

**2008 ESRI User Conference**  
**Technical Workshops**  
August 4–8, 2008

***Please!***  
*Turn OFF cell phones*  
*and paging devices*



# Building Geoprocessing Tools with Python

*Jason Pardy*  
*Dale Honeycutt*

# Questions for you

- First UC?
- Workstation AMLs?
- Have you scripted before?
- Used GP?
- System programming language (COM/C++/C#/VB/Java??)

# Workshop Outline

- **Goal:** Know how to make a script tool
  - Make your script a full-fledged member of the geoprocessing framework
- Overview of Geoprocessing Scripts
- What are Script Tools
- How to Create Script Tools
  - Using Script Tool Parameters
  - Getting Messages
  - Setting Output Messages and progress
- Script Tool Validation
- Using AMLs and EXE's as the source of Script Tools
- Documenting Script Tools

# Learn Python Scripting with ArcGIS

- Geoprocessing Resource Center
  - <http://resources.esri.com/geoprocessing/>
- ArcGIS Desktop Help
  - Specific help and samples for all of the geoprocessor's methods and properties
- ESRI Training has two classes focused on Python
- Python References
  - You should have a good Python reference to augment ArcGIS documentation
  - [Introductory Python References](#)
  - We like:
    - “Learning Python” by Mark Lutz, published by O’Reilly & Associates
    - “Core Python” by Wesley J. Chun, published by Prentice-Hall

# Why Scripting?

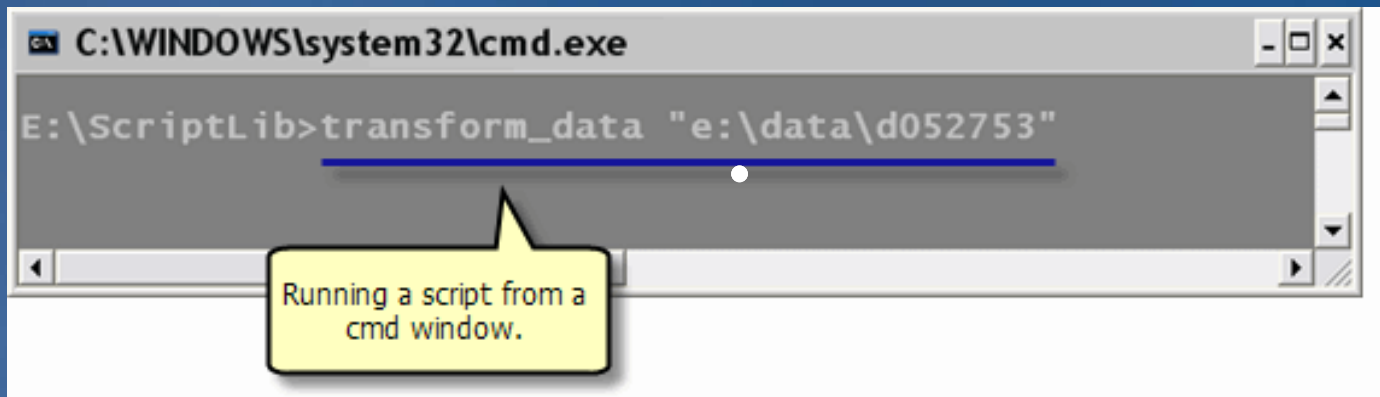
- Scripting is easy to learn & powerful
- Provides an efficient method for defining and executing a series of geoprocessing tools
- More advanced logic than ModelBuilder
- Row by Row access
- No COM development! For GIS professionals.
- Rapid development
- Interoperability (Glue language)
- Some languages are cross platform (Python)
- Access properties of data
- Advanced Error Handling

# ESRI Recommends Python

- Fulfills the needs of our user community
  - “It’s really good”
  - Simple
  - Modular
  - Object oriented
  - Easy to maintain
  - Scalable
  - Cross platform (windows & UNIX/Linux)
  - Integrated Development Environment (IDE)
  - Established and active user community
  - Most geoprocessing examples are available in Python

