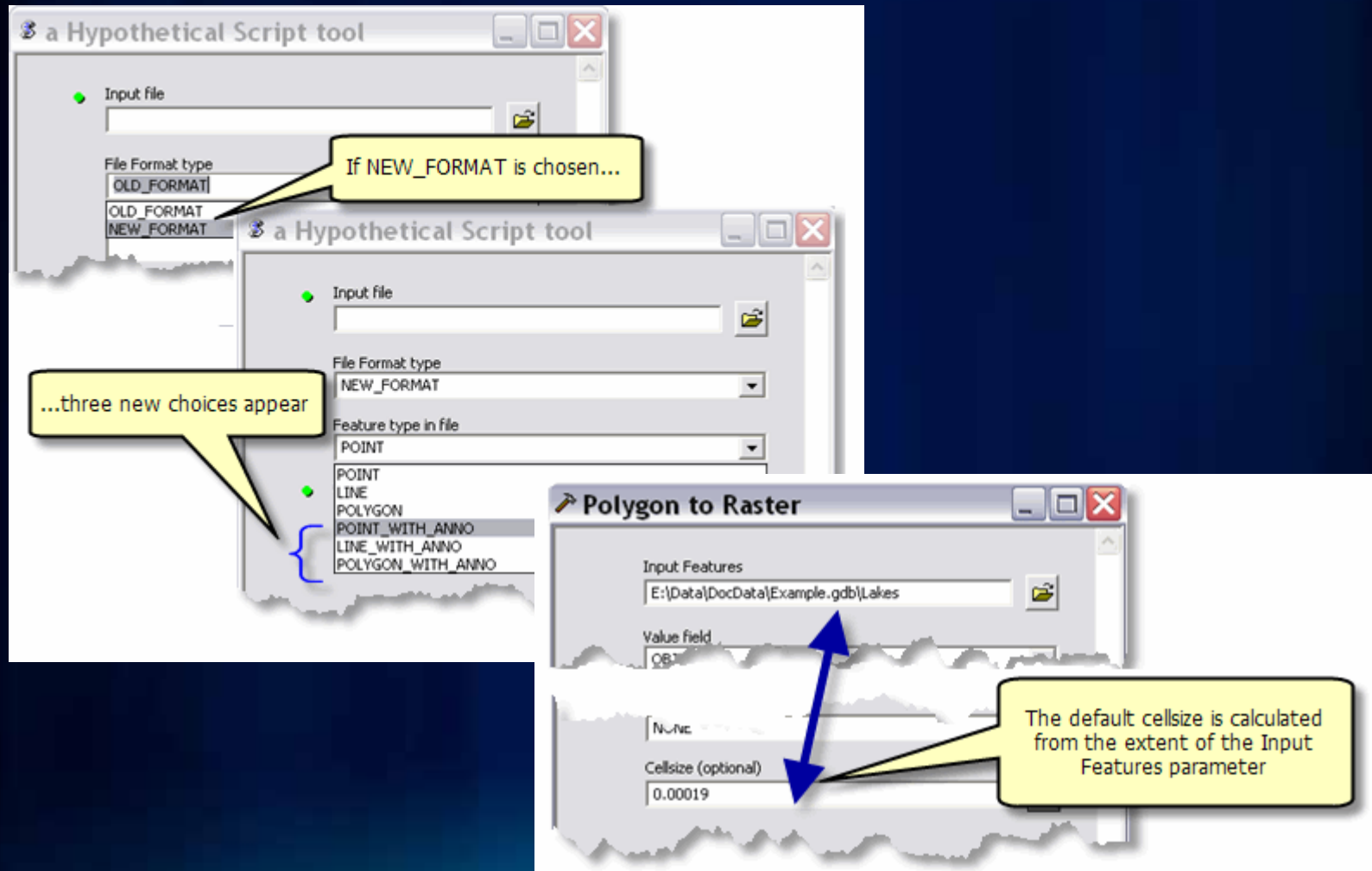
Validation is everything that happens...

