

## Advanced GIS Lab – Lab 7

Adapted from:

### *ESRI Virtual Campus Course – Geoprocessing for ArcGIS 10.1*

**Due:** Monday, October 27, 2014

#### **Objectives:**

##### *Module 4: Getting more out of geoprocessing*

- Understand the basics of how scripts work.
- Open a script in its native application and recognize standard script elements.
- Attach a script to a geoprocessing tool.
- Run a script tool as a standalone process or as part of a model.
- Use script tools to perform batch processing, simplify complex models, and add functionality to models.
- Expose script parameters as model parameters.
- Use a script to test a condition in a model

##### *Module 5: Documenting your work*

- Understand the different types of documentation that can be created with ArcGIS® Desktop.
- Create and interpret a report of a model.
- Document a model using descriptive labels.
- Create metadata for models and script tools.
- Create help for models and script tools.
- Describe some key factors regarding the sharing and distribution of geoprocessing work.

#### Assignment (continuation of Lab 5)

Geoprocessing is one of the most powerful components of a GIS. In ArcGIS Desktop, you are provided with a framework for addressing geoprocessing tasks, which includes an extensive list of geoprocessing tools organized within a set of toolboxes. You can employ the tools directly or chain them together to model a particular workflow. You can put geoprocessing tools to work in custom scripts and you can create your own tools and toolboxes. The geoprocessing framework also provides functionality for organizing and managing your work environment, performing simple and complex analyses, and making your custom tools usable by others. This course will give you the foundation you need for geoprocessing with ArcGIS Desktop.

## **Module 4: Getting More out of Geoprocessing**

In this module, you will be introduced to using Python scripts for geoprocessing. Python is the scripting environment included with ArcGIS. You will not have to write any scripts, but you will learn how to use them to add even more power and flexibility to your workflows.

Although you do not need to use scripts for geoprocessing, you will find that they can help to streamline your work and even add custom functionality.

Once you learn some basics about using scripts for geoprocessing, you will revisit the course project that you have been working on in the previous two modules. Specifically, you will use scripts to enhance the lynx habitat suitability model you created in module 3.

### **Introduction to scripting**

A script is a set of instructions in plain text, usually stored in a file and interpreted, or compiled, at run time.

In geoprocessing, scripts can be used to automate tasks, such as data conversion, or to generate geodatabases and they can be run from their scripting application or added to a toolbox.

Geoprocessing scripts can be written in any COM-compliant scripting language, such as Python, JScript, or VBScript.

You may already know more about scripts than you think. In the previous module, you worked with visually diagrammed scripts—better known as models. Using ModelBuilder, you arranged variables and tools in a logical sequence and ran them as a single operation. Any model that you create in ModelBuilder can be exported as a script file.

In this topic, you will get a brief introduction to Python scripts and learn how to recognize a few key elements. You will also learn how to run a script as a standalone operation, as a tool, and how to incorporate a custom script into a model.

### **Why use scripts?**

Scripts are similar to models in that they allow you to chain processes together and run those processes in a certain order. Also like models, scripts can be used to build functionality that might not be available from the system tools in ArcGIS Desktop.

For example, you can build custom batch processes to run through a list of datasets and perform multiple operations on each one until the list is complete. You will see how looping works in scripts in the next exercise.

You can also use scripts to process data based on certain conditions. For instance, a script can check to see if the input data is a line or a polygon feature class and then process the data accordingly. You will also see how branching works in scripts later in the module.

### **The anatomy of a script**

Even if you never write a line of code yourself, you may want to use scripts that others have written. Keep in mind that with the proper licenses and extensions, you can execute any of the system tools from a script. For this reason, you should be able to recognize some of the basic elements in a geoprocessing script, such as comments, standard code, script arguments, environment settings, and tools.

### *Comments*

In Python, any line of code preceded by one or many number signs (#) is a comment.

When you run a script comments are ignored, and only the uncommented lines of code execute.

### *Standard code*

A Python script that uses geoprocessing functionality must, at minimum, start with the following line of code:

```
import arcpy
```

### *Script arguments*

Also called system arguments, this code allows users to interact with the script and specify values for variables like output workspaces.

Below are three script arguments with comments:

```
# Argument 1 identifies the input workspace where the input
# feature classes are stored
inputWS = arcpy.GetParameterAsText(0)

# Argument 2 identifies the feature class you'll use to clip
# the input features
clipFC = arcpy.GetParameterAsText(1)

# Argument 3 identifies the output workspace where the clipped
# feature classes will be saved
outputWS = arcpy.GetParameterAsText(2)
```

Each of these script arguments must be satisfied in order for the script to run successfully. Script arguments are similar to the parameters you fill out in a dialog box or a model.

### *Environment setting*

A setting that applies to all processes run by a script, such as the default workspace.

Below is an environment setting with a comment:

```
# Set the current workspace
arcpy.env.workspace = inputWS
```

You can use environment settings in a script to override environment settings set in ArcToolbox.

### *Tools*

Any system tool can be run from a script by specifying the tool name plus the toolbox alias and the tool's parameters.

Below is code (with comments) that loops through a list of feature classes and "clips" them using the Clip tool. Can you find the tool?

```

# Loop through the list of feature classes in the input workspace
# clip them with the clip feature class and store them in the
# output workspace
fc = fcs.next()
while fc != "":
    # Set the output name for each feature class to be the same
    # as the input but stores it in the output workspace
    outputName = outputWS + "\\\" + fc
    # Print each clipping operation to the screen
    print "clipping", fc, "with", clipFC, "to produce", outputName
    arcpy.Clip_analysis(fc, clipFC, outputName, "")
    fc = fcs.next()

```

The Clip tool includes the *arcpy.* prefix and uses the scripting syntax for the tool:

Clip_analysis (in_features, clip_features, out_feature_class, cluster_tolerance)	
Parameter	Explanation
in_features (Required)	The features to be clipped.
clip_features (Required)	The features used to clip the input features.
out_feature_class (Required)	The feature class to be created.
cluster_tolerance (Optional)	<b>Cluster tolerance</b> is the distance range in which all vertices and boundaries in a feature class are considered identical or coincident. To minimize error, the cluster tolerance chosen should be as small as possible, depending on the precision level of your data. By default, the minimum possible tolerance value is calculated in the units of the spatial reference of the input.

## Using scripts for geoprocessing

You can run a script as a standalone operation or you can add a script to a toolbox and then run the script from its dialog box or incorporate it into a model.

Running a script as a standalone operation means that you can accomplish geoprocessing without ever opening an ArcGIS Desktop application. For example, you can run scripts from within a script editor, like PythonWin, or from the command prompt.

To run a geoprocessing script as a standalone operation, you should make sure you or the person you are sharing the script with has:

1. An available license to use ArcGIS Desktop
2. A license to use the geoprocessing tools in the script
3. Python installed (if you're using a Python script)
4. Access to any data and workspace referenced in the script

If you want to run a script from within an ArcGIS Desktop application, like a system tool, or use it in a model, you must first add it to a toolbox and make it a script tool.

### *Exercise 1: Run a geoprocessing script*

You can run a geoprocessing script as a standalone operation outside of ArcGIS or you can add the script to a toolbox and run it like any other tool.

In this exercise, you will take a brief departure from the lynx project data you have been working with in previous exercises so that you can focus on the nuances of running a script as a standalone operation, as opposed to running it as a script tool.

First, you will learn how to run a geoprocessing script in Python. Next, you will add the same script to a toolbox, make it a script tool, and run it from ArcCatalog.


Later in this module, you will return to the lynx project and learn how to incorporate a script tool into a geoprocessing model.

Estimated time to complete: 20 minutes

#### **Step 1: Open a geoprocessing script in PythonWin**

In this step, you will start PythonWin and open the ClipAll.py script.

Click the Windows button and type PythonWin to open it.

In PythonWin, click the Open button . In the Open dialog box, navigate to your **Geoprocess\More\Florida** folder.

Click **ClipAll.py**, then click Open.

If you take a moment to scroll through the script, you will notice it begins with some descriptive *comments*.

Next, *standard code* tells Python to use the Geoprocessor object in the script.

You also see *environment settings* and *script arguments*. Remember, the script arguments allow user input when the script is run.

And, at the end of the script, you see some code that generates a list and then loops through it, clipping each feature class with the *Clip\_analysis tool*, and putting each output feature class in the specified workspace.

#### **Step 2: Identify script arguments**

PythonWin is used primarily to write and debug scripts, but you can use it to run scripts, too. If the geoprocessing script you're running contains script arguments, like the ClipAll.py script does, you will need to supply the values for those arguments when you run the script.

In PythonWin, arrange the windows so that you can see both the ClipAll window and the Interactive Window.

With the ClipAll window active, click the Run button .

In the Run Script dialog box that appears, notice that the Arguments text box is empty. You must supply the proper values for those arguments in order for the script to run successfully.

In this case, you will need to enter full paths for each argument. Because the full path of each argument is rather long, you will copy/paste them from ArcCatalog to avoid typing errors.

### Step 3: Copy and paste paths for script arguments

Start ArcCatalog.

**Tip:** You do not need ArcCatalog open to run the script—you are just using it here to copy the correct paths for each argument and paste them into the Arguments text box in the PythonWin Run Script dialog box. You might find this trick handy whenever you need to capture paths, particularly when you need to point to feature classes and tables in a geodatabase.

In the Catalog tree navigate to your **\Geoprocess\More** folder. Expand the **Florida** folder, then expand both the **MarionCounty** and **StudyArea** geodatabases.

Make sure the Location toolbar is visible in ArcCatalog.

In the Catalog tree, click **MarionCounty.gdb**.

In the Location toolbar, click the path to highlight it, then right-click and choose Copy (or press Ctrl+C).

In PythonWin, right-click in the Arguments text box (in the Run Script dialog box) and choose Paste (or press Ctrl+V).

This path is the value for the input workspace argument (inputWS = arcpy.GetParameterAsText(0)).

In the Catalog tree, click the **StudyBoundary** feature class in the StudyArea geodatabase.

Copy the path from the Location toolbar.

In PythonWin, press the space bar to add a space at the end of the path for the input workspace argument.

**Note:** *You must insert a space between each argument.*

Press Ctrl+V to paste the path to the feature class.

This is the value for the clip feature class argument (clipFC = arcpy.GetParameterAsText(1)).

Finally, in the Catalog tree, click **StudyArea.gdb** and copy the path from the Location toolbar.

In PythonWin, insert a space after the last argument and paste the path to the output workspace.

This is the value for the output workspace argument (outputWS = arcpy.GetParameterAsText(2)).

With the required arguments for the script complete, you are now ready to run the script.

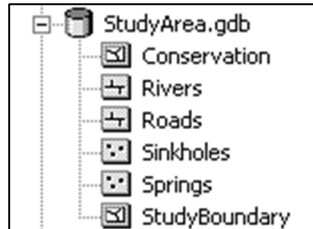
#### Step 4: Run the script

Click OK in the Run Script dialog box.

The script runs and after a few moments (\*\* This may take several minutes to run) you will see some processing statements in the Interactive Window.

Once the last feature class has been clipped and the script is finished running, return to ArcCatalog.

In the Catalog tree, right-click StudyArea.gdb and choose Refresh.



If you like, compare the feature classes in the MarionCounty and StudyArea geodatabases by previewing them.

The new feature classes in the StudyArea geodatabase are the result of running the script from PythonWin. You did not need to open an ArcGIS Desktop application, like ArcMap or ArcCatalog, to do geoprocessing.

Close PythonWin.

To finish this exercise, you will add this script to a toolbox and use it like any other geoprocessing tool.

#### Step 5: Add the script to a toolbox

In the ArcCatalog Catalog tree, right-click the ScriptTools toolbox (in MarionCounty.gdb), point to Add, then choose Script.

The Add Script wizard displays.

For Name, type **ClipAll**, and for Label, type **Clip All Feature Classes**.

**Note:** As with models, spaces and special characters are not allowed in script tool names, but they are allowed in labels, which determine how the script will appear in a toolbox.


Copy the following description and paste it into the Description text box:

**This script tool clips all feature classes in a workspace and puts the resulting feature classes in another workspace.**

Check the box next to "Store relative path names."

Using relative paths gives you the ability to share the script and script tool with others, or to move them to a different location on your system.

Click Next.

Click the Browse button  and navigate to your **Geoprocess\More\Florida** folder.

Click **ClipAll.py** and click Open.

Click Next.

Now you need to specify the three tool parameters—the three text boxes that users see when they run the tool. The parameters are dictated by the script.

Click in the cell below Display Name and type **Input Workspace**.

This is the text that the user will see above the text box.

Click in the cell below Data Type and choose Workspace from the drop-down list.

This determines which type of data will be accepted for the parameter.

In the Parameter Properties area, default values have automatically been provided for three of the properties. The default values are correct in this case—the user must fill in this parameter (Type: Required); the parameter is used to provide input for the process (Direction: Input); and the parameter cannot accept multiple values (MultiValue: No).

Now add the second parameter.

Under Display Name, type **Clip Feature Class**.

For Data Type, choose Feature Class.

Accept the default parameter properties.

Finally, add the third parameter.

Under Display Name, type **Output Workspace**.

For Data Type, choose Workspace.

Accept the default parameter properties.

**Note:** It may seem confusing initially that the Direction is specified as "Input" for the Output Workspace. When you are specifying parameter properties, input is any data that already exists. Output data is any data that gets created as a result of the tool being run. If you were to choose Output, the script would expect to create a new workspace. However, in this case, you are asking the user to specify an existing workspace. In other words, the existing output workspace is chosen through user *input*.

The screenshot shows the 'Add Script' dialog box with a table for parameter configuration. The table has two columns: 'Display Name' and 'Data Type'. The first row is filled with 'Input Workspace' and 'Workspace'. Below the table are up and down arrow buttons. A text prompt says 'Click any parameter above to see its properties below.' Below this is a 'Parameter Properties' section with a table of properties and values. The properties listed are Type (Required), Direction (Input), MultiValue (No), Default, Environment, Filter (None), and Obtained from. At the bottom of the dialog are buttons for '< Back', 'Finish', and 'Cancel'.

Display Name	Data Type
Input Workspace	Workspace

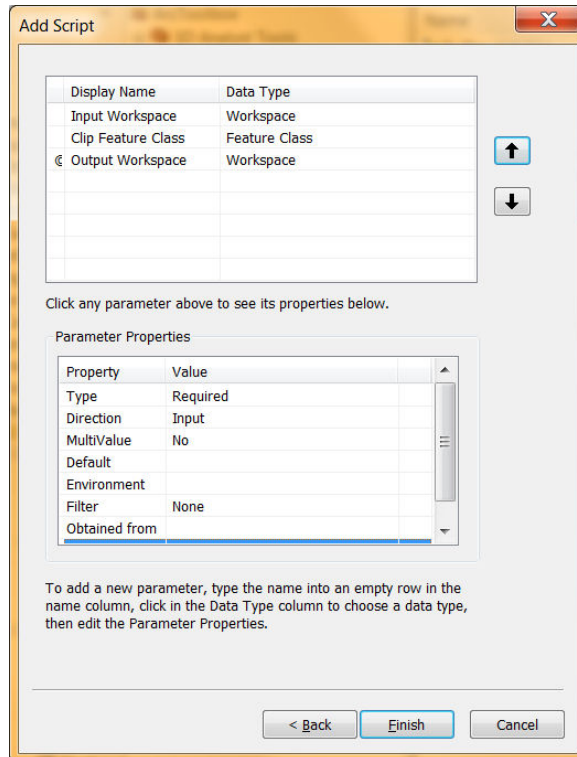
Click any parameter above to see its properties below.

Property	Value
Type	Required
Direction	Input
MultiValue	No
Default	
Environment	
Filter	None
Obtained from	

To add a new parameter, type the name into an empty row in the name column, click in the Data Type column to choose a data type, then edit the Parameter Properties.

< Back Finish Cancel





Click Finish.

The new script tool is added to the ScriptTools toolbox.

Expand the toolbox to see the new tool.

This script tool can be used like any other tool.

It is important to understand that the Script Tool wizard and the script are independent of one another. The wizard does not know which input and outputs the script requires. If you provide it with incorrect information (in other words, information that conflicts with the code in the script), the tool will either not work or it will not work correctly.

### Step 6: Check your geoprocessing options

Before you run this new script tool, verify that the geoprocessing options to "Overwrite the outputs of geoprocessing operations" and "Log geoprocessing operations to a log file" are both checked.

Remember that this new script tool is going to run the same ClipAll.py script that you ran in PythonWin in the step above. If the option to overwrite the outputs of geoprocessing operations is not set, you will get an error because those clipped feature classes already exist in the StudyArea geodatabase.

### Step 7: Run the script tool

To make sure that you have set up the parameters correctly, you will run the tool.

In the Catalog tree, double-click the Clip All Feature Classes script tool you created to open its dialog box.

The three parameters you set in the previous step are displayed in the dialog box, ready for user input.

**Tip:** Remember, with a dialog box, instead of typing or copy/pasting pathnames for these parameters, you can simply drag the object from the Catalog tree and drop it on the appropriate text box in the dialog box.