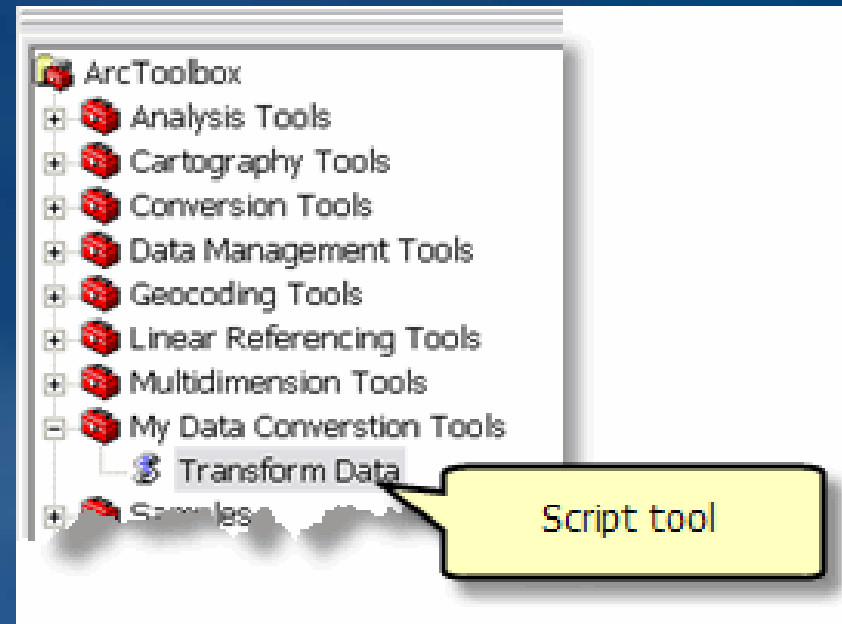
# Stand-alone Scripts

- Can be run independent of ArcGIS applications
- Can be scheduled to run daily/weekly, anytime



# Script Tools

- Source is a script
- It is a tool
  - Use in ModelBuilder
  - Use in Command Line
  - Use in other scripts
  - “Full-fledged member”
- Runs in process (9.3)
- Inherits all geoprocessing properties
- Communicates with application
  - Layers added to map, etc.
  - Messages
- More easily shared
  - Not everyone knows how to run a stand-alone script
    - Puts a familiar face on your work



## Demo – Create a Script Tool

- **CrunchData** <in\_features> <out\_feature\_class>
- In-process execution

# Demo Review

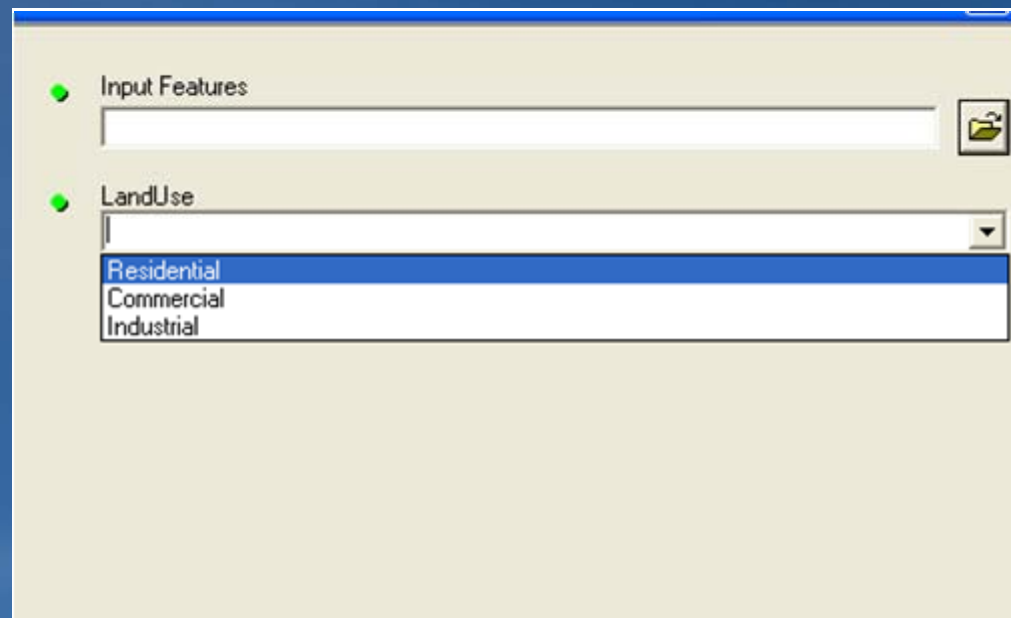
- Parameters are positional
- Parameter types
- Some of the parameter properties
- Messaging – informative, warnings, errors
- Progress (9.3)

# Parameter Type

- Required
  - <input\_featureclass>
- Optional
  - {xy\_tolerance}
- Derived

# Parameter Filter

- Used to filter values
  - Coded Value domain (fixed list of values)
  - Range domain (range of numeric values)
  - Feature type domains
  - Workspace domains