- Before the OK button is pushed

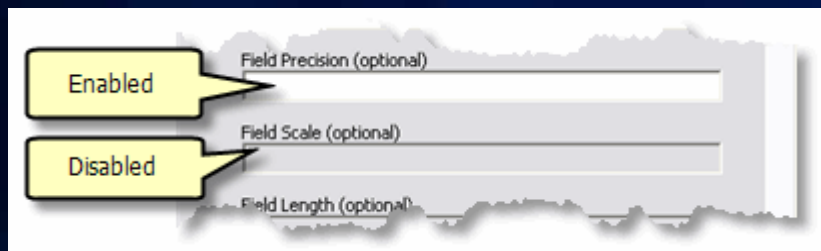
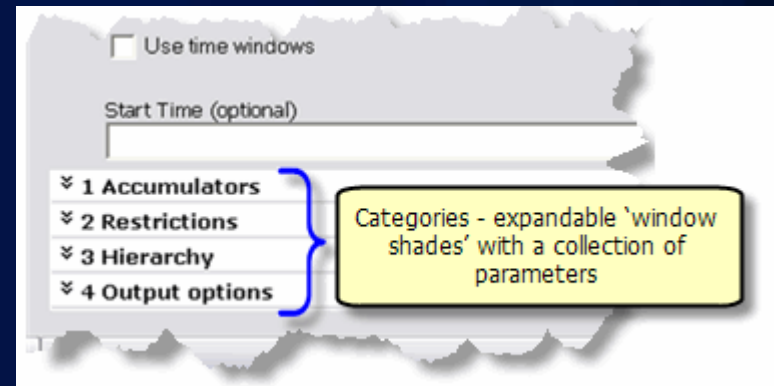
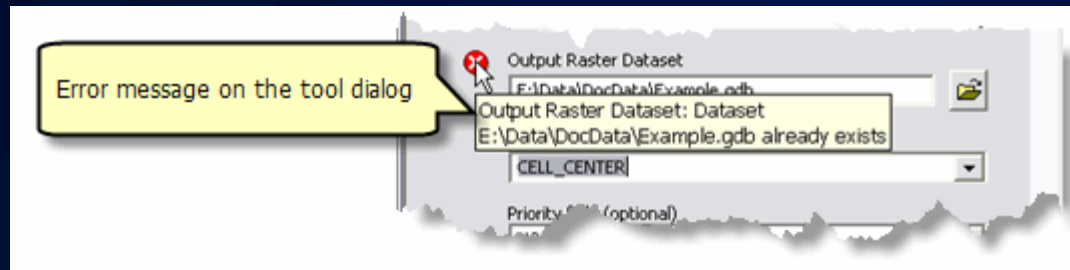
Parameter properties provide basic validation

- Have all the required parameters been supplied?
- Are the values of the appropriate data types?
- Does the input or output exist?
 - If output exists and OverwriteOutputs is false, throw error, throw warning otherwise
- Do values match their filter?