Fill in the three parameters as follows:

- Input Workspace: **MarionCounty.gdb**
- Clip Feature Class: **StudyBoundary** (feature class in StudyArea.gdb)
- Output Workspace: **StudyArea.gdb**

Click OK to run the script.

Behind the scenes, the script runs in its native scripting application, but the processing information is reported in the progress window like any other tool.

When processing is complete, right-click StudyArea.gdb and choose Refresh.

If necessary, expand the StudyArea geodatabase to see the new feature classes.

If you are continuing to the next exercise, leave ArcCatalog open. Otherwise, close ArcCatalog.

### *Exercise 2: Export a model to a script*

In the previous exercise, you learned that script arguments allow users of the script to interact with it, whether they run the script as a standalone operation or use it as a script tool. In this exercise, you will look at a Python script and determine what the script tool parameters should be.

To better understand how to set parameters for a script tool, you will start with something familiar—a simple model that has model parameters. By creating a script from the model, you will have a script that does exactly the same thing that the model does. You will use what you learned in the previous module to decipher the script.

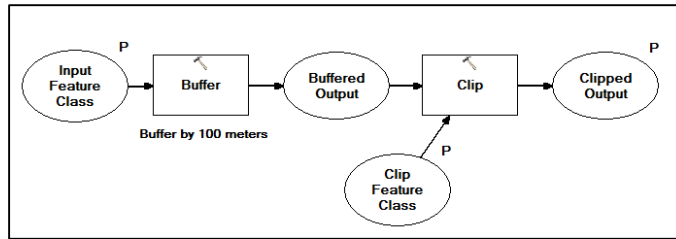
Estimated time to complete: 20 minutes

#### **Step 1: Open a model**

If necessary, start ArcCatalog.

In your **More\Florida** folder, make sure the MarionCounty geodatabase and the ScriptTools toolbox are expanded.

Right-click the Buffer and Clip model and choose Edit. If you get an error message, press OK.



This is a template model. No data has been added, so the model cannot be run. However, you can use it to create a script.

The model contains two familiar processes. The first process will buffer the features in the input feature class by 100 meters. (If you open the Buffer tool dialog box, you will see that the Distance parameter has been set to 100 meters.) The second process will clip the buffered features using another feature class.

Three of the variables in the model have been set as model parameters. As you may recall from the previous module, variables are marked as model parameters so that they can be changed when the model is run using a dialog box. Using model parameters gives the model flexibility.

The BufferedOutput variable, which is not a model parameter, is intermediate data that can be deleted when processing is complete. If you open the dialog box for the variable, you will see that the output feature class name is BufferTemp.

## Step 2: Export the model to a script

To see the code behind the model, you will export the model to a script.

From the Model menu, choose Export > To Python Script.

In the Save As dialog box, navigate to the **More\Florida** folder. Name the file **BufferAndClip**.

Click Save.

The model is exported to a Python script.

In the Catalog tree, right-click the Florida folder and choose Refresh to see the script file.

*If you have refreshed the folder and you still do not see the script you just created, you need to set the option for ArcCatalog to display Python files in the Catalog tree.*

1. From the Customize menu, choose ArcCatalog Options.
2. In the Options dialog box, click the File Types tab, then click New Type.
3. In the File Type dialog box, for File extension, type **py**.
4. Click in the Description of type input box and type "Python File" if it does not automatically populate.
5. Click OK to close the File Type dialog box, then click OK again to close the Options dialog box.
6. Refresh the Florida folder.

*ArcCatalog should now display Python scripts.*

Leave the Buffer and Clip model open and minimize ArcCatalog.

### Step 3: Examine the script in PythonWin

Open up PythonWin.

In PythonWin, open BufferandClip.py from your More\Florida folder.

The script contains the code to run the same processes run by the model and comments (green italic text) that document what the code does.

Take a minute to compare the script to the model it was created from.

You should be able to identify the four variables (shown as script arguments and local variables) and the two tools.

### Step 4: Find the script arguments

When you add a script to a toolbox, the Add Script wizard prompts you for the name of the script and any parameters for the script tool. Therefore, you will need to determine the appropriate parameters as dictated by the script.

Notice the three lines of code under the Script arguments section of the script. These lines specify script arguments and the variables that will contain the values for the script arguments. (Remember, script arguments are also called system arguments, represented in the code as `arcpy.GetParametersAsText`).

The script arguments correspond to the three elements that were marked as model parameters in the model. Script arguments serve the same purpose as model parameters—they allow parameter values to be changed in the script tool's dialog box.

For each script argument, you will need to set a parameter in the script tool. The number in the parenthesis next to the script argument identifies the order in which the parameters must be added.

In the Add Script wizard, you will specify three parameters.

The first parameter will be the input feature class for the Buffer process.

The second parameter will be the Clip feature class—the feature class that will be used to clip the buffered output.

The third parameter will be the final output, the clipped feature class.

**Note:** In the script, a default path and feature class name are provided for this parameter in case this information is not provided when the script is run. However, you will specify that this is a required parameter. Therefore, the code creating the default value becomes irrelevant.

The Buffered\_Output variable has its path and name specified in the script (in other words, it has been *hard coded*). That means when the script is run, you cannot change this variable. The output feature class will be written to StudyArea.gdb and it will be named BufferTemp.

However, the 10.2 export to script tool seems to have a problem with the Buffered\_Output variable as when it is exported from the model, this has been set to be the same as the input feature class variable. Using this in its current form will not work

and will result in an error when the buffer tool is executed. To correct this follow the steps below...

1. Go to ArcCatalog and select the \\More\\Florida\\StudyArea.gdb
2. Copy the Study Area geodatabase location
3. Go to the BufferAndClip script in PythonWin
4. In line 24, there is a call setting the Buffer\_Output variable to the Input\_Feature\_Class. You need to set it to the path that you copied for the StudyArea.gdb, but add \\BufferTemp to the end:  
Buffer\_Output = "...StudyArea.gdb\\BufferTemp"
5. Save your script

Minimize PythonWin.

### Step 5: Add a script to a toolbox

Restore ArcCatalog.

Close the ModelBuilder window.

In the Catalog tree, expand the MarionCounty geodatabase.

Right-click the ScriptTools toolbox, point to Add, and choose Script.

In the Add Script wizard, name the tool **BufferClip** and label it **Buffer and then Clip**.

Check the box next to "Store relative path names."

Click Next.

Add the **BufferandClip** script from the **Florida** folder.

Click Next.

Now you will set up the three parameters—one for each of the script arguments in the same order that they appear in the script. If you want, you can refer back to the script in PythonWin to remind yourself about the order of the arguments and how they are used.

Click in the cell below Display Name and type **Input Feature Class**.

For Data Type, choose Feature Class from the drop-down list.

Under Parameter Properties, default values have been filled in for three of the properties. These default values are correct for the Input Feature Class parameter—the user must fill in this parameter (Type: Required); the parameter is used to provide input into the process (Direction: Input); and the parameter cannot accept multiple values (MultiValue: No).

Now you'll add the second parameter.

For Display Name, type **Clip Feature Class**. For Data Type, choose Feature Class.

The default parameter properties are also appropriate for this parameter.

Finally, you'll add the third parameter.

For Display Name, type **Output Feature Class**.

For Data Type, choose Feature Class.

This time, you need to change one of the parameter properties.

For the Direction property, click in the Value column and choose Output from the drop-down list.

By choosing Output for the Direction property, you are specifying that the data for this parameter will be created as a result of the tool being run.

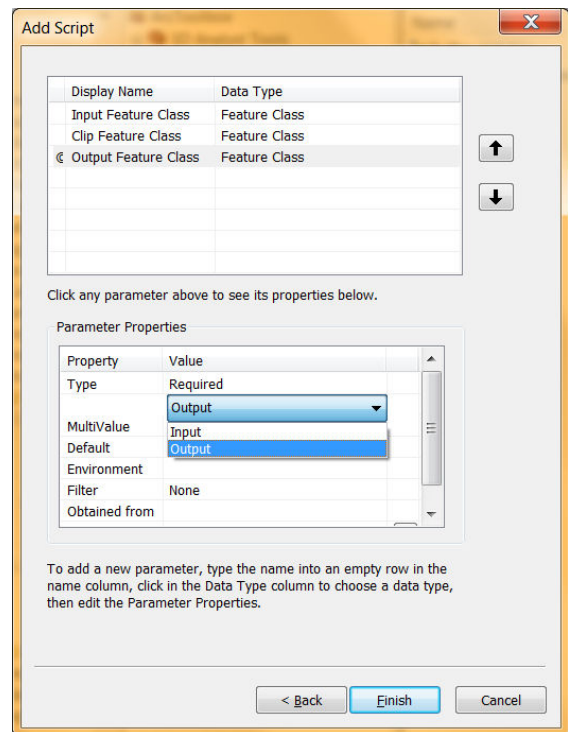
You have now entered the necessary parameters for the script tool.

Click Finish.

The new script tool is added to the ScriptTools toolbox.

You now have two different Buffer and Clip tools in the ScriptTools toolbox. One is the template model you used to create a script. The other is the custom script tool.

Close PythonWin.



## Step 6: Run the script tool

Now you will run the script tool.

Double-click the Buffer and then Clip script tool to open its dialog box.

The three parameters you set in the previous step are displayed in the dialog box, ready for user input.

**Tip:** Remember, you can drag existing feature classes from the appropriate geodatabase in the Catalog tree and drop them into the tool dialog box.

For Input Feature Class, drag and drop **Sinkholes** from **MarionCounty.gdb**.

For Clip Feature Class, drag and drop **StudyBoundary** from **StudyArea.gdb**.

For the third parameter, you must specify an output workspace and give the new output feature class a name.

Click the Browse button and navigate to **More\Florida\StudyArea.gdb**. Name the new feature class **SinkholeAreas**.

Click Save.

Click OK to run the Buffer and then Clip script tool.

When processing is complete, right-click StudyArea.gdb and choose Refresh.

One new feature class has been added: SinkholeAreas.

## Step 7: Delete intermediate data

The BufferTemp feature class was created as output from the Buffer tool and used as input for the Clip tool. Because you are only interested in the final output feature class, SinkholeAreas, you'll delete this intermediate data.

Right-click BufferTemp and choose Delete.

Click Yes to confirm the deletion.

Close ArcCatalog.

---

## Using scripts in models

Now that you have learned how to work with scripts and script tools, you are ready to learn how to add a script tool to a model. In this topic, you will return to the lynx project. The exercises will demonstrate three different ways you can use scripts to enhance the lynx habitat suitability models you worked with in the previous module, with each exercise increasing in complexity.

In the first exercise, you will learn how to add a script to a model. The script used in this exercise was written specifically for the lynx submodels and cannot be used in other models. The user of the models cannot change the values used by the script without modifying the code in the script itself.

In the second exercise, you will work with a script that can be used with any appropriate data. When you add the script to the model, you will create a model parameter from one of the script tool's parameters so that its value can be changed easily each time the model is run.

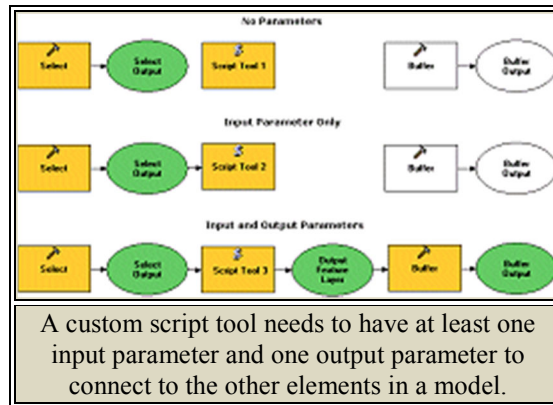
In the third exercise, you will add a script tool that creates conditional branching in a model. The script will check to see if a condition is true or false. If the condition is true, the model will run the processes in one branch. If the condition is false, the model will run the processes in the other branch.

## Script tool parameters for models

If you are using a custom script tool as an element in a model, you may need to consider some additional factors when you set up the tool parameters. For example, a script tool that will only be run as a standalone tool (such as the Clip All Feature Classes tool you created in the previous topic), may only have input parameters. However, when a tool is added to a model, it may also need to have an output parameter specified. Without an output parameter, you cannot connect the custom script tool to the next process in the model.

In the example below, no tool parameters have been set for Script Tool 1; therefore, it cannot be connected to any other element in the model. (Note the lack of connectors on both sides of the tool.) Script Tool 2 only has an input parameter. It can take input data from the model, but without an output parameter, it cannot connect to the next process.

Script Tool 3 has both an input parameter and an output parameter. It can take input data from the previous process and provide output data that can be used by the next process in the model.



Values for the input and output parameters can either be supplied by adjacent processes in the model, or by the user of the model, depending on how the script is written and the how the tool parameter properties are specified.

A script may be written for specific data and operations so that when it becomes a script tool, it will only work in the model for which it was written. Other scripts are written to be more generic so that they can be used in different models. You will get practice incorporating both types of scripts into models in the following exercises.

### *Exercise 3: Add a script to a model*

As you saw in the previous exercise, you can use a script to do the same things that a model does. In this exercise, you will learn how a script can be used in a model to replace repeated processes, thus simplifying the number of elements in the model. You will work with the lynx project data that you worked with in the previous two modules.

In Module 3, you used a common scale to rank a specific attribute field for each of the lynx habitat criteria (landforms, elevation, and prey habitat). To do this, you used the Select Layer By Attribute and Calculate Field tools.

This resulted in many elements being added to each of the submodels. In this exercise, you will replace the repeated elements in the Calculate Elevation Preference model with a script. This replacement has already been done for you in the other submodels.

Estimated time to complete: 20 minutes

#### **Step 1: Start ArcMap and open a map document**

Start ArcMap and open **LynxElevation.mxd** from your **More\LynxProject** folder.

Display ArcToolbox, if necessary.

This map document contains elevation data for the lynx habitat study area and the LynxTools\_More project toolbox.

#### **Step 2: Open the elevation model**

In the LynxTools\_More toolbox, expand the Working Toolset

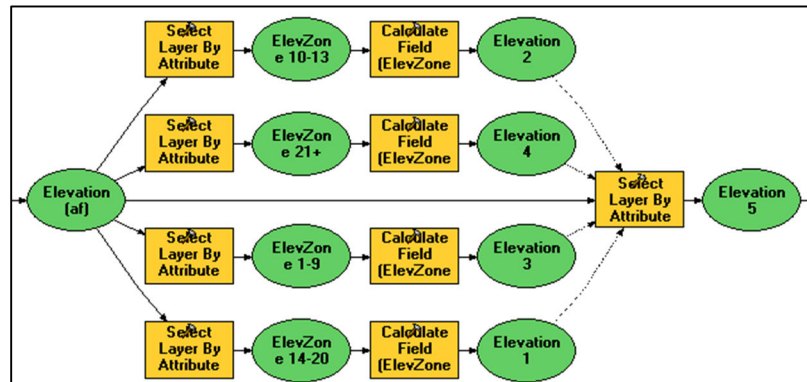


Open the Calculate Elevation Preference (2) model in edit mode.

### Step 3: Identify repeated elements

In this model, four elevation zones are selected and assigned a utility measure value from 1 to 9, with 9 offering the highest quality habitat, and 1 offering the lowest quality habitat.

Use the Pan and Zoom buttons to navigate the model and find the repeated elements that carry out these operations.



The repeated elements create selected sets and then assign a utility value for the ElevUtil field. The last Select Layer By Attribute element then clears the final selection set. (A precondition was set to force this tool to run after the others.)

In the remaining steps, you will simplify the model by replacing the Make Feature Layer, Add Field, Select Layer By Attribute, and Calculate Field processes with a custom script tool.

Before you replace the elements, it is important that you understand that one feature class will supply all the input data that will be used in the script.

Double-click the Elevation\_FC element.

This variable references the Elevation\_FC feature class stored in the LynxScratch geodatabase.

In the next few steps, you will learn what the implications are for using a custom script tool in a model when the results of processing are written back to the input data, instead of new output data being created.

Click Cancel to close the element dialog box.

### Step 4: Examine the script

A script has already been created to replace these elements, and the script has been added to a toolbox.

Open PythonWin.

In PythonWin, open **SelectandCalculate\_10\_1.py** from your **More\LynxProject\Scripts** folder.

Reposition or resize ModelBuilder, if necessary.

This script was written to work with all the lynx submodels. It uses the name of the input feature layer to determine which block of code to run.

The first block of code (beneath the script argument) checks for Elevation\_FC, the input feature class for the repeated processes in the Calculate Elevation Preference model. If Elevation\_FC is found, the block of indented code will run.

Read the comment to see what this block of code does.

Scroll down through the script.

Notice that for each block of code, the possible input feature class is provided, there are comments that tell you what the code does, and the set of processes is basically the same (assign utility values to a selected set of records).

In this script, there is only one script argument, the one for the input feature class. There is no script argument for the output feature class because, as you saw in the previous step, the results of the processing are written back to the input feature class (Elevation\_FC).

Close PythonWin.

### Step 5: Review the script tool parameters

Now that you know what the script looks like, you will take a look at how its tool parameters have been set up.

In ArcToolbox, right-click the Select and Calculate tool and choose Properties. Click the Parameters tab.