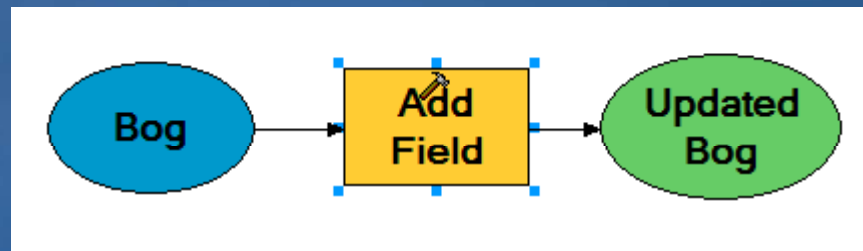


# Filters - Demo

- Filter feature type ( points only)
- Range filter

# Derived output parameter

- Used to indicate when the tool updates the input parameter i.e. AddField
- Does not show up on the dialog or command line
- Used in ModelBuilder to indicate changed state of the value and to provide proper chaining i.e. Add Field



# Obtained From

- Field parameters are commonly dependent on another table/feature class parameter.
  - E.g. The list of fields for a field parameter is dependent on the input table.
- Derived parameters are commonly dependent on an input table or feature class parameter.
  - A derived output with a dependency on the input will automatically set the output parameter's value to be the same as the input dataset in a model.

# Obtained From - Demo

- Derived output parameter
- Fields

# Multivalues

- To handle a list of input values rather than just one value, you should set the MultiValue property to Yes.
  - For example, you might want to input multiple feature classes

**Crunch Data Properties**

General | Source | **Parameters** | Validation | Help

Display Name	Data Type
@ in	Feature Class
out	Feature Class

Click any parameter above to see its properties below.

Parameter Properties

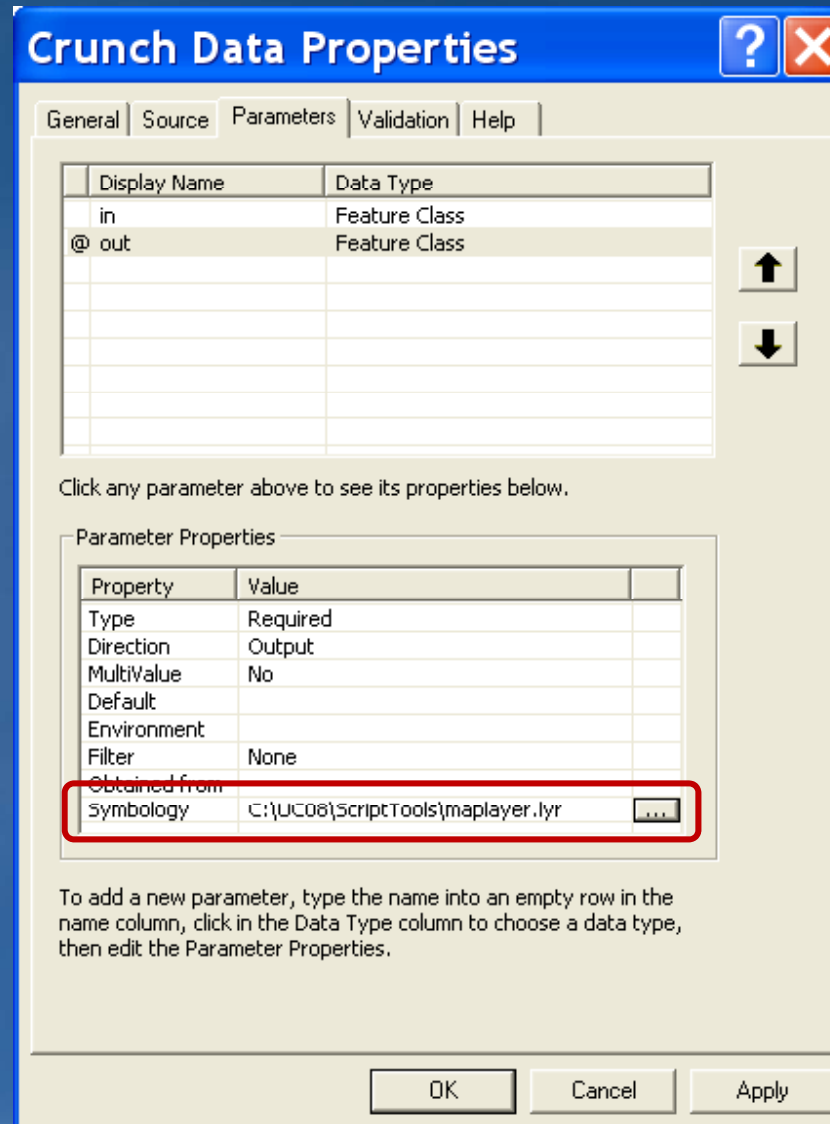
Property	Value
Type	Required
Direction	Input
MultiValue	No
Default	No
Environment	Yes
Filter	none
Obtained from	
Symbology	