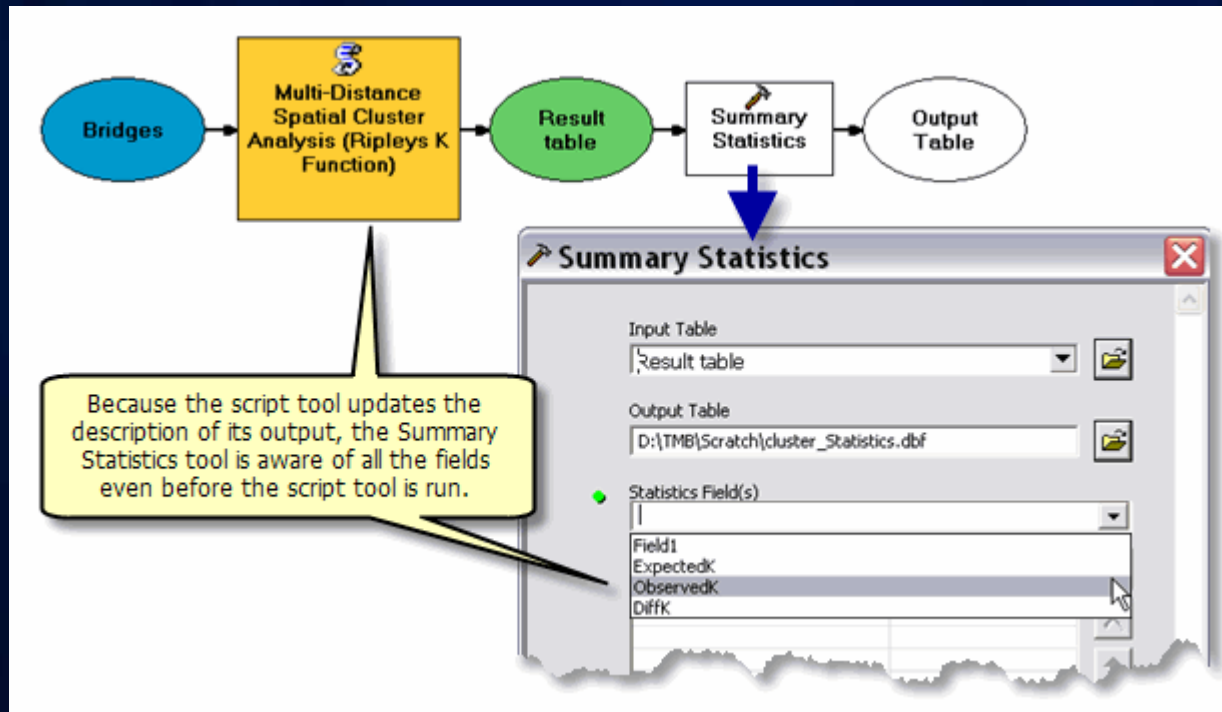
Full validation



Full validation (continued...)



Full validation (continued...)