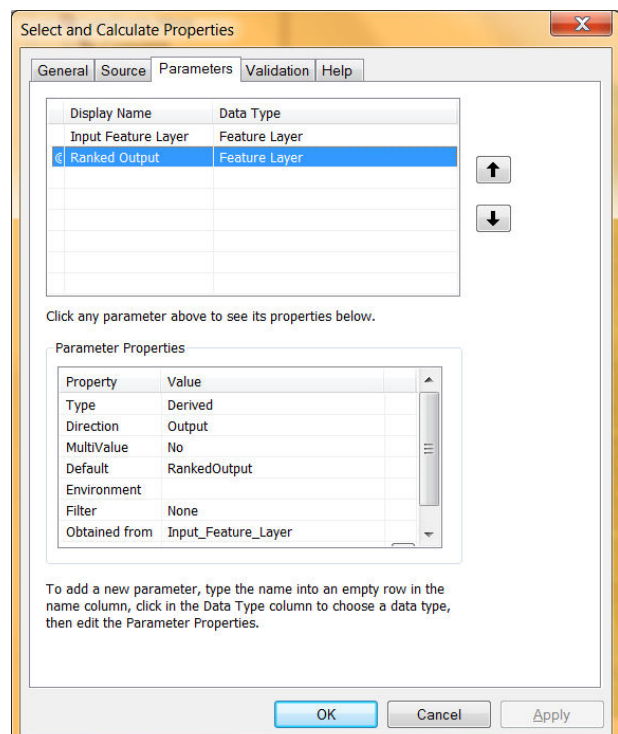
From your review of the SelectandCalculate\_10\_1 script, you know that only one script argument was created in the script, for the input feature layer. That script argument corresponds to the Input Feature Layer parameter you see in the dialog box.

A second parameter has also been set up for the tool, although there is no corresponding script argument for this parameter. This second parameter was added so that when the tool is added to the model, it can be connected to the next process.

Click RankedOutput in the cell below Display Name to see the parameter properties.

Three of the property settings are different from those you have worked with so far.

Notice that this is a derived parameter and a dependency has been set on the input feature layer.



The parameter type is specified as derived because it is automatically generated as a result of the geoprocessing operations—it is not a user-specified parameter.

The dependency property specifies that the parameter is dependent on the value specified for another parameter (in this case, the input feature layer). These two properties work together to set the output parameter's value to automatically be the same as the input parameter's value in the model. (Remember, in this case, the results of the processing are written back to the input feature layer.)


The Default property provides a default value for the parameter (in this case, a temporary name for the output element).

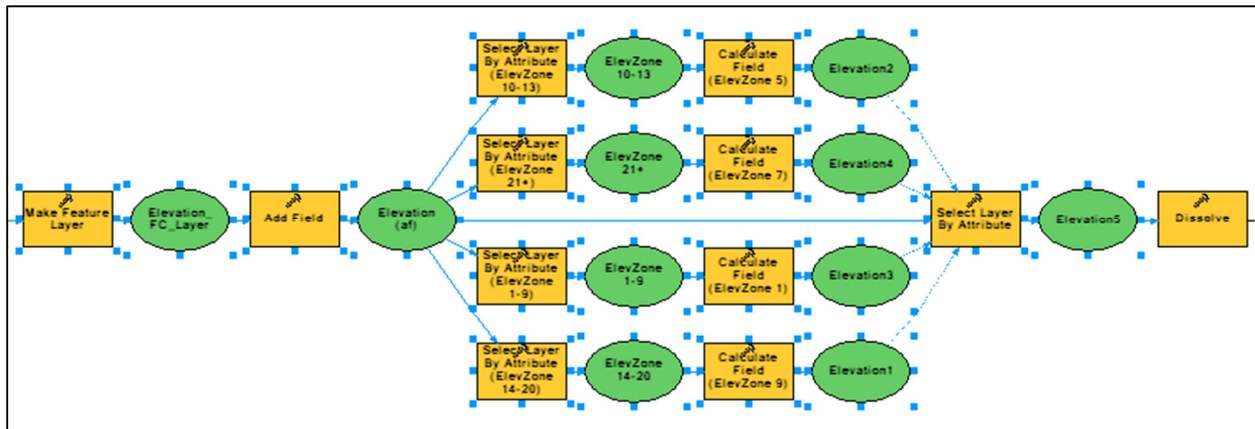
Close the dialog box.

You are now ready to add the Select and Calculate tool to the model.

### Step 6: Replace model elements with the script

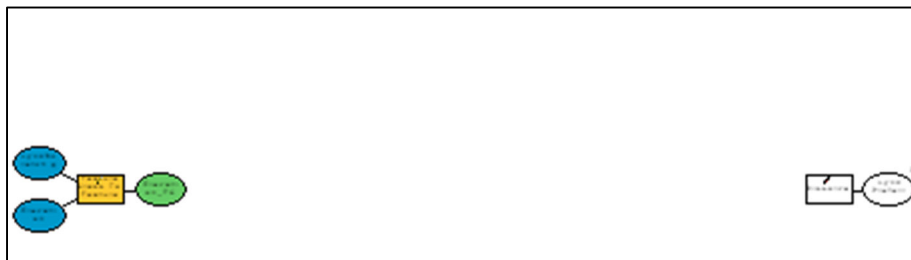
Before you add the script tool to the model, you need to delete the elements it will replace.

In ModelBuilder, click the Select tool  and drag a box around all the elements between the Make Feature Layer tool and the Elevation5 element (Make Feature Layer and Elevation5 should also be selected).



**Tip:** If you accidentally select an extra element, hold down your Ctrl key and click the element to unselect it (use the Shift key to add an element to the selection).

Press the Delete key, then click Yes to confirm the deletion.

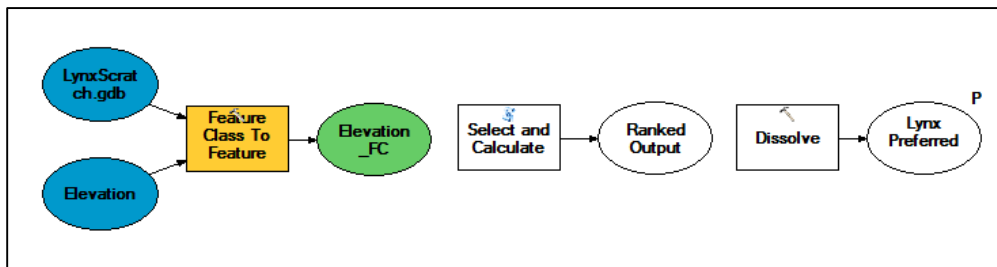


From ArcToolbox, add the Select and Calculate tool to the model to the right of the Elevation\_FC element.


There is now a large gap between the Ranked Output element and the next element in the model, the Dissolve tool.

Drag the Select and Calculate Script tool from the Working Toolset into the model.

Use the Select tool to select the Dissolve tool and its output element (use the Shift key to select both elements), then drag both elements over to the right of the Ranked Output element.



Next, you will connect the script tool to the other elements in the model.

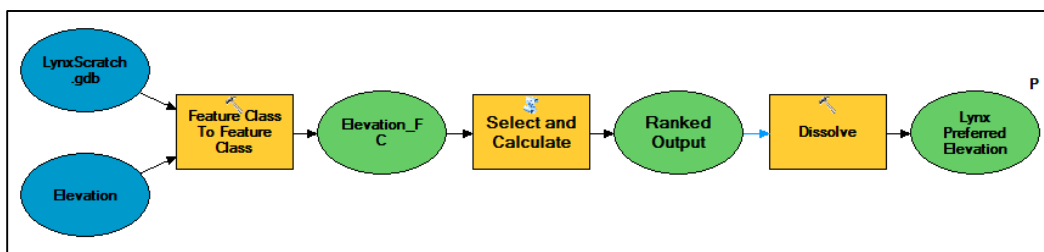
Click the Add Connection button  and draw a connection between the Elevation\_FC element and the Select and Calculate element.

When you connect Elevation\_FC to the Select and Calculate script tool, the first parameter set for the tool enables it to be connected with an appropriate input data element. (Remember, earlier you verified that the Elevation\_FC variable references Elevation\_FC.)

The Ranked Output element was created by the second parameter set for the tool but has been renamed automatically to reflect the input to the script tool. Rename it back to **Ranked Output**.

Draw a connection between the Ranked Output element and the Dissolve tool element.

Click the Auto Layout button .



The two parameters set for the custom script tool allowed you to connect the tool to the rest of the model, and the updated model is ready to run.

Step 7: Run the model

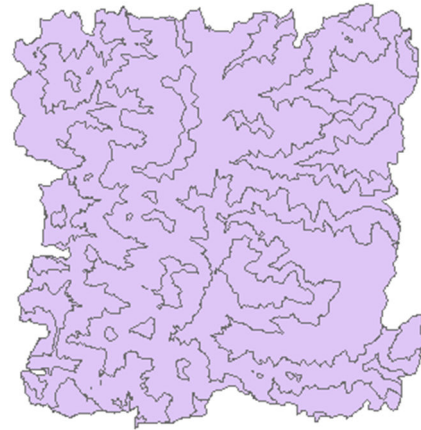
Click the Full Extent button .

From the Model menu, choose Run Entire Model.

When the model has executed, close the ModelBuilder window and save the changes to the model.

The Lynx Preferred Elevation layer has been added to the map display.

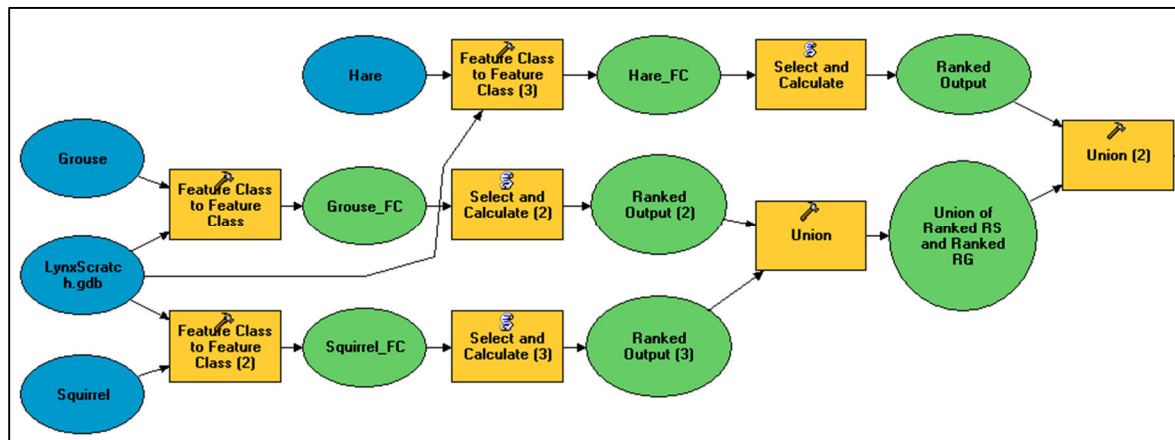
**Optional:** To visualize the results of the model, symbolize the Lynx Preferred Elevation layer to show quantities based on the ElevUtil field. Choose a graduated color ramp with nine classes. (In this case, there are actually only four classes represented in the model, but ArcMap makes the adjustment automatically.)



## Step 8: Review the other two submodels

The replacement has already been done for you in the other two submodels.

Expand the Complete toolset, then open the Calculate Prey Distribution model in edit mode.

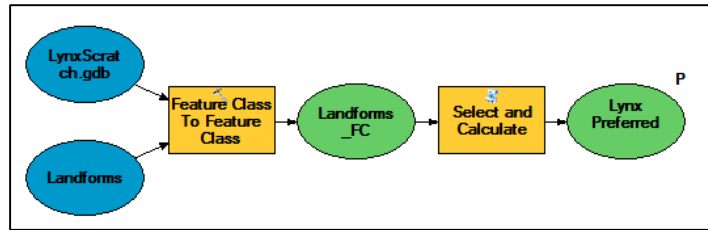


In this model, the Make Feature Layer, Add Field, Select Layer By Attribute and Calculate Field tools have been replaced with the Select and Calculate custom script tool for each lynx prey species.

Open the dialog box for each of the three Ranked Output elements to see which feature layer is being referenced, and therefore, where the utility measure values are being added.

Close the Calculate Prey Distribution model.

Open the Calculate Landform Preference model in edit mode.



Notice that the output from the Select and Calculate script tool is named Lynx Preferred Landforms instead of Ranked Output, which is the default output name for the script tool.

The name of the element was changed after the tool was added to the model.

In the Calculate Landform Preference model, no processes were required after the utility measure values were calculated. Therefore, the output element from the Select and Calculate script tool was the last element in the model. As you may recall from the previous module, the final element in each of these three models was marked as a model parameter so that each one could function as a submodel to the Final Lynx Habitat model.

Changing the name of the final element in the Landform model to match the original element name allows the submodel to be recognized by the Final Lynx Habitat model.

You will work with the Final Lynx Habitat model in the next exercise.

Close the Calculate Landform Preference model.

### Step 9: Save the map document

If you want, save the map document as **LynxElevation2.mxd** in your **More** folder.

If you are continuing on to the next exercise, leave ArcMap open. Otherwise, exit ArcMap.

-----

## Exercise 4: Use Python expressions in a tool

In the previous exercise, you replaced repeated elements in the Calculate Elevation Preference model with a script. Did you know that many geoprocessing tools allow you to include code within a code block parameter?

In this exercise, you will replace the repeated elements in the Calculate Elevation Preference model with the Calculate Field tool using a Python code block.

Estimated time to complete: 15 Minutes

### Step 1: Start ArcMap and open a map document

Start ArcMap if necessary and open **LynxElevation.mxd** from your **More\LynxProject** folder.

If necessary, display ArcToolbox.

This map document contains elevation data for the lynx habitat study area and the LynxTools\_More project toolbox.


## Step 2: Open the elevation model

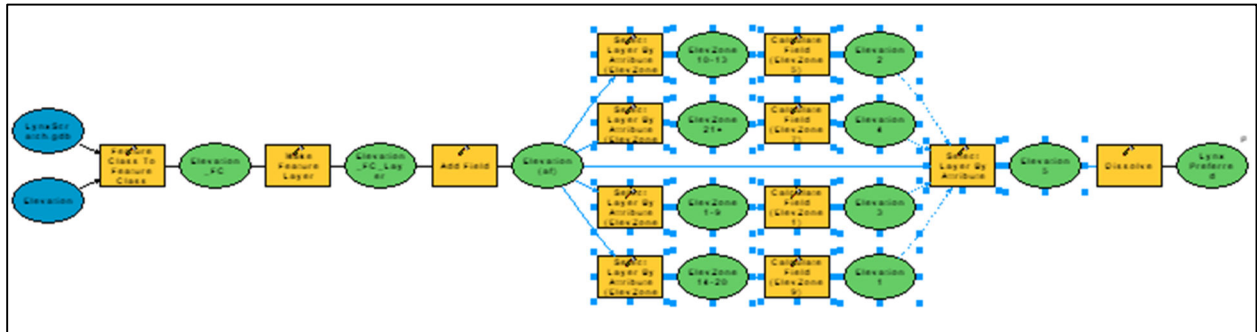
In ArcToolbox, expand the LynxTools\_More toolbox, then expand the Working toolset.

Open the Reclass model in edit mode.

## Step 3: Replace model elements with a tool

Before adding the Calculate Field tool to the model, you need to delete the elements it will replace.

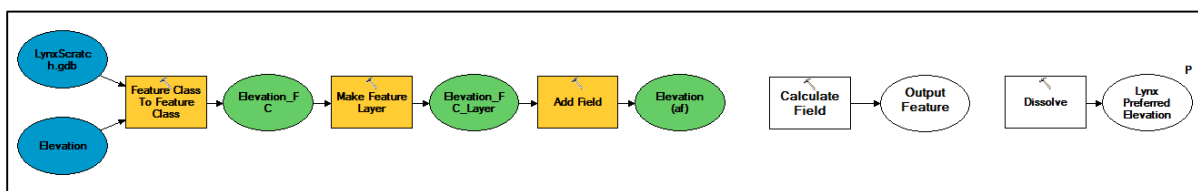
Click the Select tool  and draw a box that encompasses all the Select Layer By Attribute tools and the Elevation5 element.




Press the Delete key, then click Yes to confirm the deletion.

From ArcToolbox, add the Calculate Field tool to the right of the Elevation (af) element.

Use the Select tool to select the Dissolve tool and its output element and drag both elements to the right of the Output Feature Class element.



Next, you will connect the script tool to the other elements in the model.

Click the Add Connection button  and draw a connection between the Elevation (af) element and the Calculate Field element.

Connect the Output Feature Class element to the Dissolve tool element.

Click the Auto Layout button .

## Step 4: Set parameters for the tool

In this step, you'll reclassify a range of values within the Calculate Field tool using a Python code block.



Double-click Calculate Field to specify its parameters.

For Input Table, Elevation (af) should already be selected.

For Field Name, choose ElevUtil.

For Expression, type **reclass(!elevzone!)**

This is the Python calculation expression. In Python, field names are always enclosed in exclamation marks.

For Expression Type, choose PYTHON.