Input Features

- C:\Uc2003\Portland\_OR.gdb\streets
- C:\Uc2003\Portland\_OR.gdb\dtwn\_bike\_rte
- C:\Uc2003\Portland\_OR.gdb\downtown

# Multivalued - Demo

# Symbology – ArcGIS 9.3



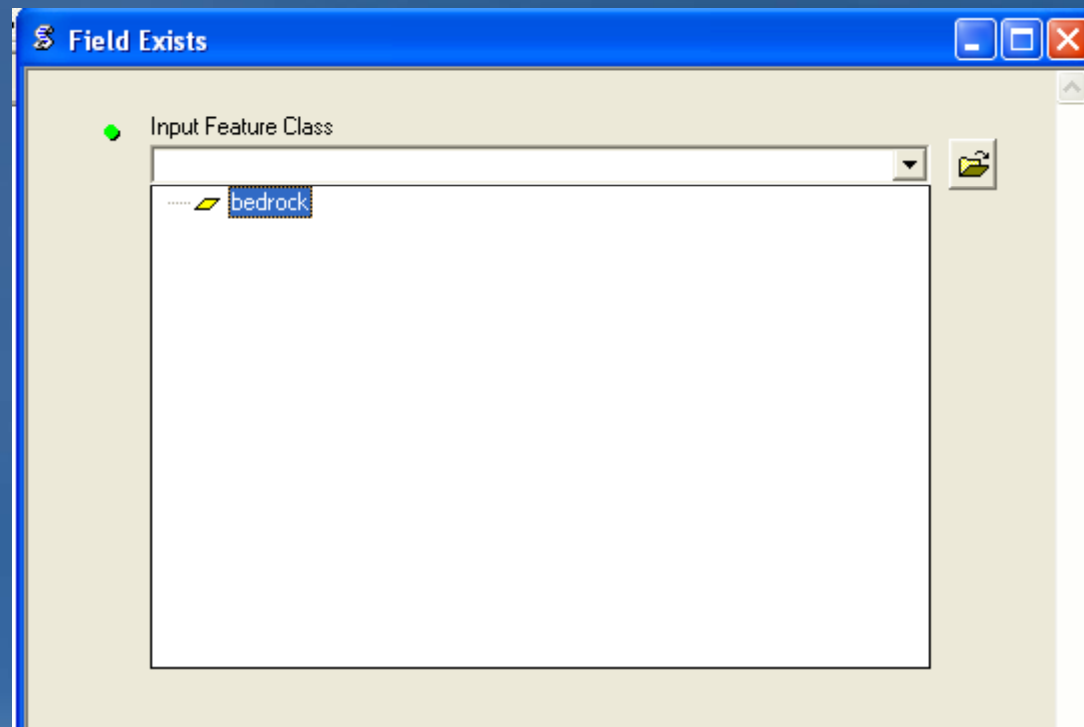
# Digging Deeper – Data Types

# Parameter DataType

- Defines the datatype of each parameter
  - DataTypes: FeatureClass, Table, Rasters, ...
  - FeatureSets, RecordSets
  - Basic Types: String, Double, Boolean, ...
  - Geoprocessing Structures: spatial reference, extent, cell size, remap table, ...
  - Multivalue of any datatype

# Use Layers

- Supporting layers and tables in ArcMap and ArcGlobe:
  - TableView, FeatureLayer, RasterLayer...



# Getting Input Parameter Values

- If a script is the source of a script tool, it can use the *GetParameterAsText* method to access the input parameter values.
  - Recommended instead of the built in python `sys.argv[]`.

```
# creates the geoprocessing object  
import arcgisscripting  
gp = arcgisscripting.create(9.3)  
  
# Get the input feature class or layer  
in_features = gp.GetParameterAsText(0)  
  
# Get the input Field  
in_fieldName = gp.GetParameterAsText(1)
```

# Every DataType has a String Representation

```
“PROJCS['World_Eckert_IV', GEOGCS['GCS_WGS_1984', DATUM['D_WGS_1984',  
SPHEROID['WGS_1984',6378137.0,298.257223563]], PRIMEM['Greenwich',0.0],  
UNIT['Degree',0.0174532925199433]], PROJECTION['Eckert_IV'],  
PARAMETER['False_Easting',0.0], PARAMETER['False_Northing',0.0],  
PARAMETER['Central_Meridian',0.0], UNIT['Meter',1.0]]”
```

- Except:
  - FeatureSet
  - RecordSet

# Getting Input Parameters - GetParameter

- The *GetParameter* method returns an object.