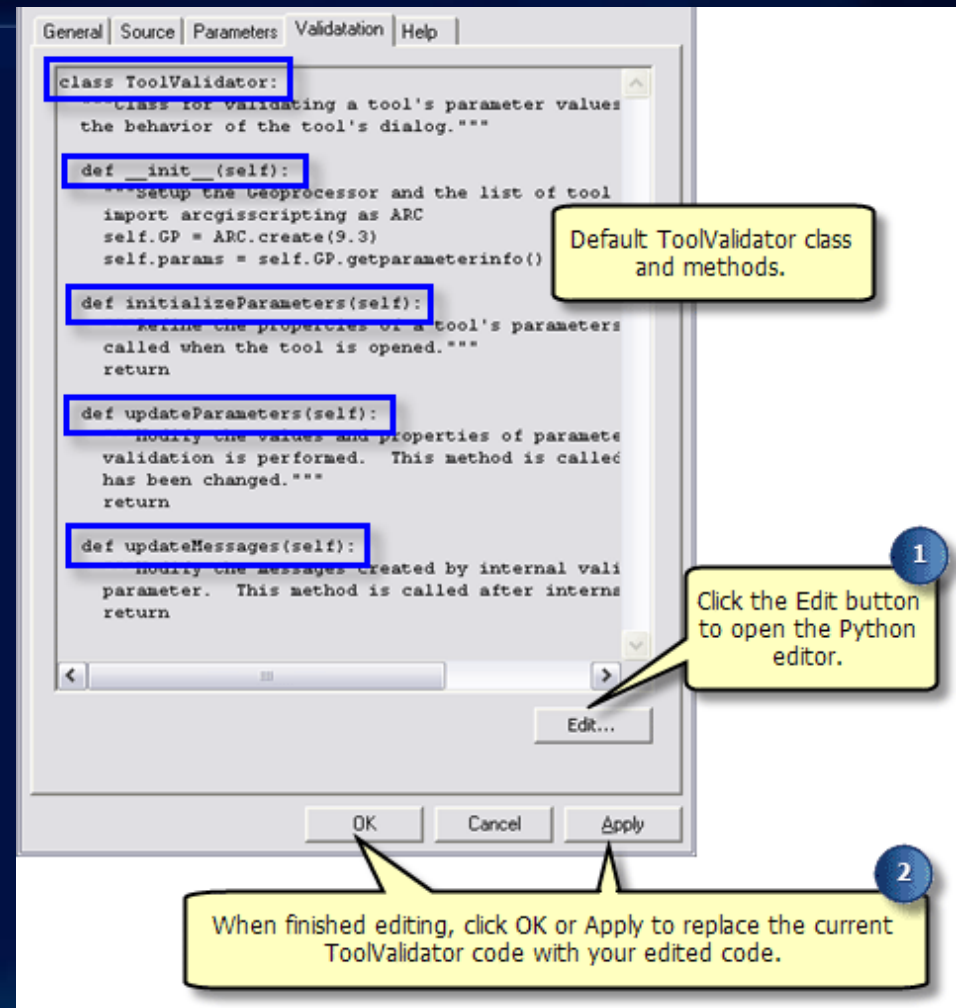


ToolValidator Class

- Introduced at 9.3
- A Python class that you program
- Allows full control of dialog
 - Better UI, validating relationships between parameters, messaging
- Allows you to fully describe outputs for chaining in ModelBuilder
 - Through the use of a **schema object**
 - This is where you define the schema (fields, etc) of derived outputs

Tool Validator Class

- **initializeParameters()** – whenever a tool's signature is requested
- **updateParameters()** – called whenever a parameter value is changed
- **updateMessages()** – called after **updateParameters()**



Demo: controlling the UI



Demo review -- Basic ToolValidator

- The basics of editing a ToolValidator class
- Setting up keyword lists, categories
- Dynamic update of keyword lists
 - Keyword list changes based on values in another parameter
 - (The kind of stuff basic validation cannot do)
- See documentation – [Customizing script tool behavior](#)

Internal validation — called after updateParameters()

- 1. If a required parameter is empty, post the "Value is required" message to the tool dialog (green dot)**
- 2. Check that the value the user entered is of the right type.**
- 3. Check filter membership.**
- 4. Check existence of input datasets**
- 5. Generate a default catalog path for output datasets.**
- 6. Update the description of the output data based on a set of rules contained in a special object called a Schema.**
- 7. Check existence of output datasets against the Overwriteoutputs environment setting.**

But wait! There's more!

9. If the parameter is a Field data type, check that the field exists on the associated table.
10. Check that the output dataset isn't the same as the input dataset (unless the output is derived)
11. For parameters containing linear and areal unit data types, set their default values by examining the corresponding values in ArcMap
12. If the output is a coverage, grid, or INFO table, check the 13 character file name limit for these datasets.

What internal validation *doesn't* do

- Update filters based on interaction with other parameters
- Enable/Disable parameters
- Calculate default values
- Perform any tool-specific parameter interaction
- No custom error/warning messages

Parameter states: Altered & HasBeenValidated



parameter.altered

- Once the user specifies a value for a parameter, it is **altered** and remains altered forever
 - ...unless the user empties (blanks out) the parameter
- Once a parameter has been altered by a user, you should never reset its value