Using Windows Explorer, navigate to your **\More\LynxProject** folder. Open the **PythonCodeBlock.txt** file in Notepad.

Select and copy all the text, then close the text file and Windows Explorer.

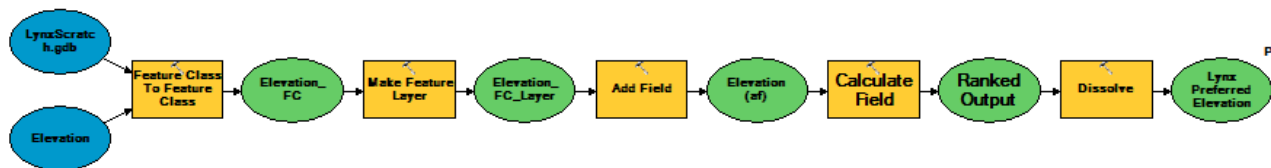
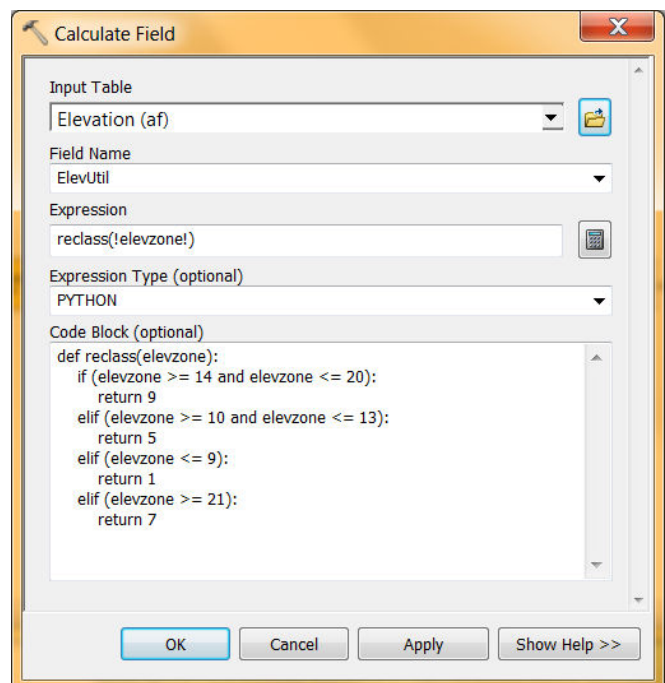
In the Calculate Field tool dialog box, paste the text into the Code Block box.

**Note:** Make sure to keep the indentation exactly as shown, or else the script may not work.


In Python, functions are defined using the "def" keyword followed by the name of the function and the function's input parameters. In this case, the function is called "reclass" and it has one parameter named "elevzone." Values are returned from a function using the *return* keyword.

Click OK.

Rename the output element **Ranked Output**.



## Step 5: Run the model

Click in empty space to unselect any selected element, then click the Full Extent button .

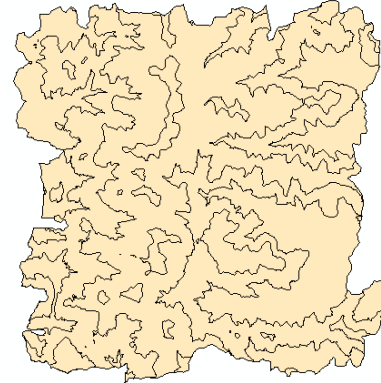
From the Model menu, choose Run Entire Model.



When the model has executed, close the ModelBuilder window and save the changes to the model.

The Lynx Preferred Elevation layer has been added to the map display.

**Optional:** To visualize the results of the model, symbolize the Lynx Preferred Elevation layer to show quantities based on the ElevUtil field. Choose a graduated color ramp with nine classes. (In this case, there are actually only four classes represented in the model, but ArcMap makes the adjustment automatically.)



### Step 6: Save the map document

If you want, save the map document as **LynxElevation3.mxd** in your **More** folder.

If you are continuing on to the next exercise, leave ArcMap open. Otherwise, exit ArcMap.

-----

### Exercise 5: Expose script tool parameters as model parameters

In the previous module, you ranked lynx habitat criteria using a scale of 1 to 9. However, suppose some of the project stakeholders need the habitat criteria ranked on a scale of 1 to 3. Without a script, you would have to rebuild a good portion of your model.

In this exercise, you will use a custom script tool that will reclassify the lynx criteria, then you will expose the appropriate script tool parameter as a model parameter, allowing anyone who runs the model to easily specify different reclassification values—no rebuilding required.

Estimated time to complete: 30 minutes

### Step 1: Open a map document

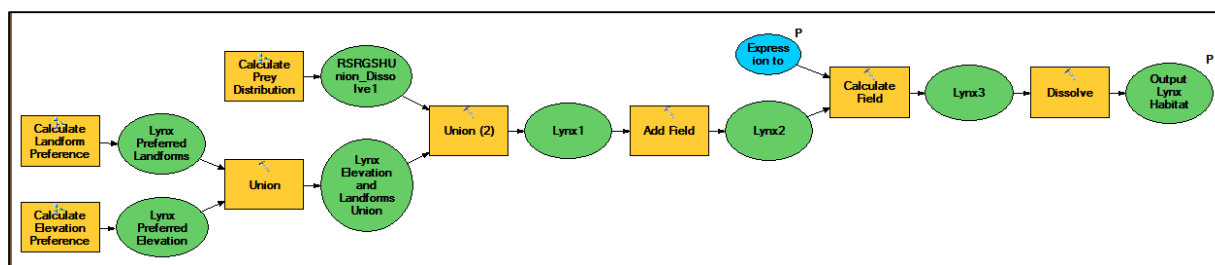
If necessary, start ArcMap and open **LynxReclass.mxd** from your **More\LynxProject** folder.

The map shows one of the potential outcomes of your weighted suitability model. To create this data, you assigned weights of 60 percent to elevation, 15 percent to landforms, and 25 percent to prey. The values were symbolized based on the HabitatUtil field using nine classes, with dark green representing the most suitable habitat.

### Step 2: Review the model

In the LynxTools\_More toolbox, expand the Working toolset.

Open the Final Lynx Habitat model in edit mode.




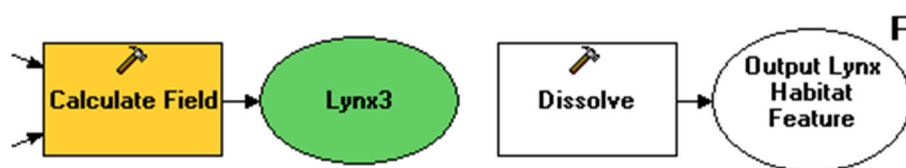
This model is identical to the one you created in the previous module.

In this exercise, you will add elements to the model to reclassify values in the HabitatUtil field from values ranging from 1 to 9 to new values of 1 to 3, with 3 being the most suitable lynx habitat.

To reclassify the habitat values, you need to add a field to store the new values and add a custom script tool to perform the reclassification. These elements will be added to the model before the dissolve is performed.

Zoom in if necessary.

Using the Select tool , select the connector between the Lynx3 and Dissolve elements, then delete it.



In an upcoming step, you will insert the new elements here.

Minimize the model.

### Step 3: Determine the parameters for the script tool

The script that reclassifies the values has already been written, but it has not been added to a toolbox. You will review the script in PythonWin to understand how it works and to determine how to set up the script tool parameters.

Open PythonWin.

In PythonWin, open **ReclassVector\_10\_1.py** from your **More\LynxProject\Scripts** folder.

Read the script comments (green text) to get an understanding of how the script works.

**Note:** The code in this script is written more efficiently than in the last script you worked with. If you are not familiar with Python and the script did not contain comments, it might be difficult to understand what the script is designed to do and how to set the parameters. You can see how important it is that scripts be adequately documented.

Notice that there are no hard-coded paths in this script. All inputs and outputs to the script are handled by arguments. This way the script can be used with any appropriate data, rather than being limited to use with a single dataset.

Each argument requires that a tool parameter be created when this script becomes a script tool. Because all of these are input parameters, a fifth parameter will be required to create an output element for the model.

**Note:** This script is similar to the `SelectandCalculate_10_1` script you used in the last exercise in that the results of the processing from the script are written back to the input data.

Minimize PythonWin so that you can refer back to the script if necessary in the next step.

#### Step 4: Add the script to a toolbox

In ArcToolbox, right-click the Working toolset, click Add, then choose Script.

In the Add Script wizard, for Name, enter **ReclassValues**, and for Label, enter **Reclass Values**.

For Description enter: **This script tool reclassifies values in an attribute field based on user-specified ranges.**

Check the box next to "Store relative path names."

Click Next.

Browse to your **LynxProject\Scripts** folder and open **ReclassVector\_10\_1.py**.

Click Next.

Now you will set up five parameters: the four parameters that correspond to the script arguments plus the output parameter required to connect to the next element in the model.

Enter the information for each parameter using the information below. Under Parameter Properties, use the default values unless otherwise specified.

##### First parameter

Display Name: **Input Table**

Data Type: Table View

**Note:** A [table view](#) is similar to a feature layer, but is more inclusive. The table view data type enables the parameter to accept any of the following: Table View, Table, Feature Layer, Feature Class, or Layer file.

##### Second parameter

Display Name: **Original Values Field**

Data Type: Field

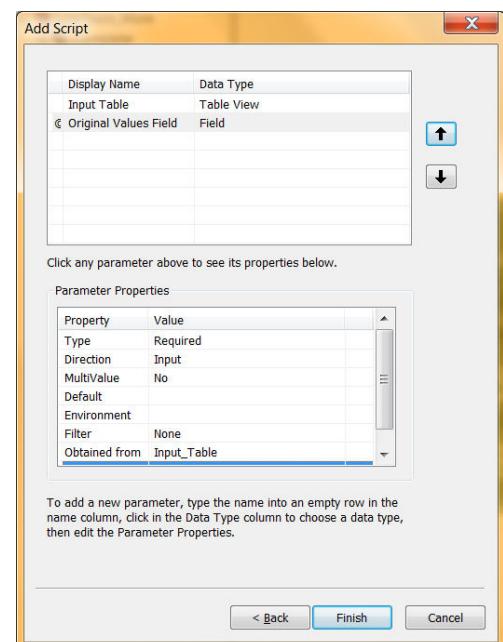
Obtained from: `Input_Table`

##### Third parameter

Display Name: **Break Points for Ranges**

Data Type: Double

MultiValue: Yes



**Note:** The multivalue property allows the parameter to accept more than one value.

#### Fourth parameter

Display Name: **Reclassified Values Field**

Data Type: Field

Obtained from: Input\_Table

#### Fifth parameter

Display Name: **Reclassified Output**

Data Type: TableView

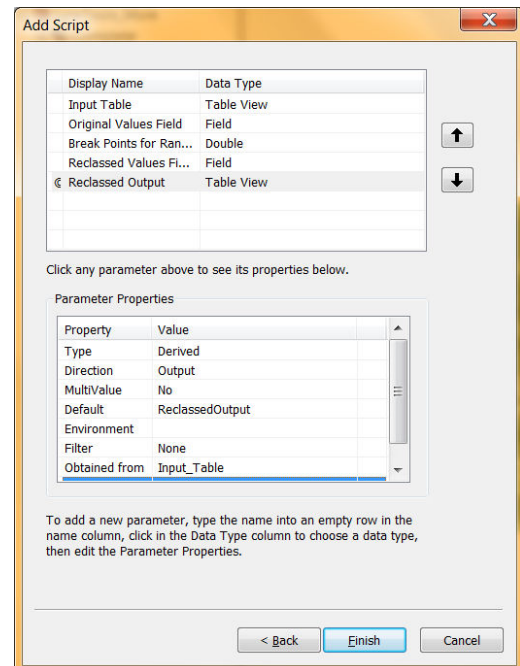
Type: Derived

Direction: Output

Default: **ReclassifiedOutput**

Obtained from: Input\_Table

**Reminder:** As you learned in the previous exercise, the derived and dependency properties work together so that the value entered for the input parameter is automatically used for the output parameter. The default property provides a default name for the output parameter.



Click Finish.

The Reclass Values tool is added to the Working toolset and is ready to be added to the model.

This script tool can also be run as a standalone tool. When it is run outside of a model, the output parameter is not needed and is ignored.

Close PythonWin.

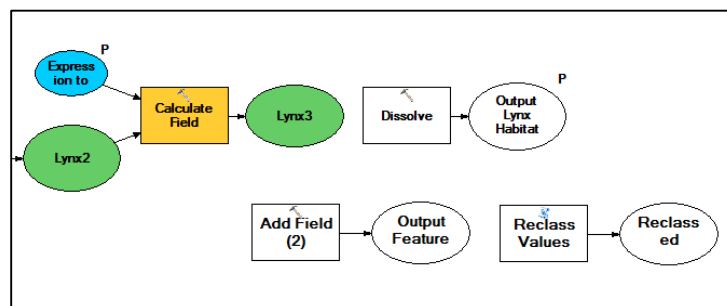
### Step 5: Add tools to the model

Restore the Final Lynx Habitat model.

The Lynx3 element in the model will provide the input to your new script tool. The data referenced by Lynx3 contains the HabitatUtil field with the values to be reclassified. However, it does not include a field to store the reclassified values. You will add a new field.

Drag the Add Field tool into the model below the Lynx3 element.

Drag the new Reclass Values tool into the model to the right of the Add Field [2] process.



## Step 6: Set parameters for the tools

In the model, double-click Add Field [2] to specify its parameters.

For Input Table, choose Lynx3.

For Field Name, enter **UtilReclass**.

For Field Type, choose SHORT.

Click OK.

Rename the output element to **Lynx4**.

Now you will specify the parameter values for the Reclass Values tool.

Open the Reclass Values tool dialog box.

Only the parameters that correspond to the four script arguments are displayed in the dialog box. The fifth parameter is not displayed. You specified the fifth parameter as a derived value; therefore, its value cannot be entered by the user.


Except for the Break Points for Ranges parameter, the parameter values will be supplied by the model. The break points will be specified by the user when the model is run.

For Input Table, choose Lynx4.

For Original Values Field, choose HabitatUtil.

Although you want the user to choose the break points, you will need to specify at least one default value.

Type **5** and click the Add

button . (This would result in values from 1 to 5 being reclassified into class 1, and values from 6 to 9 being reclassified into class 2.)

For Reclassed Values Field, choose UtilReclass.

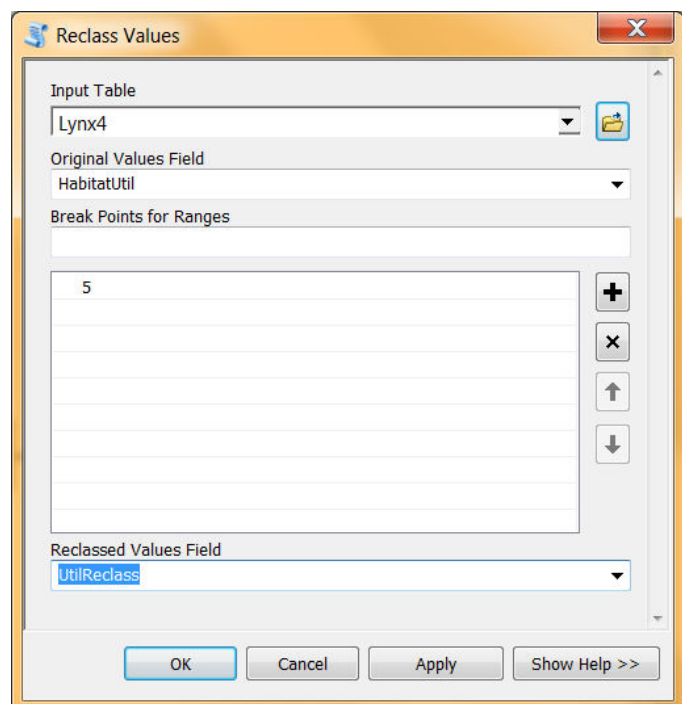
Click OK.

In the previous module, you learned how to create a variable from a tool parameter and then expose the variable as a model parameter. When the model is run from its dialog box, the parameter value can be easily changed.

You will do this for the Break Points for Ranges parameter.

In the model, right-click Reclass Values and choose Make Variable > From Parameter > Break Points for Ranges.

The variable is added to your model as a new element.



Make the new element a model parameter.

Rename the Reclass Values output element to **Habitat Reclass**.

Finally, you will change parameter values for the Dissolve tool.

Open the Dissolve tool dialog box.


For Input Features, choose Habitat Reclass.

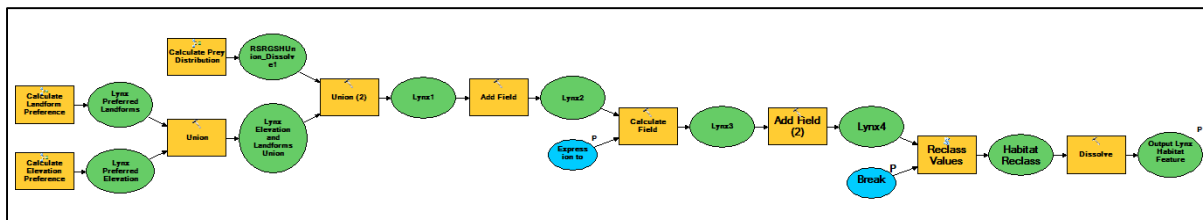
For Dissolve Field(s), uncheck HabitatUtil, then check UtilReclass.