## # Create the Geoprocessor

```
import arcgisscripting  
gp = arcgisscripting.create(9.3)
```

## # Get a FeatureSet object

```
FS = gp.GetParameter(0)
```

## # Run Buffer

```
gp.Buffer_analysis(FS, outputFC, "100 METERS")
```

# Setting Output Parameters

- Script tools must define their outputs so tools work as expected in models and on the command line.
- Script tools must populate the values of their output parameters so they can be used as input to other tools in a model.
  - E.g. Scalar, Boolean, or String return values
- The *SetParameterAsText* method will set the value of an output parameter using a text string.

# Setting Output Parameters - SetProperty

- The *SetProperty* method sets the specified parameter property using an object.

## # Create the Geoprocessor

```
import arcpy
```

```
gp = arcpy.CreateGeoprocessor(9.3)
```

```
infc = gp.GetParameterAsText(0)
```

```
SR = gp.Describe(infc).SpatialReference
```

```
gp.SetParameter(1, SR)
```

# Setting Output Parameters - Preconditions

- Setting output parameter values can be used to set precondition values for other processes in the model (Branching)

```
#Check if field exists and set the parameters to true or false
```

```
if bFieldExists == 1:
```

```
    gp.SetParameterAsText(3, "True")
```

```
    gp.SetParameterAsText(4, "False")
```

```
# If the field doesn't exist, the value for the second derived  
output
```

```
# parameter (Does Not Exist) is set to true. The branch of the  
model # that runs Make Layer, Add Join, then Select is  
executed.
```

```
else:
```

```
    gp.AddMessage("Field does not exist.")
```

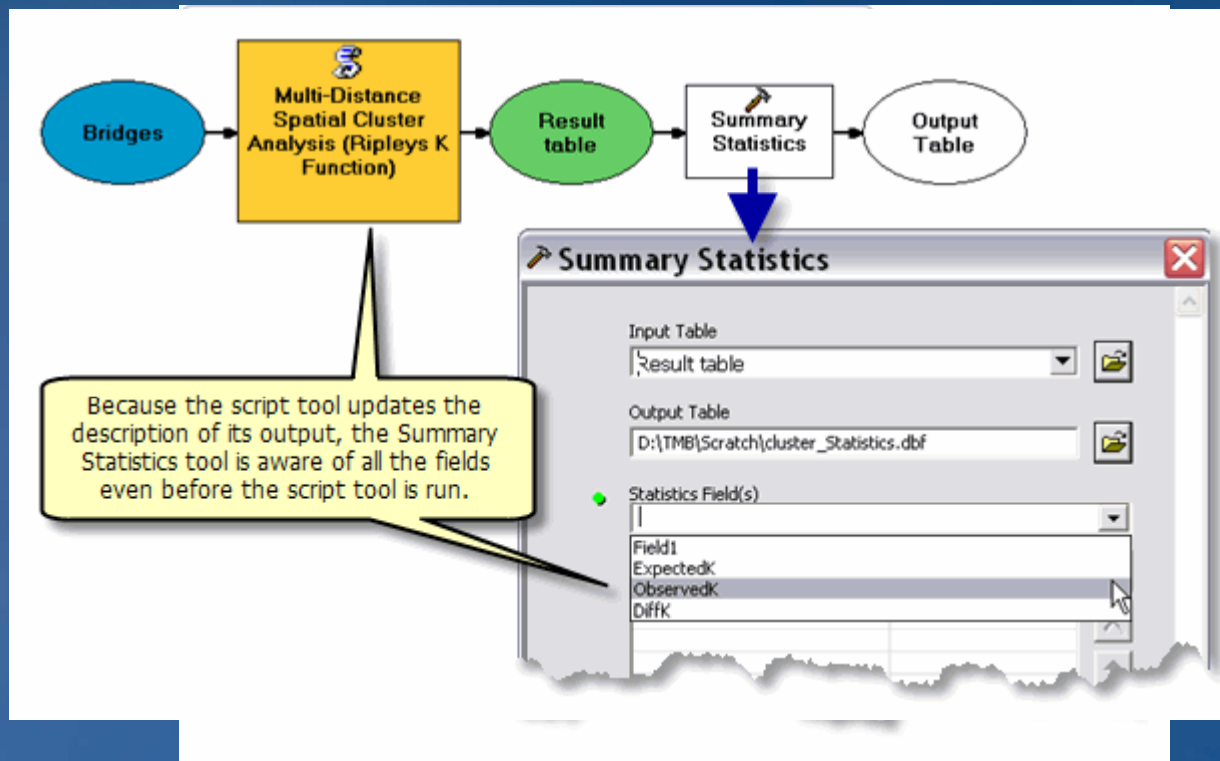
```
    gp.SetParameterAsText(3, "False")
```

```
    gp.SetParameterAsText(4, "True")
```