parameter.hasBeenValidated

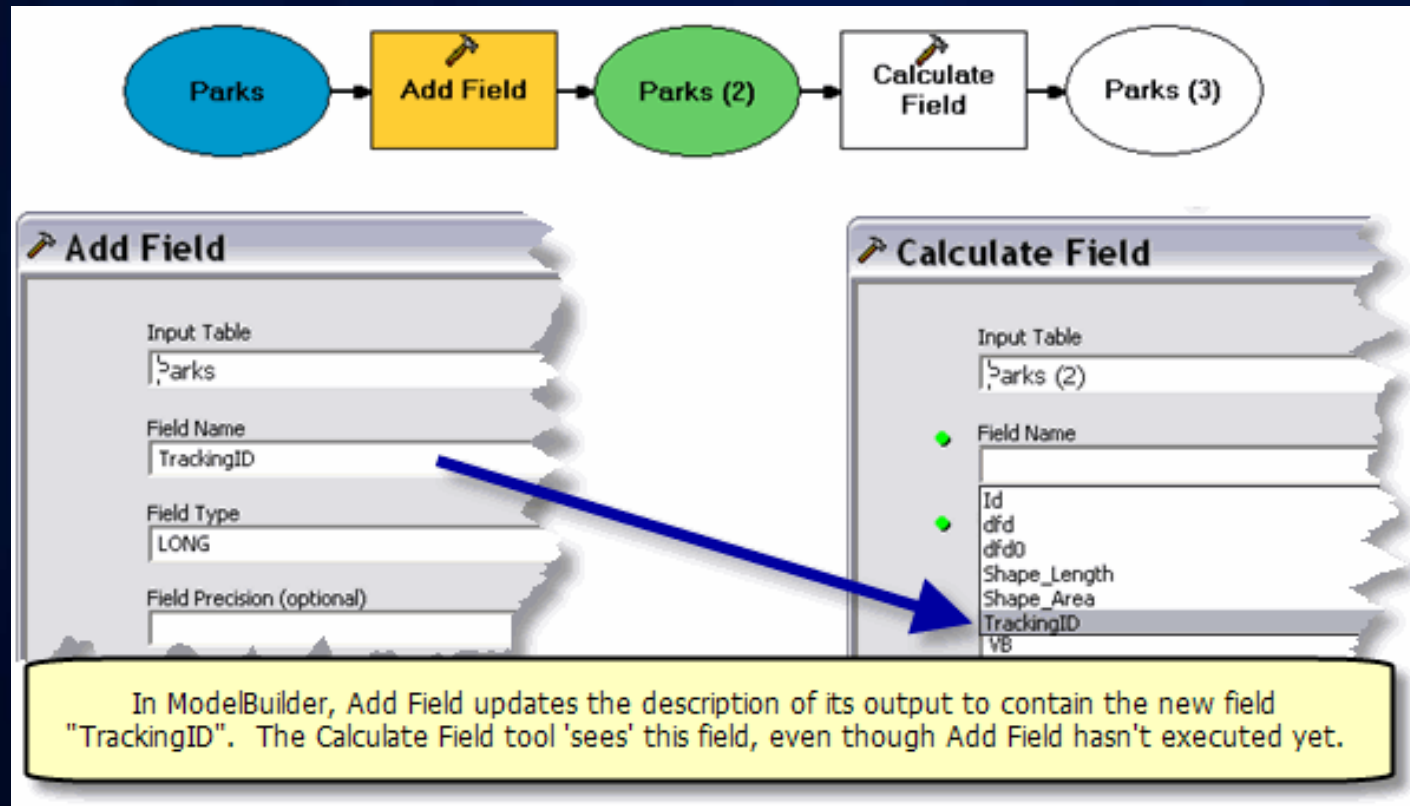
- **hasBeenValidated** is true if the user has changed the value since the last time **updateParameters()** was called.