If you see a yellow warning icon next to the Output Feature Class text box, it means that the output feature class name exists and will be overwritten when the tool runs. This is OK because you can change the name when the model is run from a dialog box.

Click OK.

Click the Auto Layout button  to organize the model.

Click the Full Extent button  to see all the processes in the model.



Save the model and close ModelBuilder.

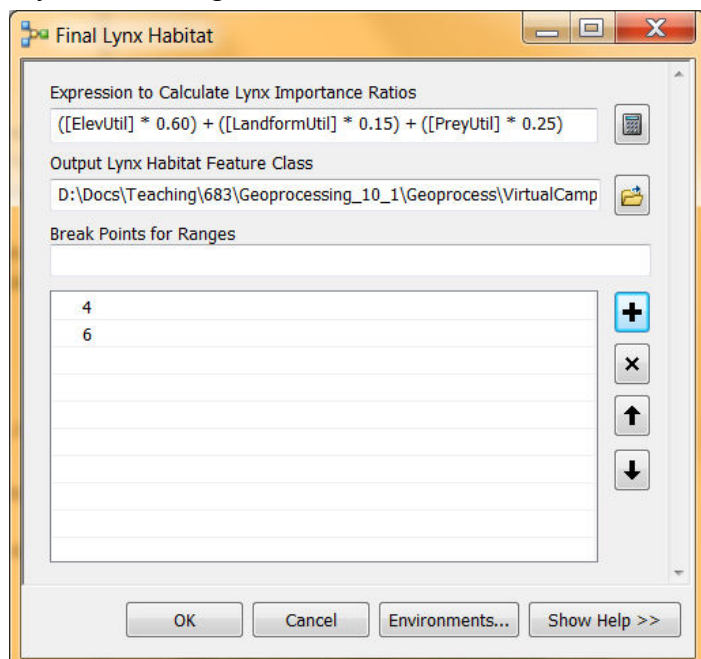
## Step 7: Run the model from its dialog box

In ArcToolbox, double-click the Final Lynx Habitat model to open its dialog box.

You'll leave the expression as is, but you will change the name and location of the feature class so that you do not overwrite the original one.

Modify the Output Lynx Habitat Feature Class parameter so that the output is stored in your **LynxHabitat.gdb** geodatabase with the name **LynxE60L15P25\_r3**.

The value that you filled out for the Break Points for Ranges parameter is shown as a default value. You want to reclassify the values into three classes: 1 to 4, 5 to 6, and 7 to 9. Therefore, you will delete the default value and enter two new break point values.





Click 5 to select it, then click the Remove button .

In the text box below Break Points for Ranges, enter 4 and click the Add button .

Next, enter 6 in the text box and click the Add button.

Specifying two break points results in the creation of three classes.

Click OK to run the model.

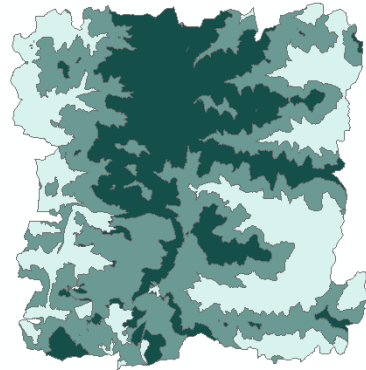
### Step 8: Symbolize the new layer

When the model has executed, the LynxE60L15P25\_r3 layer is added to the map.

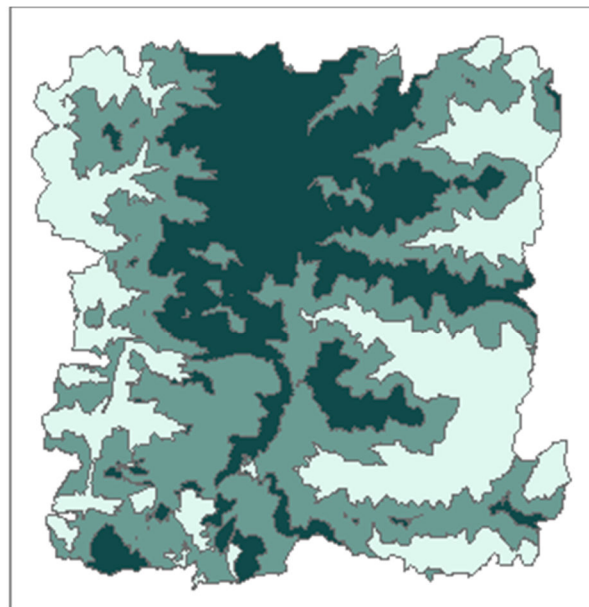
Symbolize the layer to show quantities (graduated colors) based on the UtilReclass field. Use three classes and the Blue-Green Light to Dark color ramp.

Compare the differences in habitat suitability between the two map layers.

Notice how using three classes instead of nine changes the results of your analysis.



**LynxE60L15P25**



**LynxE60L15P25\_r3**

If you did not want to use the Reclass Values script, you could have achieved similar results by symbolizing the LynxE60L15P25 layer using three classes, but that would not have resulted in a new feature class. More importantly, by enhancing the Final Lynx Habitat model with the Reclass Values script, you have built in additional control for generating different types of outputs.

Now, you can not only change the weight values in the expression and the name of the output feature class, but also the utility value classification—all from a simple dialog box.

### Step 9: Save the map document

If you want, save your map document as **LynxReclass2.mxd** in your **More** folder.

Exit ArcMap.

-----

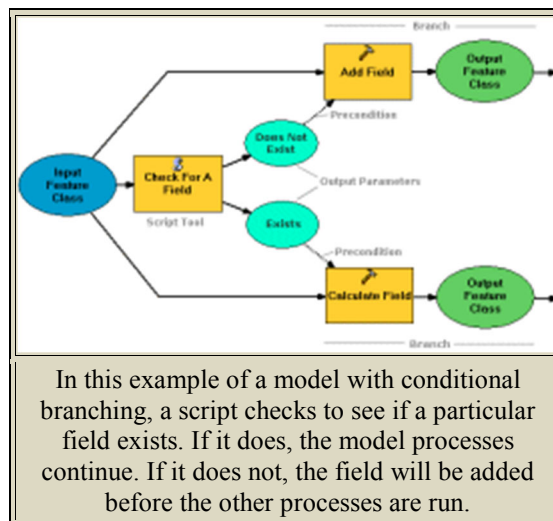
### Decision making in a model

When programmers write scripts, they can use statements native to the scripting language to test a condition. If one condition is true, then the appropriate set of code will run, and if the condition is false (or another condition is true) then another set of code will run. You can incorporate this same functionality into your models by using a script tool to enable conditional branching.

For example, you may want your model to work with shapefiles, coverages, or geodatabase feature classes. Your script would check to see which type of data is being used as the input and report this information as output for use by the model.

- If the input is a shapefile, the branch of the model that begins with the Feature Class to Feature Class process would run.
- If the input is a coverage, the branch of the model that starts with the Select Data process would run to extract a feature class.
- If the input is a geodatabase feature class, the branch beginning with the Select process would run.

Using conditions allows you to add flexibility to your model.





## Exercise 6: Conditional branching with scripts

In this exercise, you will use a custom script tool to incorporate conditional processing (branching) into a model.

The steps for creating branching in a model are:

1. Create or obtain a script that sets up the condition or conditions to be tested.
2. Add the script to a toolbox, specifying output parameters that will tell the model how to branch.
3. Add the script tool to the model, and apply precondition properties to the first tool in each branch to control flow.

To see how you might use a script tool to test a condition in a model, you will work with an excerpt from the Final Lynx Habitat model. The script and script tool have already been created.

Estimated time to complete: 20 minutes

### Step 1: Start ArcCatalog

Start ArcCatalog.

In the Catalog tree, navigate to your **More\LynxProject\Tools** folder.

Expand the Tools folder, the LynxTools\_More toolbox, and the Working toolset.

This toolset contains a script tool named Check For A Field, which checks whether a field in a feature attribute table exists. The script sets up a condition that you can use to control the flow of processing in a model.

### Step 2: Examine the script

In this step, you will examine the script and the conditional code it contains.

Open PythonWin.

In PythonWin open **CheckForField.py** from your **More\LynxProject\Scripts** folder.

Read the script comments to gain an understanding of what the script does.

Notice that there is a script argument for the input feature class.

At the bottom of the script, you will find arguments that look a little different. They create two output parameters and assign a value to each parameter.

```
-if bFieldExists == True:
    arcpy.AddMessage("The field exists")
    arcpy.SetParameterAsText(1, "True") ←
    arcpy.SetParameterAsText(2, "False") ←

# If the field doesn't exist, the value for the second derived output
# parameter is set to true.
-else:
    arcpy.AddMessage("The field does not exist")
    arcpy.SetParameterAsText(1, "False") ←
    arcpy.SetParameterAsText(2, "True") ←
```

The parameter values are assigned as follows:

- If the field exists in the input feature class, the first parameter is assigned a value of True, and the second parameter is assigned a value of False.
- If the field does not exist in the input feature class, the first parameter is assigned a value of False, and the second parameter is assigned a value of True.

When this script tool is added to the model, the parameter that receives a value of true runs its branch of the model.

This script has already been added to a toolbox and the script tool created.

Next, you will take a look at the script tool.

Close PythonWin.

### Step 3: Examine the script tool

In the Catalog tree, right-click the Check For A Field tool again, but this time choose Properties.

Click the Parameters tab.

You see the three parameters that correspond to the script arguments. The first parameter references the InputFC = `sys.argv[1]` script argument. The other two parameters reference the Boolean type arguments of "The field exists" and "The field does not exist."

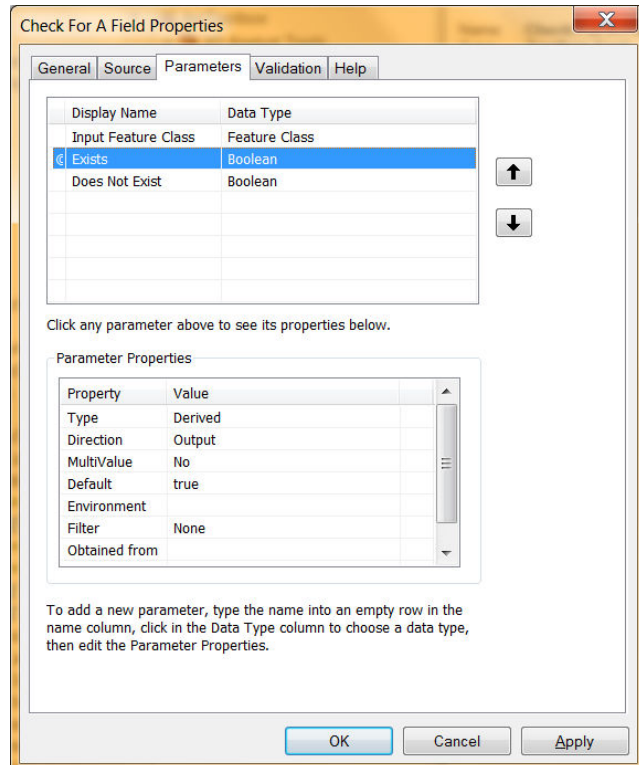
Click the Exists parameter to view its parameter properties.

Notice that this parameter is derived. This means that its value cannot be set by the user. A default value has been set for the parameter. While the values for the Boolean parameters are set by the script when it runs, you must also provide a value here in the parameter properties of true or false (either will work) so that the script tool can be connected to the other elements in the model.

Click the Does Not Exist parameter.

The properties for this parameter are the same as for the Exists parameter.

Click Cancel to close the script tool dialog box.



Next, you will add this script tool to an empty model.

#### Step 4: Add the script tool to a model

Create a new model in the Working toolset.

From the Model menu, choose Model Properties. Click the General tab.

For Name, enter **FieldCheck**.

For Label, enter **Check for field**.

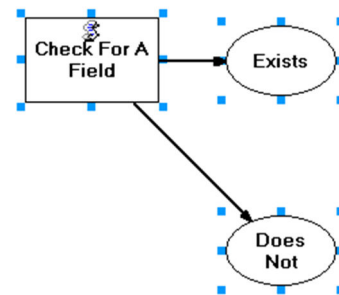
Check the box next to "Store relative path names."

Click OK.

Drag the Check For A Field script tool into the new model.

You see the script tool and its two derived parameters. What you do not see is the input feature class variable. You will set up the input feature class variable in the next step.

Save the model.



#### Step 5: Create an input feature class variable

In this step, you will add a feature class variable to the model and make it a model parameter. You will also add the Make Feature layer tool to the model and connect all the elements.

Right-click in an empty area of the model and choose Create Variable.

In the Create Variable dialog box, scroll down the list of variables and choose Feature Class.

Click OK.

Make the Feature Class variable a model parameter.

Drag the Make Feature Layer tool into the model, then open the Make Feature Layer tool dialog box.

For Input Features, choose **Feature Class**.

For Output Layer, enter **FC\_Layer**.

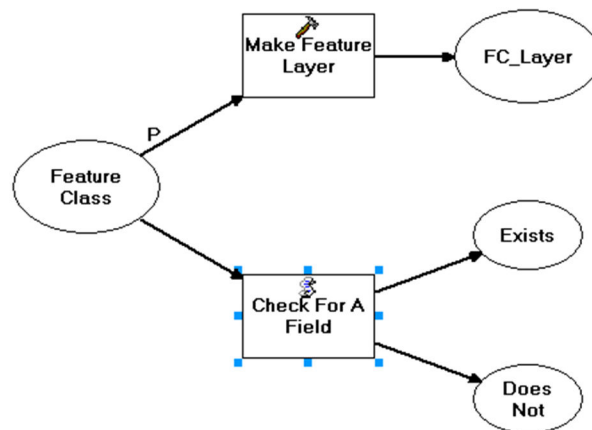
Click OK.

Now open the Check For A Field tool dialog box.

For Input Feature Class, choose **Feature Class**.

Click OK.

Click the Auto Layout button .



Currently, this model might seem a little odd because the feature layer (FC\_Layer) and the Check For A Field script tool are not connected. In the next step, you will start tying them together by creating the first branch in the model.

#### Step 6: Create the first branch

The field name that the Check For A Field script tool is checking for, HabitatUtil, is *hard-coded* in the CheckForField.py script. For this exercise, if the field exists in the input feature class, you will instruct ArcGIS to calculate all the values in the field to be 1.

Add the Calculate Field tool to the model, then open its tool dialog box.

For Input Table, choose FC\_Layer.

For Field Name, enter **HabitatUtil**.

For Expression, enter **1**.

Click OK.

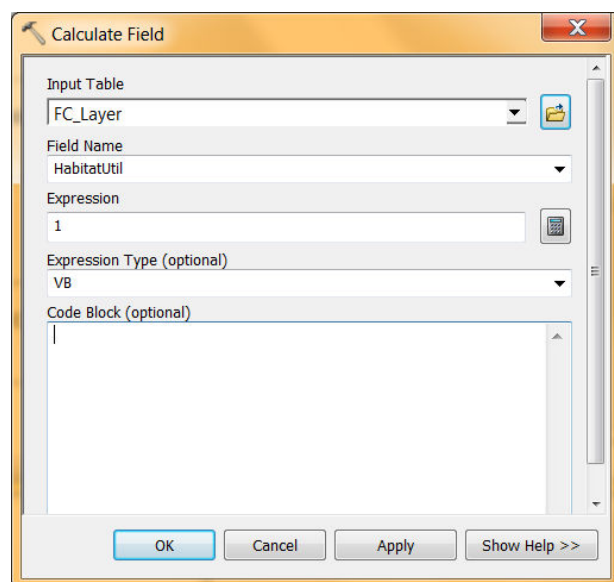
Rename the output element for the Calculate Field tool to **Field Calc'd**.


Right-click the Calculate Field element and choose Properties.

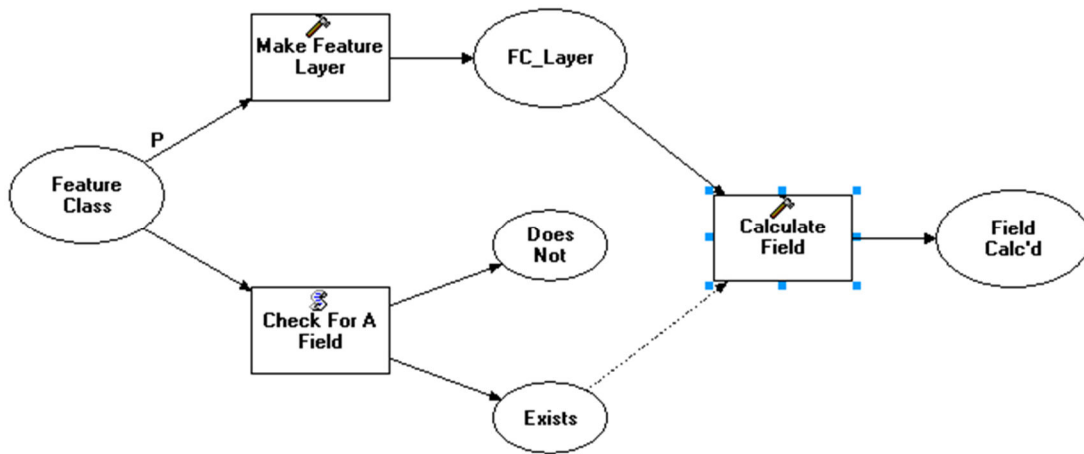
In the Calculate Field Properties dialog box, click the Preconditions tab and check the box next to Exists.