# Default Script Tool Validation

- Script tools do provide basic validation:
  - Have all the required parameter values been supplied?
  - Are the values of the appropriate data types?
  - Does the input or output exist?

# At 9.3, you can control validation



# Tool Validator Class

- **InitializeParameters()** – essentially the same as the properties page
- **UpdateParameters()** – called **BEFORE** basic validation
- **UpdateMessages()** – called **AFTER** basic validation

The screenshot shows the ArcGIS Tool Validator dialog box with the Python code editor open. The code defines the `ToolValidator` class with methods `__init__`, `initializeParameters`, `updateParameters`, and `updateMessages`. The `__init__` method sets up the Geoprocessor and the list of tool parameters. The `initializeParameters` method refines the properties of a tool's parameters. The `updateParameters` method modifies the values and properties of parameters. The `updateMessages` method modifies the messages created by internal validation.

Callout 1: Click the Edit button to open the Python editor.

Callout 2: When finished editing, click OK or Apply to replace the current ToolValidator code with your edited code.

```
class ToolValidator:
    """Class for validating a tool's parameter values
    the behavior of the tool's dialog."""

    def __init__(self):
        """Setup the Geoprocessor and the list of tool
        import arcgisscripting as ARC
        self.GP = ARC.create(9.3)
        self.params = self.GP.getparameterinfo()

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

    def updateParameters(self):
        """Modify the values and properties of paramete
        validation is performed. This method is called
        has been changed."""
        return

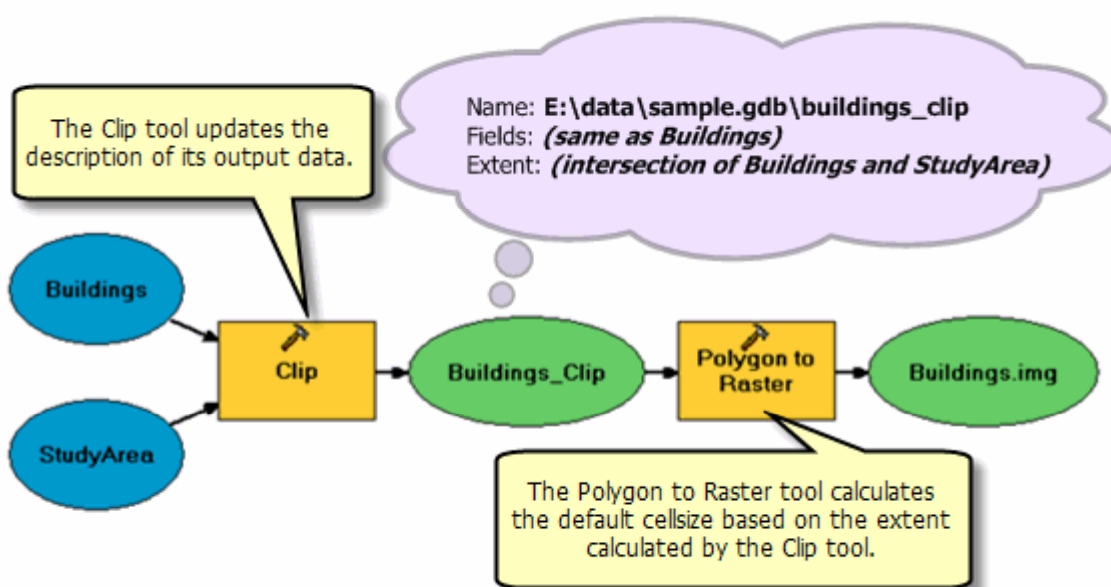
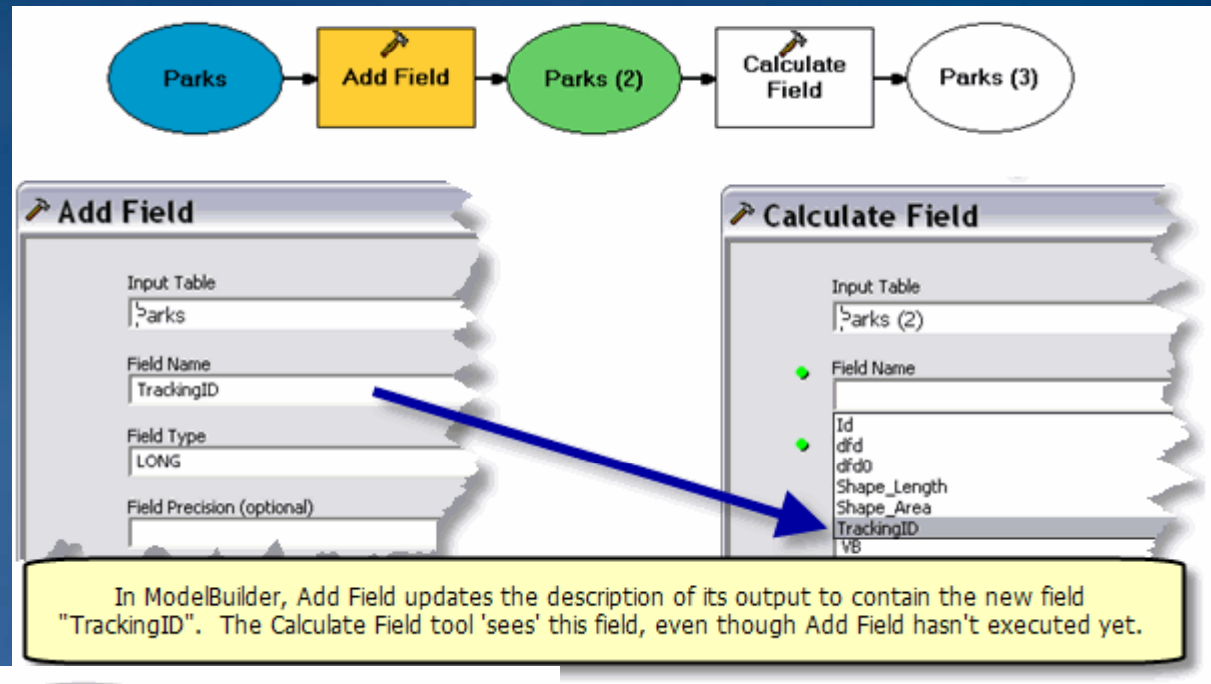
    def updateMessages(self):
        """Modify the messages created by internal vali
        parameter. This method is called after intern
        return
```

# Demo – Multiring Buffer

# Here's the doc, lots of examples

- [http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Customizing\\_script\\_tool\\_behavior](http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Customizing_script_tool_behavior)

# Validation important for ModelBuilder chaining



# Getting Messages

- Executing a tool will produce messages. These can be:
  - Informative messages
  - Or warning messages
  - Or error messages
- Scripts can trap for errors (using standard TRY EXCEPT blocks)
- The *GetMessages* method will return all messages for the specified severity (0=informative, 1=warning, 2=error, ()=ALL).

```
try:  
    gp.Clip("roads","urban_area","urban_roads")  
except:  
    # shows only the error messages if Clip fails  
    print gp.GetMessages(2)
```