Describing the output

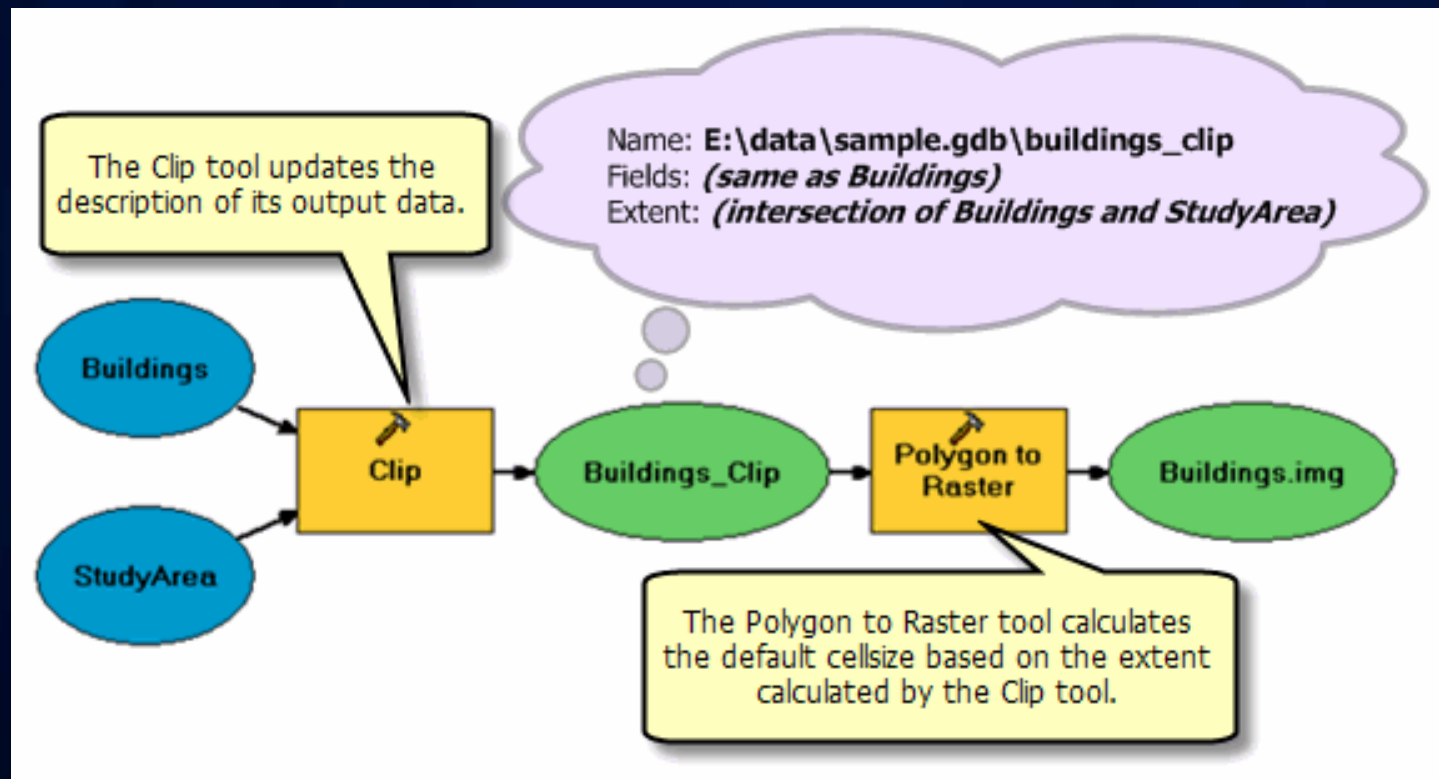
Using the schema object



Describing the output



Describing the output



Demo – Updating the description of the output

- Tool takes a line feature class and produces a new point feature class containing the endpoints and midpoints of each line
- Output
 - Point feature class
 - All the same attributes as the input
 - Additional field named STATUS with these values:
 - 0 = from point
 - 1 = mid point
 - 2 = end point
 - Roughly the same spatial extent

Demo: updating output schema



Demo review

- Output datasets have a schema object that describes the dataset
- You set up rules on how you want to construct the output dataset description
 - We used `fieldsRule = "All"`
- Parameter dependencies declare the initial schema of the output
- You apply rules to the schema
 - Feature/Geometry type
 - Extent
 - AdditionalFields

Schema object methods (rules)