As you learned earlier in this course, preconditions control the sequence of processing in your model. In this case, the Check for A Field process must complete before the Calculate Field tool can run.

Click OK.



Click the Auto Layout button .



Now, if the HabitatUtil field exists in the input feature class, its values will be calculated based on the expression.

In the next step, you will build the branch of the model that executes when the HabitatUtil field does not exist.

### Step 7: Create the second branch

In this exercise, if the HabitatUtil field does not exist in the input feature class, you will instruct ArcGIS to create it.

Drag the Add Field tool into the model and open its tool dialog box.

Specify the following parameters:

- Input Table: FC\_Layer
- Field Name: **HabitatUtil**
- Field Type: SHORT

Click OK.

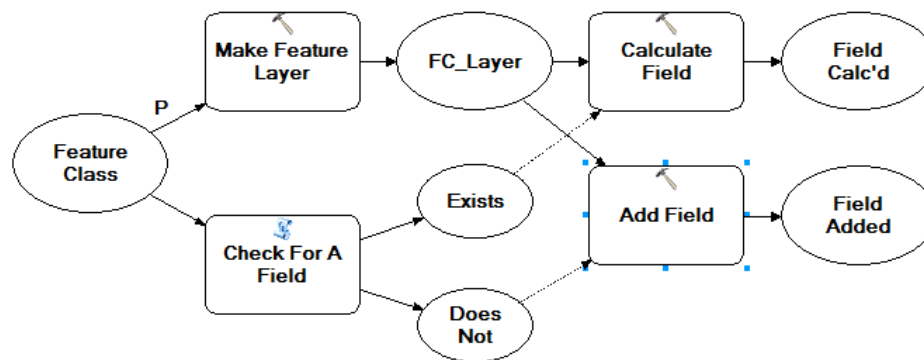
Rename the output element of the Add Field tool to **Field Added**.

Open the properties for the Add Field element.

In the Preconditions tab, check the box next to Does Not Exist.

Click OK.

Click the Auto Layout button .



This model is an example of how you can use scripts to control the flow of processing. If the designated field exists in the input feature class, then the values for all the records in that field are calculated to 1. If the field does not exist, then it is created. The model presents two different results based on the testing of a condition.

Close ModelBuilder. Click Yes to save changes.

Now you need to test the model. You will run it from its dialog box.

### Step 8: Test the model

The new model, Check for field, has been saved to the Working toolset in your LynxTools\_more toolbox. Remember, you can run a model from its dialog box just like any other geoprocessing tool.

Double-click the Check for field model to open its dialog box.

You only need to supply one parameter value, the input feature class.

In the Check for field dialog box, for Feature Class, browse to **LynxProject\Data\LynxHabitat.gdb** and add the **LynxPreferred** feature class.

The HabitatUtil field does exist in the LynxPreferred feature class.

Click OK to run the model.

After the model executes, in the Catalog tree, navigate to the LynxHabitat geodatabase, expand it and click the LynxPreferred feature class.

Preview the LynxPreferred attribute table.

Scroll over to the HabitatUtil field and notice that all the values in this column are 1.

FID_Elevation_Dissolve	ElevUtil	HabitatUtil	Shape_Length	Shape_Area
2	5	1	5046.50882317861	0.256554210325703
4	9	1	6173.46870210745	0.439278362318873
4	9	1	7928.86308104073	0.433135418875445
2	5	1	1429.54908832491	0.108027924616181
4	9	1	273.073718588527	1.12953915992625E-02
3	7	1	2501.28390607923	0.104877753794426
4	9	1	75.6488931860108	0.013705036891281
1	1	1	1511.90191880224	4.93935117186714E-02
1	1	1	2999.54587339163	0.223270270595094
2	5	1	5229.54276082171	0.253083929640312

While you are previewing attribute tables, click the LynxUnion feature class and examine its fields.

The LynxUnion attribute table does not have a HabitatUtil field.

Now you will test the model for when the condition (HabitatUtil field does not exist) is true.

Double-click the Check for field model again but this time, choose the **LynxUnion** feature class.

Click OK to run the model.

After the model executes, preview the LynxUnion attribute table again.

The HabitatUtil field has been added as the last column in the table and has been populated with <Null>.

### Step 9: Close ArcCatalog

Close ArcCatalog

-----

**GO TO EXAM 1**

## **Module 5: Documenting Your Work**

The obvious outcomes of your geoprocessing workflow are databases with new datasets and toolboxes with new models and script tools. However, there is an important final step to your geoprocessing workflow—documentation.

Documentation is about communication and explanation. A GIS project can be a complicated and lengthy undertaking, and it might be tempting to skip the documentation step. However, all of your hard work is made much more valuable when you take the time to explain the tools, procedures, and methodology you used.

In this module, you will learn how to easily create supportive information for your geoprocessing work.

### **Documentation for geoprocessing**

When you have completed your geographic analysis, it's important to be able to explain the process by which data and custom tools were created as well as the reasoning behind your methodology. Your models and script tools should have accompanying instructions explaining what they do and what purpose they serve.

The ArcGIS Desktop geoprocessing framework includes functionality for keeping a record of geoprocessing tasks and documenting your work. Once you discover these tools and learn how you can take advantage of them, you will be ready to share your work with others.

Throughout this course, you have been working through a project to define habitat for the Canada lynx in northeastern Washington. In the exercises that follow, you will create different types of documentation for the tools you worked with in previous modules.

### **Why is documentation important?**

Documentation serves a number of functions. It can:

- Act as a reminder. If you want to revisit former work you have done, documentation will help you remember the reasons for the choices you made. If you make it a point to create documentation as you work through a lengthy project, it can help you recall what you've done so far and resume where you left off.
- Communicate with others. If your models and script tools are well-documented, it should be no problem for others to use them successfully. If they don't know how to use a tool, what exactly it does, or why a particular process is used, they may not be able to take advantage of the work you have done.
- Validate your work. Without adequate documentation to describe why you chose the methods and parameter values that you did, it may be difficult to convince others of your study's legitimacy.

### **Types of documentation**

In ArcGIS, there are several methods for documenting your custom models and script tools. You can:

- Generate a report from a model. A report describes the variables, processes, and current state of your model.



- Add labels to your model diagram. Model labels serve as visual reminders or notes explaining how your model works.
- Create metadata for any custom tool. Metadata describes what the tool does, who created it, keywords, and any constraints for using the tool.
- Build help for any custom tool, to teach others and to remind yourself how it works.

In addition to documentation that you create, ArcGIS will automatically record a history of your geoprocessing operations if you choose to log them to a log file (or a history model at 9.2). As you may recall, history models are saved in the History toolbox located in the My Toolboxes folder.

Adding a description to your custom tool's properties can also be considered a simple form of documentation. You learned how to do this earlier in this course.

### *Exercise 7: Run a report using ModelBuilder*

Earlier in this course you developed a model to identify potential lynx habitat in northeastern Washington. The result was a fairly complex model involving many tools, parameters, and datasets. With so many different components, a simple account of the model that categorizes its essential ingredients would be handy.

You can get this simpler view of the model by creating a report. A report is a useful way to create a quick snapshot of your model's variables and processes. It describes the components, parameters, and variables the model possesses. It also alerts you to any errors and provides the status of each process.

In this exercise, you will examine a few of the messages that are generated when you run a model. Next, you will create a report of the model and examine it. You will learn how to read the report so that you and others can easily understand how the model works.

**Note:** For this exercise to work properly, you must have Internet Explorer set as your default Web browser. Also, in Internet Explorer, click Tools > Internet Options > Advanced tab and uncheck the option "Reuse windows for launching shortcuts."

Estimated time to complete: 15 minutes

#### **Step 1: Start ArcCatalog**

Start ArcCatalog.

In the Catalog tree, navigate to your **Geoprocess\Document** folder.

Expand the LynxProject folder and the Tools folder.

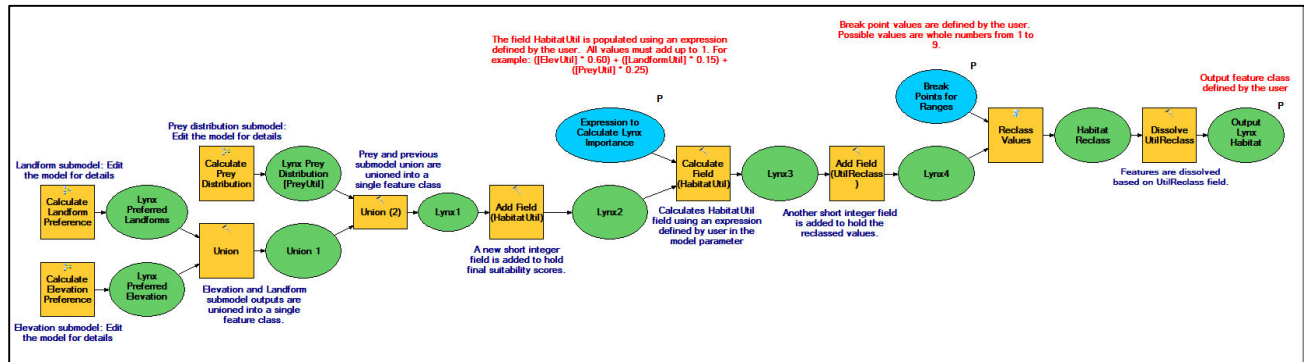
In the Tools folder, expand the LynxTools\_Document toolbox and the Working toolset.

The Final Lynx Habitat model you see here is similar to the one you created in the previous module.

## Step 2: Run a model

When you run a model, messages are generated for every tool. You can view these messages interactively in ModelBuilder.

Right-click the Final Lynx Habitat model and choose Edit.



As you can see, the model now contains labels. You will work with these labels in the next exercise.

Run the model.

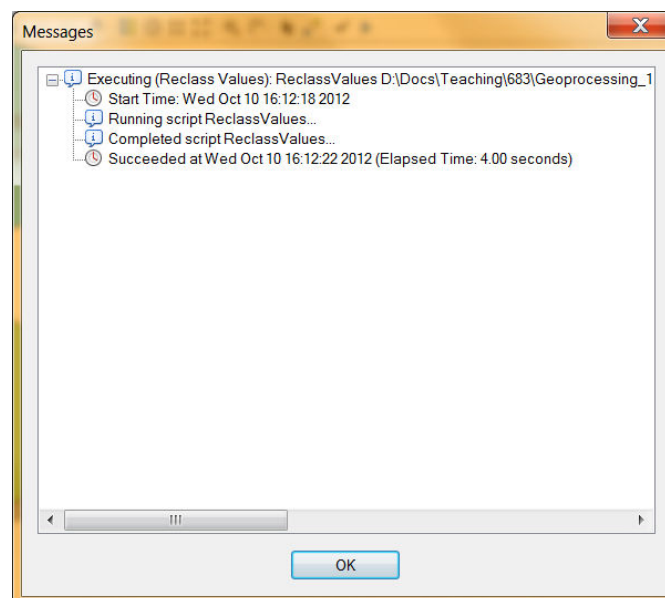
As the model runs, messages are stored with each tool indicating the status of the process.

When the model finishes processing, close the progress window, if necessary.

## Step 3: View a process message in ModelBuilder





Zoom in to the Reclass Values script tool element (next to last model tool element).

Right-click the Reclass Values element and choose View Messages.



**Note:** The messages you see will be slightly different from the View Result graphic above.

Icons identify four types of messages. A process can contain one or more of each type.

- An info balloon  describes which process is executing and whether it was successful.
- A clock  indicates the start and end times of a process.
- A yellow warning  flags possible errors. These errors usually include required parameter values that are missing or names of output datasets that already exist.
- A error icon  tells you that a process failed to run.

If you like, explore the messages for other tools in the model.

When you're finished, close the Messages dialog box.

#### Step 4: Create a report

You can consolidate all of the messages, as well as other descriptive information about the model by creating a report.

From the Model menu, choose Report.

There are two options. The first allows you to simply view the report. The second saves the report to an [XML](#) file.

When a report is generated, it captures the current state of the model. If you were looking for a quick answer, the first option is likely the best choice. However, if you plan to use the report to compare the current model with different versions, it's best to save the report to a file.

Click the option to "Save report to a file."

Browse  to your **Document\LynxProject** folder.

Name the report **LynxHabitatModel\_report.xml** and click Save.

In the Model Report dialog box, click OK to create the report.

Close the model. If prompted, click Yes to save the model.

#### Step 5: Open the report

In the Catalog tree, right-click the LynxProject folder and choose Refresh to see the XML file you just created.

Double-click the report to open it in a new browser window.

**Note:** If another program (i.e. DreamWeaver) has attempted to open the file, right click on the file in ArcCatalog and select properties. In the General Tab, select which program should open the file by clicking on the Change button. Also if Internet Explorer displays a security message that content has been blocked, click the message, then choose Allow blocked content. This will allow you to expand and collapse the report contents.

Notice that the report is divided into two categories, Variables and Processes.

## Step 6: Examine the report

A report captures the current state of the model. It includes the data type and value of variables, the tool source and parameters, and messages explaining the status of each process.

There are two ways to view the details of the report, one element at a time or all at once.

Under Variables, click Lynx Preferred Elevation [ElevUtil].

The details about that variable are exposed.

Alternatively, you can expose the details for all model elements.

In the upper right corner of the report, click the Expand/Collapse All link.

The details for all variables and processes now display.

Click Expand/Collapse All again to hide the details.

In the previous modules, you created three model parameters: Expression to Calculate Lynx Importance Ratios, Break Points for Ranges, and the Output Lynx Habitat Feature Class. Using a report, you can determine what these values were when the model was last run.

Find the variable called Expression to Calculate Lynx Importance Ratios.

Click the name to expand it.

Below the name of the variable, you will find the variable's data type and value.

Under Processes, find the Reclass Values process and click its name to expand it.

Click Parameters.

A table appears that contains details about the parameters used by the Reclass Values tool. (*Note: To see the table you may have to click the Parameters link.*)

Click Messages (below the Parameters section).

A list of messages about the Reclass Values process appears. Notice that the messages for this tool are the same as when you viewed them in the model.

If you like, explore the report further.

When you're finished, close the report.

If you are continuing to the next exercise, leave ArcCatalog open. Otherwise, close ArcCatalog.

---

## Exercise 8: Create and edit model labels

Documentation helps remind you and others about what you did and why you did it. In ModelBuilder, you can add labels to a model. Model labels can be informal notes about particular processes or more formal information used for presentation purposes, such as a title, the author's name, and the date the model was produced.

In this exercise, you will create additional labels for the Final Lynx Habitat model, which has been partially annotated already. You will also modify existing labels by changing the font, color, size, and background color.

Estimated time to complete: 20 minutes

### Step 1: Open the Final Lynx Habitat model

Open ArcCatalog, if necessary.

In the Catalog tree, navigate to your **Document\LynxProject** folder.

Expand the Tools folder, the LynxTools\_Document toolbox, and the Working toolset.

Open the Final Lynx Habitat model in edit mode.

In ModelBuilder, zoom in and examine some of the labels that have already been created.

Well placed labels can tell the story about what your model does, as well as emphasize various important elements.


There are two types of model labels to consider: free-floating labels and labels attached to model elements. You will learn how to create each type in the next two steps.

### Step 2: Create a free-floating label

Free-floating labels remain where you put them regardless of how you position model elements. They are useful for general paragraphs or statements, titles, names, dates, and so forth.

Right-click in an empty space above the model and choose Create Label.

A free-floating label is added to the model space.

Using the Select tool , double-click the label to edit the text.

Type **Lynx Habitat Model** and press Enter.

Right-click the label and choose Display properties to open the Display properties dialog box.

The Display properties dialog box is used to specify attributes for your labels.

Select the cell for Font, then click the ellipsis button  in the column to the right.

In the Font dialog box, change the size to **72**.

Make sure that color is set to black and the style is bold.

Click OK.