# Setting Output Messages

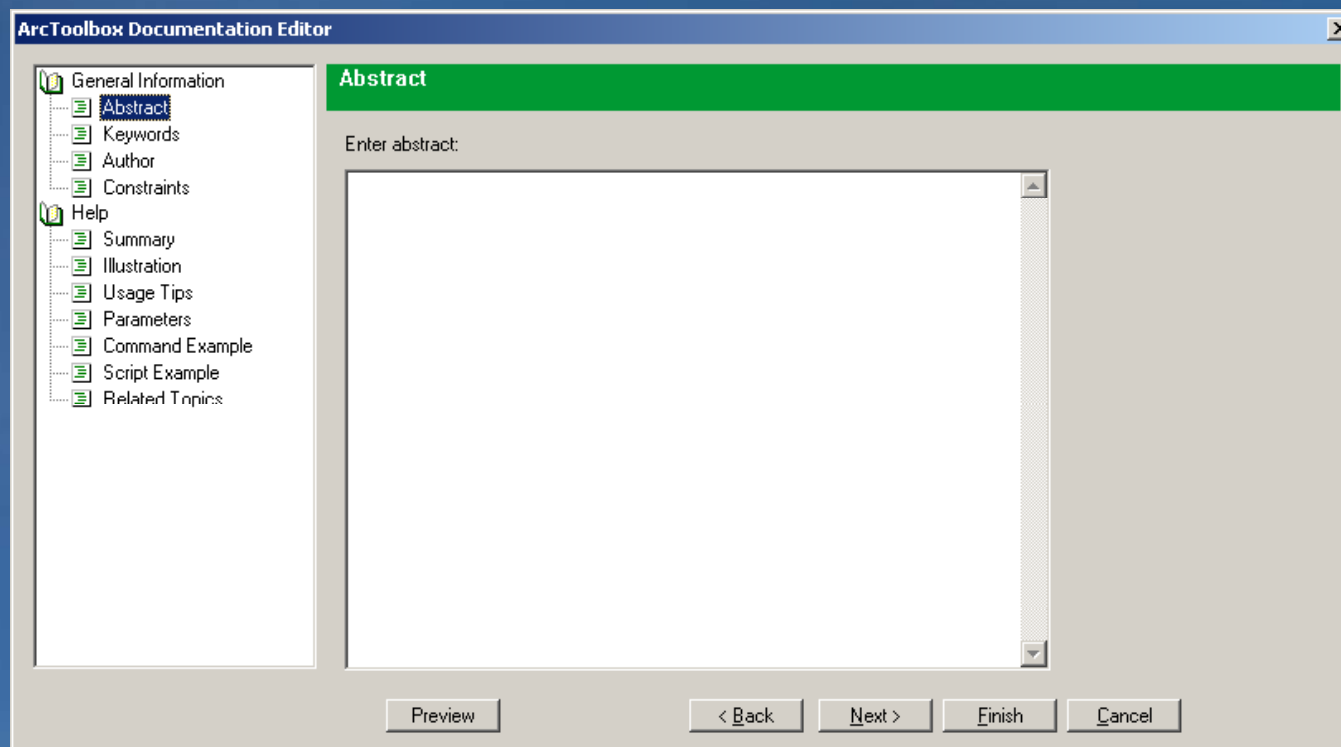
- When a script tool is executed, messages often need to be returned to the user, especially when problems arise.
- The geoprocessor has several methods for adding messages:
  - [AddMessage](#) (message string)
  - [AddWarning](#) (message string)
  - [AddError](#) (message string)
- Messages added to the geoprocessor are immediately returned to the application or script executing the tool.

# Using AMLs with Scripts Tools

- It's possible to use AMLs as the source to a new geoprocessing script tool.
- An ArcInfo license and ArcInfo Workstation is required to be installed.
- It is possible to run commands from other prompts within ArcInfo Workstation, such as GRID, TABLES, ArcPlot, ArcEdit, etc., but the AML must start at the Arc: prompt.
- There are no restrictions.
  - Display 9999, &Menu, etc. work
- The AML file extension through the windows file type manager must execute Arc
  - ESRI provides a registry setting file to facilitate this.

# Documenting Script Tools

- Don't forget to document your new script tools.
- Providing good documentation for your tools is the first step to sharing your tools with others.
- The Documentation Editor is where you enter and manage documentation for your tools. Help is stored with the tool.



# Summary

- **Scripts can be tools.** Script tools behave like all other Geoprocessing tools.
- Script tools must define output parameters.
- Script tools provide basic validation .
  - Script tools can now populate the properties and values of output parameters.
- Script tools should return messages – good practice.
- AMLs can be script tools.
- Document your script tools.

**Thank-you. Questions?**

**Please fill out survey**

# Analysis & Geoprocessing Sessions

Wednesday, 06. August 2008

- Spatial Analysis Island - Demo Theater Presentations

9:00-10:00 – Working with Environment Settings

11:00-12:00 – What's new with Python scripting in ArcGIS 9.3

5:00- 6:00 – Python Utility Modules for script tool development in ArcGIS  
9.3

# Analysis & Geoprocessing Sessions

Thursday, 07. August 2008

- **Technical Workshops**

- 8:30- 9:45 – **Python Scripting – An Introduction II • Rm 5A/B**

- 10:15-11:30 – **Python Scripting – Advanced Techniques II • Rm 5A/B**

- 1:30- 2:45 – **Building Geoprocessing Tools with Python II • Rm 4**