Property name	Value(s)
type	String: "Feature", "Table", "Raster", "Container" (for workspaces and feature datasets). (Read-only property.)
clone	Boolean
featureTypeRule	String: "AsSpecified", "FirstDependency"
featureType	String: "Simple", "Annotation", "Dimension"
geometryTypeRule	String: "Unknown", "FirstDependency", "Min", "Max", "AsSpecified"
geometryType	String: "Point", "Multipoint", "Polyline", "Polygon"
extentRule	String: "AsSpecified", "FirstDependency", "Intersection", "Union", "Environment"
extent	Extent object
fieldsRule	String: "None", "FirstDependency", "FirstDependencyFIDsOnly", "All", "AllNoFIDs", "AllFIDsOnly"
additionalFields	Python list of field objects
cellSizeRule	String: "AsSpecified", "FirstDependency", "Min", "Max", "Environment"
cellsize	double
rasterRule	String: "FirstDependency", "Min", "Max", "Integer", "Float"
rasterFormatRule	String: "Img", "Grid"
additionalChildren	Python list of datasets to add to a workspace schema.

The validation flow of control is as follows:

1. When the tool dialog is first opened, `initializeParameters()` is called. You set up the static rules (rules that don't change based on user input) for describing the output. No output description is created at this time since the user hasn't specified values for any of the parameters (unless you've provided default values).
2. Once the user interacts with the tool dialog in any way, `updateParameters()` is called.
 - `updateParameters()` modifies the schema object to account for properties that can't be determined from the parameter dependencies (such as adding a new field).
3. After returning from `updateParameters()`, the *internal validation* routines are called and the rules found in the schema object are applied to update the description of the output data.
4. `updateMessages()` is then called. No changes to the schema can be made here (output description has already been updated).

Tool characteristics

What makes a good tool?



Functional decomposition

- A tool does one elemental operation well
- Use ModelBuilder to sequence tools into a workflow (compose functionality)

A tool must have output

- In order to work in ModelBuilder, a tool **MUST HAVE OUTPUT**
 - Even if it is just a Boolean pass/fail
 - See, for example, the **Delete** tool

Tool validation

- **A tool validates itself**
 - Checks and updates values, repaints UI, messages
 - Uses data elements (descriptions of data) rather than opening datasets
 - Basic validation (w/o using tool validator)
 - Advanced validation (using tool validator)
- **A tool describes its output**
 - Prior to execution, creates an output data element
 - For ModelBuilder chaining

Parameter naming

- **“Input Features” not “Input feature class or layer”**
 - **“Dataset” means user will browse to disk, not use layer**
 - **All initial caps**
- **Label – what shows up in tool dialog (has spaces)**
- **Name – what shows up in scripting syntax**
 - **label with underscores instead of spaces**
 - **You cannot control the name – it’s always the label minus the spaces**

Parameter ordering

- Parameter ordering:
 1. Required input datasets
 2. Required output datasets
 3. Required modifiers
 4. Optional inputs
 5. Optional modifiers
 6. Optional outputs
 7. Derived outputs
- Once a tool is released, you cannot insert parameters or change parameter order
 - You can put an optional parameter at the end
- Tools built with ArcObjects can have different ordering for dialog vs. scripting

Keywords

- **Keywords should be upper case, no spaces, no special characters other than underscore**
 - **“INVERSE_DISTANCE”, not “Inverse distance”**
 - **Keywords are never localized into another language**
- **Provide default values for keywords and Booleans**