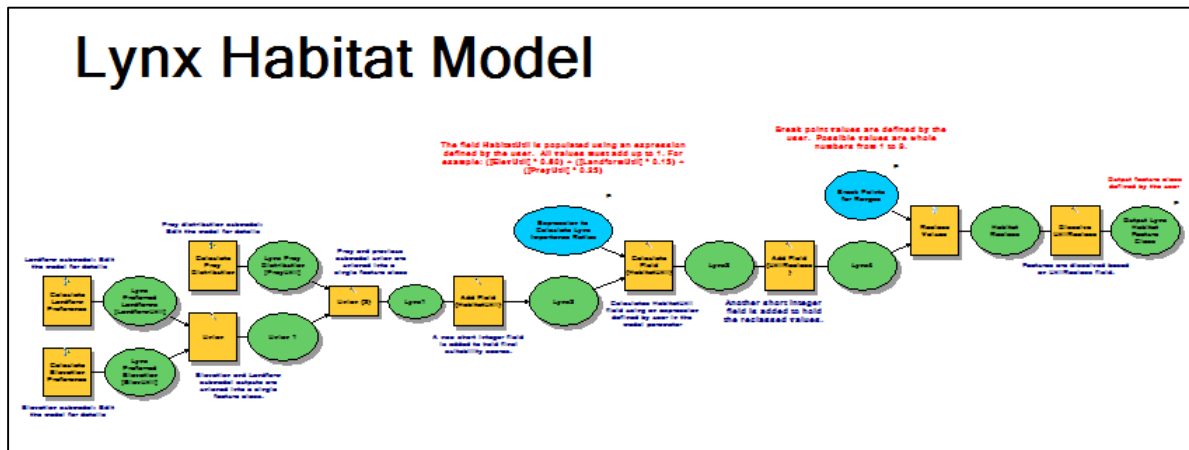
The new font is applied to the label.

Close the Display properties dialog box.

You can move this free-floating label anywhere within the model space—it is not associated with any model element.

Position the label so that it is centered at the top of the model, zooming out as needed.

Click in an empty area to unselect the label.




### Step 3: Create a label attached to a model element

A label that is attached to a model element will move when the element is moved.

Use the Zoom and Pan tools to navigate to the Reclass Values element.

Right-click Reclass Values and choose Create Label.

Recall that this element is a Python script tool that reclassifies utility values based on your input for the Break Points for Ranges parameter. Here, you will create a label to remind yourself of this fact.

With the Select tool , click somewhere in empty white space to unselect the element and attached label, then select the label only.

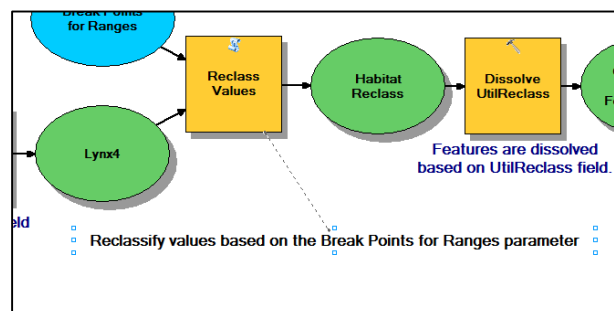
Double-click the label and type the following text:

**Reclassify values based on the Break Points for Ranges parameter.**

Press Enter.

Move the label below the Reclass Values element.

As you do, notice the dashed line that connects the label to the element. This indicates that the label is attached to the element.



Test this by selecting the Reclass Values element and moving it down slightly.

The relative position of the label is maintained when the element is moved.

In the next step, you will modify this new label.

#### Step 4: Edit the label properties

The description that you typed for the Reclass Values element does not have to appear as one line.

Double-click the label text to make it editable.

Click inside the text and place your cursor immediately in front of the word *on*.

Hold down the Ctrl key and press Enter to create a line break.

Create another line break after the word *Points*.

With the label selected, right-click it and choose Display properties.

In the Display properties dialog box, click Background color, then the ellipsis button, and change the color to a gray.

Click OK.

Change the font to have a white color and a size of 12.

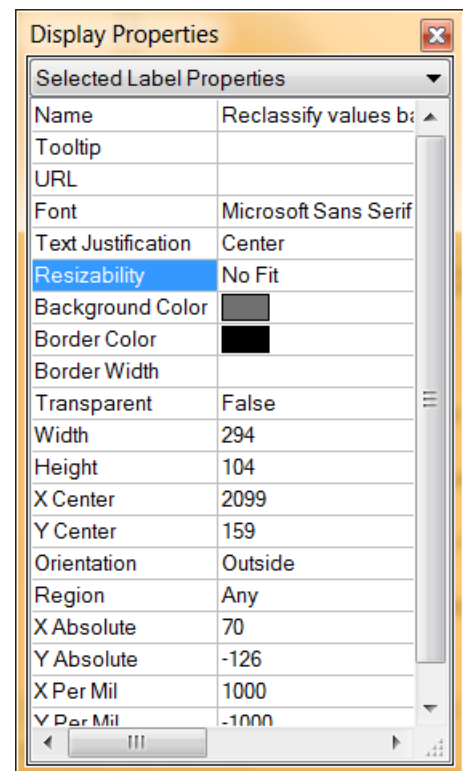
For the Resizability property, choose No Fit from the drop-down list.

Close the dialog box.

Did you notice that the blue anchor points surrounding the label changed from hollow to solid? When the anchors are solid, they are adjustable.

Click the lower-right anchor and resize the label box, making it large enough that the entire label fits within the gray box.

Move the label so that it fits neatly below the Reclass Values element.



#### Step 5: Modify existing label properties

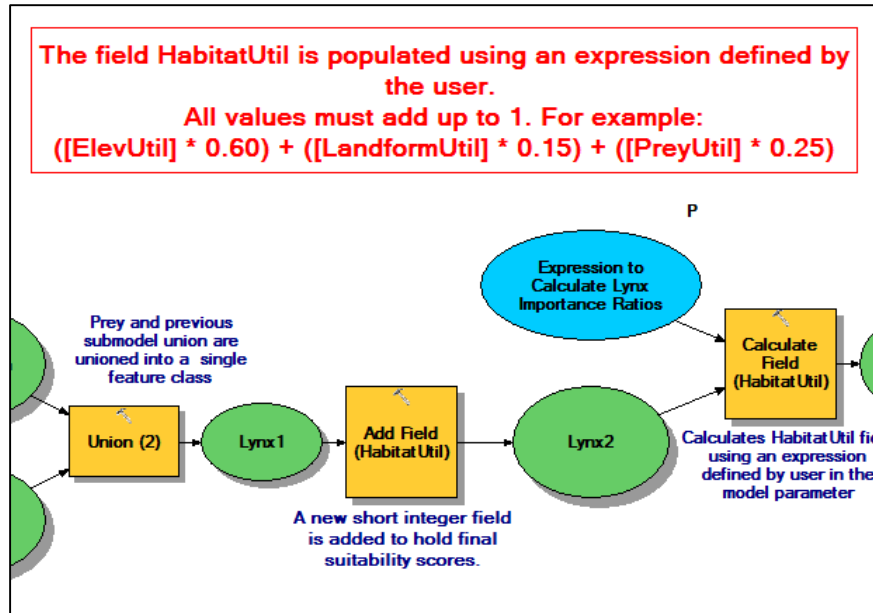
Now that you know how to modify model label properties, try modifying one on your own.

Zoom out to the full extent of the model.

This model has three model parameters. Each parameter element has an attached label, displayed with a red font.

For the first parameter's label, add a thin red border and increase the font size to 14.

For the Resizability property, choose No Fit from the drop-down list.



### Step 6: Export the model as a graphic

Once you are satisfied with the labels and appearance of the model, you can export everything to a single graphic file that you can add to map layouts, metadata, help documentation, and Web pages.

In this step, you will create a JPEG image (.jpg file) that you will use in the next exercise.

From the Model menu, choose Export > To Graphic.

For file type, choose JPEG Image (\*.jpg).

Browse to your **Document\LynxProject** folder and name the file **LynxHabitatModel.jpg**.

Click OK.

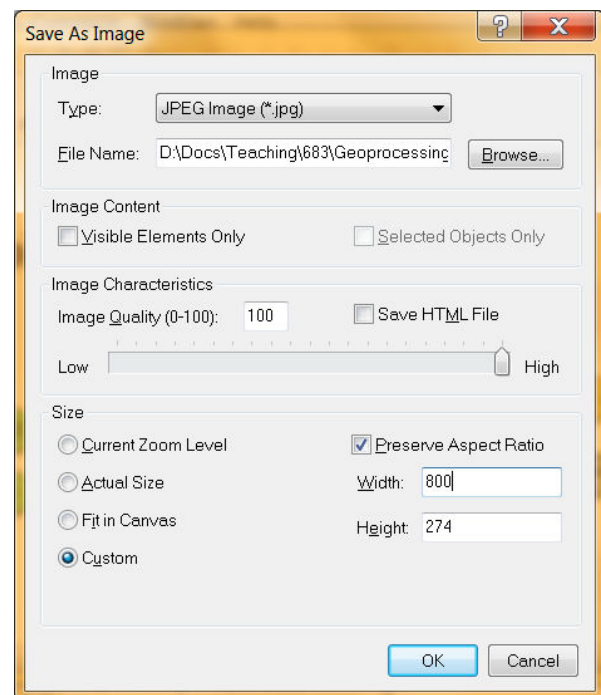
In the Size section of the Save As Image dialog box, click the Custom option.

If necessary, check the box next to "Preserve Aspect Ratio."

For Width, type **800**.

**Note:** The height shown in your dialog box may differ from that shown in the View Result graphic below.

Click OK.



### Step 7: Save the Model

Close the Final Lynx Habitat model and click Yes to save your changes.

Now you will preview the image you created in the last step.



In the Catalog tree, right-click the LynxProject folder and choose Refresh.

If you do not see the graphic file, add **JPG** to the list of file types that are shown in the Catalog.

If you are continuing to the next exercise, leave ArcCatalog open. Otherwise, close ArcCatalog.

-----

## Metadata and help for custom tools

You've seen how easy it is to generate a geoprocessing report and add labels to models. The other two forms of documentation—metadata and help—are created using the ArcToolbox Documentation Editor. Once you get familiar with the editing environment, you will find that creating metadata and help for your models and script tools is also an easy task.

The Documentation Editor is divided into two sections: General Information and Help.

### *General Information*

The General Information section is where you briefly describe the tool and provide keywords, author information, and use constraints. What you enter here can be viewed by selecting the tool in the Catalog tree and viewing its metadata.

General Information	
Section	Description
<b>Abstract</b>	The abstract is a short paragraph that describes the function of the tool. The abstract will appear in the tool's metadata and in the help panel when you run the tool from its dialog box.
<b>Keywords</b>	Keywords are used for finding your tools. As long as the tools have been added to ArcToolbox, you can use the Search function to locate the tools based on keywords.
<b>Author</b>	The author section is where you take ownership of your work. Information that you provide here will make it easier for those who have received your tools to contact you.
<b>Constraints</b>	Constraints are restrictions you want to specify about your tools. They might include copyright information, distribution limitations, the proper use of the tool, or a disclaimer.

### *Help*

The information that you enter in the Help section populates two locations: the quick reference guidelines that can be viewed from the help panel of a tool's dialog box, and the comprehensive help page that can be launched from the tool's dialog box or accessed by right-clicking the tool in ArcToolbox and choosing Help.

Help	
Section	Description
<b>Summary</b>	A summary is a more detailed description of your tool. It can be similar to or the same as the abstract in the General Information section, but here you are able to create paragraphs, bulleted lists, subsections, and hyperlinks.

<b>Illustration</b>	You can choose to include a graphic that describes your custom tool.
<b>Usage Tips</b>	Usage tips can include reasons why you would use the tool as well as important restrictions.
<b>Parameters</b>	<p>There are two sections for each parameter.</p> <ul style="list-style-type: none"> <li>• The Dialog Reference is for specifying what appears in the help panel of a tool's dialog box. To view this text, select the parameter's text box in the dialog box.</li> <li>• The Command Reference appears on the tool's help page. If you have an entry in the Command Reference, but not the Dialog Reference (or vice versa), the other will appear in its place.</li> </ul>
<b>Command Example</b>	Contains sample code for running the tool at the command line.
<b>Script Example</b>	Contains sample code for including the tool in a script.

### *Exercise 9: Work with the Documentation Editor*

When creating models and script tools, it is important to include detailed documentation in order to tell the whole story. You can do this by creating metadata and help. The metadata describes what the tool does and who created it. The help guides the user through using the tool. Both are stored with the tool.

In this exercise, you will learn how to use the Documentation Editor to create metadata and help for the Final Lynx Habitat model.

**Note:** To do this exercise, you must have successfully completed the previous exercise, *Create and edit model labels*. Also, you must have Internet Explorer set as your default Web browser.

Estimated time to complete: 35 minutes

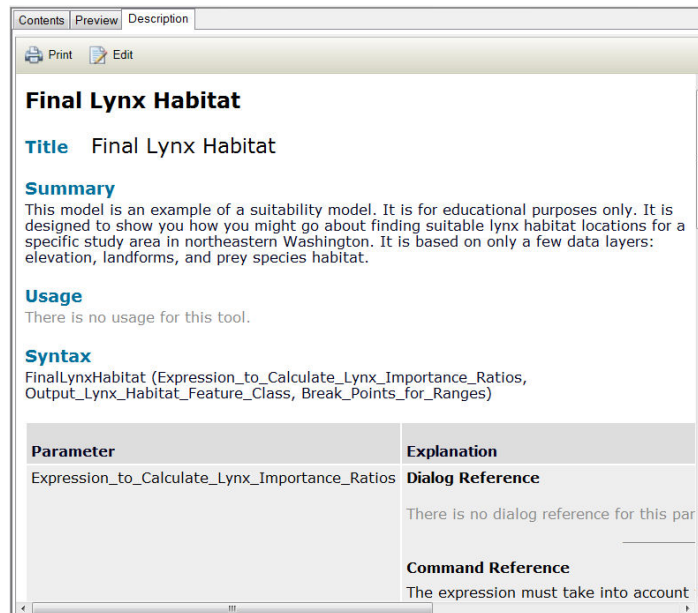
#### **Step 1: Examine model metadata**

Open ArcCatalog, if necessary. In the Catalog tree, navigate to your **Document\LynxProject\Tools** folder.

Expand the LynxTools\_Document toolbox and the Working toolset.

Click the Final Lynx Habitat model and then click the Description tab on the right.

As you can see, metadata has been created for the model.



Later in the exercise, you will modify some of the metadata using the Documentation Editor.

Take a few moments to review the abstract, contact information, and keywords.

Next, you will look at the model's help documentation.

## Step 2: Examine the model help

Just as system geoprocessing tools include help documentation, custom tools like the Final Lynx Habitat model can also be supplemented with help.

Double-click the Final Lynx Habitat model to open its dialog box.

Click the Show Help button at the bottom of the dialog box to display the help panel.

At the top of the Tool Help panel, you'll notice the same short abstract that you viewed in the model's metadata.

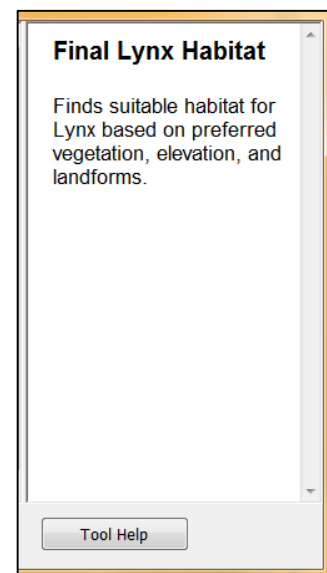
In the dialog box, click in each parameter's text box and read the help for each.

Instructions have been created for each parameter except Break Points for Ranges. You'll create help for this parameter later in the exercise.

Click the Tool Help button.

**Note:** If Internet Explorer displays a security message, right-click the message and choose Allow blocked content.

A new browser window appears that contains a more comprehensive help page. Notice that the format is the same as that of any system tool.



## Final Lynx Habitat

**Title** Final Lynx Habitat

### Summary

This model is an example of a suitability model. It is for educational purposes only. It is designed to show you how you might go about finding suitable lynx habitat locations for a specific study area in northeastern Washington. It is based on only a few data layers: elevation, landforms, and prey species habitat.

### Usage

There is no usage for this tool.

### Syntax

FinalLynxHabitat (Expression\_to\_Calculate\_Lynx\_Importance\_Ratios, Output\_Lynx\_Habitat\_Feature\_Class, Break\_Points\_for\_Ranges)

Parameter	Explanation	Data Type
Expression_to_Calculate_Lynx_Importance_Ratios	<b>Dialog Reference</b> There is no dialog reference for this parameter.  <b>Command Reference</b> The expression must take into account the three variables: elevation, landforms, and prey species habitat. Multiply each input by a percentage represented in hundredths. The sum of the three percentages must add up to one. Example: $([ElevUtil] * 0.60) + ([LandformUtil] * 0.15) + ([PreyUtil] * 0.25)$	SQL Expression
Output_Lynx_Habitat_Feature_Class	<b>Dialog Reference</b> There is no dialog reference for this parameter.  <b>Command Reference</b> Name the output. You must include the path and geodatabase where the output feature class is to be saved.	Feature Class
Break_Points_for_Ranges	<b>Dialog Reference</b> There is no dialog reference for this parameter.  <b>Command Reference</b> There is no command reference for this parameter.	Multiple Value

### Code Samples

#### Script Example

```
# Import system modules
import sys, string, os, win32com.client

# Create the Geoprocessor object
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")
```

Compare the Command line syntax and Scripting syntax sections.

The text for these two sections is almost identical because both sections describe the same parameters; however, the syntax is different for each. Also notice that the Explanation text is the same as that listed in the help panel of the tool's dialog box for each parameter.