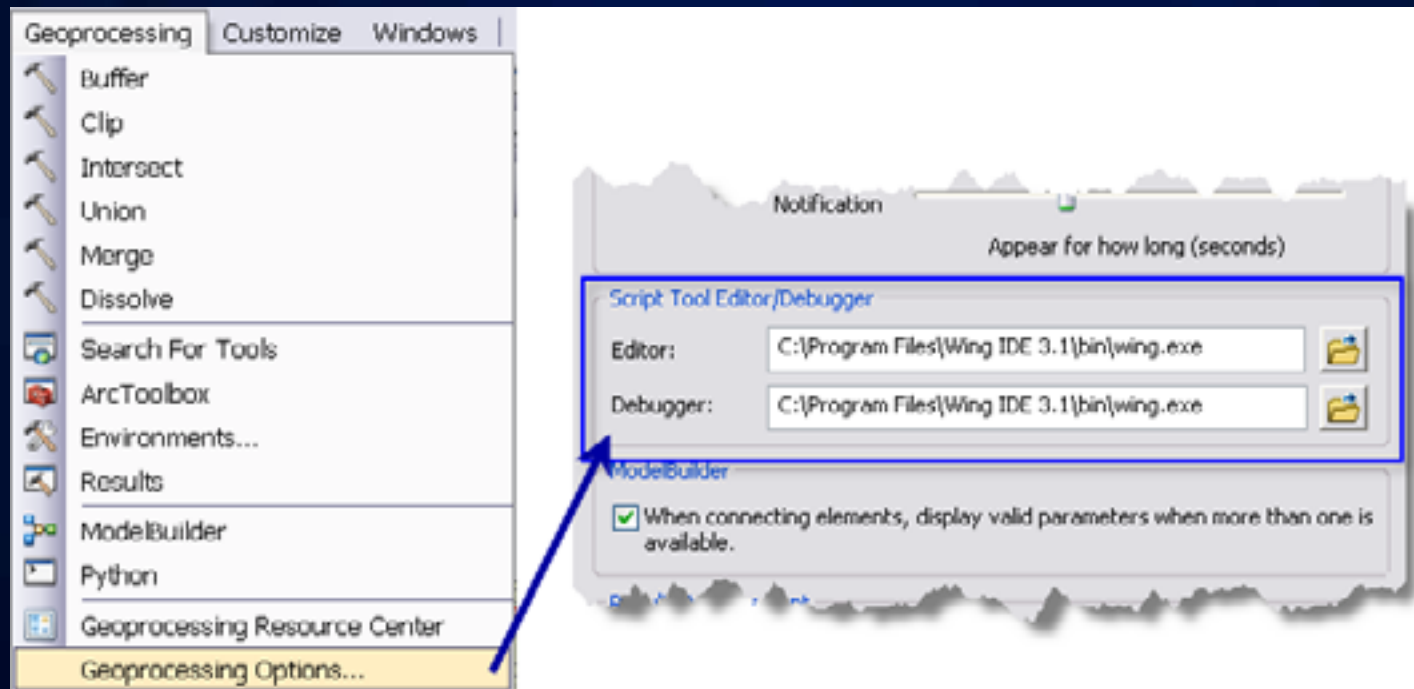
Foreground versus Background processing

- Background processing new in 10.0
- Tools running in background have no knowledge of layers that are not tool parameters
- In order for your tool to run in background:
 - All layers used by your tool must be parameters
 - (You can use any dataset or .lyr file because these reside on disk, not in the application like layers in the TOC)

Embedding code

- Once you've finished developing your script, you can embed it with the toolbox
 - New at 10.0
 - Right click script tool and select "Import Script..."
 - You can password protect the script so that no one can export it

Debugging script code



For 9.3, see blog post [Tips and Tricks – Debugging Python script tools](#)

Adding outputs to map display

- Suppose that within your script you use the Buffer tool
- You want the output of the Buffer tool to be shown in ArcMap
- You will need a script parameter (output parameter) to hold the results of buffer
- If you want the output to be derived (the buffers “auto-magically” appear w/o the user entering the output pathname)
 - Create a derived output parameter
 - In your script, use `SetParameterAsText()` to set the pathname to the output of buffer

Coming at 10.1 – Python toolboxes

- **A Python Class that you implement**
- **Everything needed is in the class:**
 - **Tool names, labels**
 - **isLicensed property**
 - **Parameter definitions**
 - **Validation**
 - **Execution**

Session Evaluations

- www.esri.com/sessionevals

Questions?



esri