Explanation text has not been created for the Break Points for Ranges expression in both sections. You will fill in these fields later on.

When you are finished examining the help, close the window and the Final Lynx Habitat dialog box.

### Step 3: Open the documentation editor

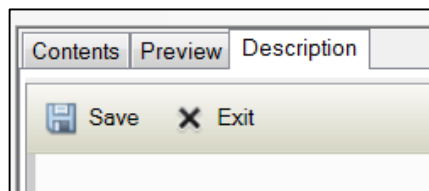
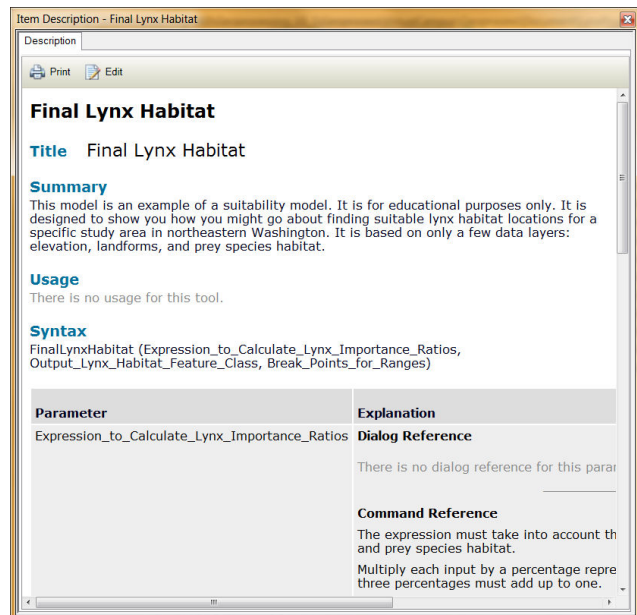
Now you will explore the environment in which you create and edit metadata and help for custom tools—the Documentation Editor. The Documentation Editor is accessed from either from ArcToolbox or the Description tab in ArcCatalog.

To view it in ArcToolbox:

- If necessary, open ArcToolbox.
- Drag the LynxTools\_Document toolbox from the Catalog tree and drop it into the ArcToolbox window.
- In ArcToolbox, expand the LynxTools\_Document toolbox and the Working toolset.
- Right-click the Final Lynx Habitat model and choose Item Description.

To view/edit Metadata in ArcCatalog:

- Click on a model, shapefile, etc in the Catalog Tree and then click on the Description tab in the viewing window



You use the Documentation Editor to create both metadata and help for custom tools.

Notice that the abstract text is the same as what you saw in the metadata and the help for the Final Lynx Habitat model.

### Step 4: Complete metadata for the model

The model's metadata is created in the General Information section. In this step, you will add to the abstract and keywords portions of the metadata.

Click the Edit button at the top of page.

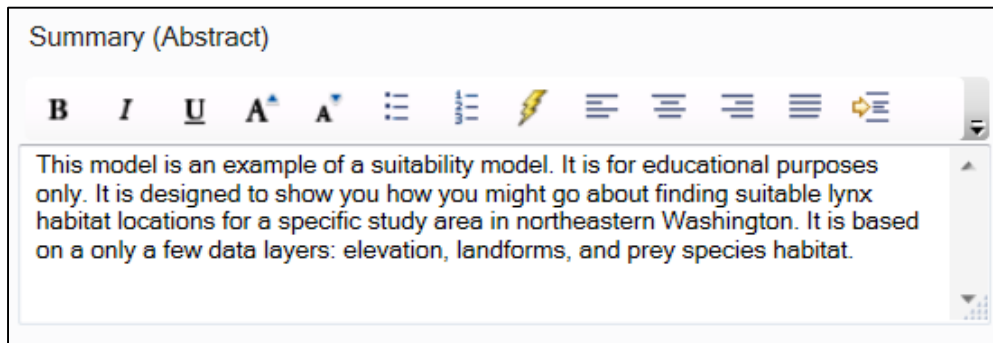
As you scroll down the Documentation Editor (the table of contents), you will notice the Summary(Abstract) section.

You will replace the existing abstract text with new text.

Highlight and copy the following text (It may be necessary to remove the bold formatting and decrease the text size):

**This model is an example of a suitability model. It is for educational purposes only. It is designed to show you how you might go about finding suitable lynx habitat locations for a specific study area in northeastern Washington. It is based on only a few data layers: elevation, landforms, and prey species habitat.**

Paste the copied text over the existing text in the Abstract.



Next, edit the Tags.

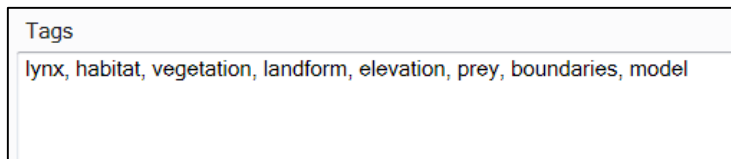
Using appropriate tags or keywords will help you (and others) find the tool when searching for it in ArcToolbox.

As you can see, keywords have been created for you, but you will add a few more.

Click the empty space behind the last keyword (vegetation).

Type the words:

**landform**  
**elevation**  
**prey**  
**boundaries**  
**model**



If you like, scroll down and view the Credits (Author) and Use Limitations to see how those sections have been filled in. Only the Abstract is repeated in the tool's help.

### Step 5: Preview the Help Section

Now you will take a look at the Help section of the Documentation Editor.

Under Syntax, expand each of the 3 entries.

Syntax

Expression\_to\_Calculate\_Lynx\_Importance\_Ratios

Dialog Explanation

Scripting Explanation

The expression must take into account the three variables: elevation, landforms, and prey species habitat. Multiply each input by a percentage represented in hundredths. The sum of the three percentages must add up to one.

Example:

$$([ElevUtil] * 0.60) + ([LandformUtil] * 0.15) + ([PreyUtil] * 0.25)$$

Output\_Lynx\_Habitat\_Feature\_Class

Break\_Points\_for\_Ranges

Code Samples

Title

The three items you under syntax correspond to the three model parameters. Remember, you reviewed the help text for these parameters earlier in this exercise.

Each parameter has two sections that can be completed: Dialog Explanation and Scripting Explanation.

Notice that the content in the Documentation Editor appears in the Scripting Explanation section that corresponds to the <Expression\_to\_Calculate\_Lynx\_Importance\_Ratios> parameter.

Also notice, as you did in step 2, that Explanation text has not been created for the Break Points for Ranges expression. You will add it in the next step.

In the Syntax list, collapse Expression to Calculate Lynx Importance Ratios.

You might wonder why there is no content for the Dialog Explanation sections of these parameters. If nothing is entered for Dialog Explanation, the help panel of the tool's dialog box will be populated with the content that was entered for Scripting Explanation, and vice versa. That way, if you want the content to be the same for both forms of help, you only have to enter the text once. You will see how this works in the next step.

### Step 6: Create help for the Break Points for Ranges parameter

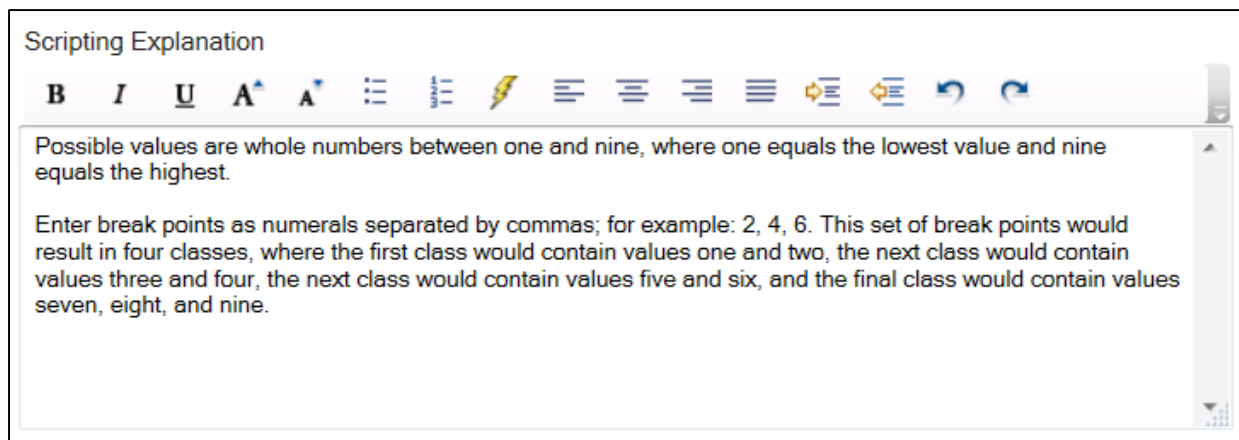
In the box on the right, copy and paste the following sentence into the Scripting Explanation of the <Break\_Points\_for\_Ranges> parameter:

**Possible values are whole numbers between one and nine, where one equals the lowest value and nine equals the highest.**

For the second paragraph, copy and paste the following text:

**Enter break points as numerals separated by commas; for example: 2, 4, 6. This set of break points would result in four classes, where the first class would contain values one and two, the next class would contain values three and four, the next class would contain values five and six, and the final class would contain values seven, eight, and nine.**

Highlight the entirety of the text and unbold the text.



Next, click in the Dialog Explanation under Break Points for Ranges.

This is where you can provide separate instructions for the help panel in the tool's dialog box. Remember, if you do not have any text in the Dialog Explanation section, the instructions will be the same as the Scripting Explanation sections. In this case, you will change the instructions slightly.

Add a paragraph to the Dialog Explanation section, then copy/paste the following text:

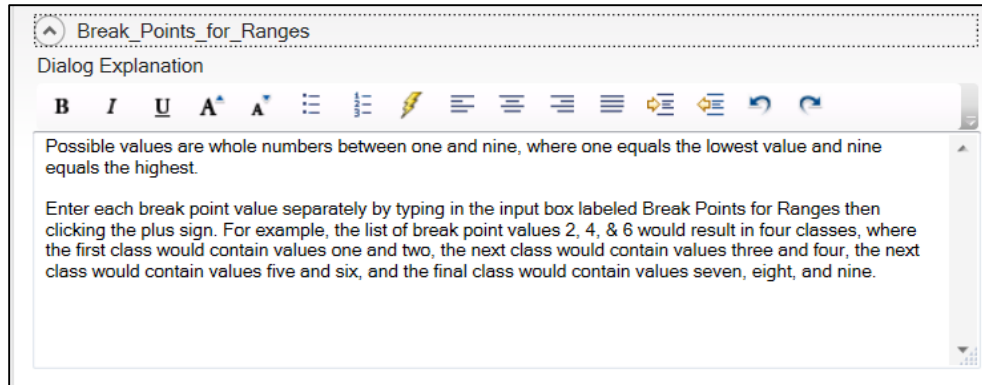
**Possible values are whole numbers between one and nine, where one equals the lowest value and nine equals the highest.**

This is the same sentence that you used for the Scripting Explanation because it is true whether the model is being run from the command line or from a dialog box.

Add a second paragraph to the Dialog Explanation section and populate it with the following text and remember to remove the bold formatting:

**Enter each break point value separately by typing in the input box labeled Break Points for Ranges then clicking the plus sign. For example, the list of break point values 2, 4, & 6 would result in four classes, where the first class would contain values one and two, the next class would contain values three and four, the next class would contain values five and six, and the final class would contain values seven, eight, and nine.**





To complete the help, you will add the graphic of the model that you created in the previous exercise as an illustration.

### Step 7: Add an illustration to the help documentation

Scroll up to the top of Documentation Editor (Item Description), under Thumbnail, click the Update... button.

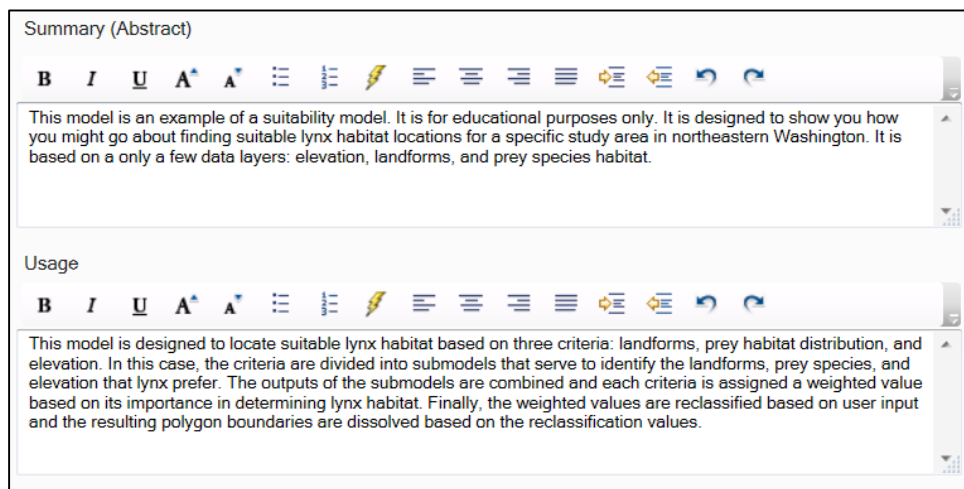
For Path, browse to your **Document\LynxProject** folder.

Select the **LynxHabitatModel.jpg** file that you created in the previous exercise, then click Open.

For Title, type **Lynx Habitat Model**.

For Usage, add the following text:

**This model is designed to locate suitable lynx habitat based on three criteria: landforms, prey habitat distribution, and elevation. In this case, the criteria are divided into submodels that serve to identify the landforms, prey species, and elevation that lynx prefer. The outputs of the submodels are combined and each criteria is assigned a weighted value based on its importance in determining lynx habitat. Finally, the weighted values are reclassified based on user input and the resulting polygon boundaries are dissolved based on the reclassification values.**



Click Save and close the Item Description.

## Step 8: Examine the documentation

Open the Final Lynx Habitat model's dialog box and, in the Help panel, verify that the new Abstract text appears.

Click in the Break Points for Ranges text box.

The new information that you entered for the Dialog Reference in the previous step displays.

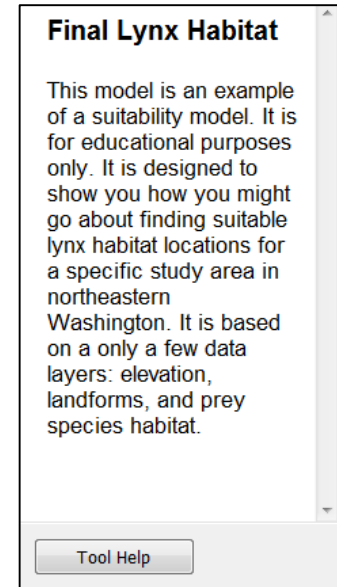
Next, click the Tool Help button.

Look at the Command line syntax and Script syntax sections.

The Explanation text is now filled in for the Break Points for Ranges parameter.

As you noted earlier, the Reclass Values element does not have a description. You will add this in the next step.

Close the help page and the model dialog box.



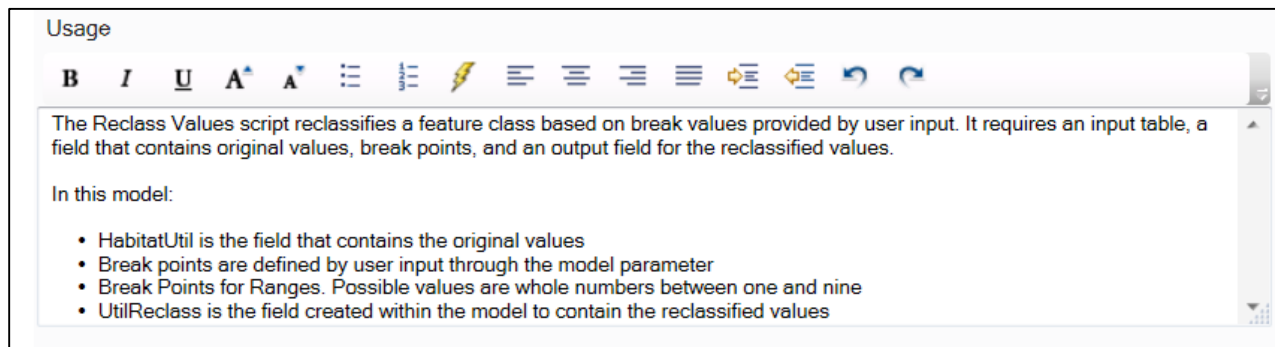
## Step 9: Document a script tool

ModelBuilder provides a way to document individual processes that make up a model.

In the Catalog Tree, click on the script tool Reclass Values and activate the description Tab.

Click on Edit.

The toolbar should be familiar to you, so use what you have learned in the previous steps and follow the image below to create documentation for the Reclass Values script.



Click Save to save your work and close the editor.

The content that you added to the Reclass Values Documentation dialog box now appears as an explanation for the Reclass Values element in the Model section of the help page.

Close the help window.

Close ArcCatalog.

-----

## Sharing your work

Sharing your project data and tools usually falls into one or more of the following categories:

### *Sharing the results*

You may share only the results of your analysis, such as maps, reports, and final data. These types of products are usually designed for specific audiences.

### *Sharing the procedures*

You may want people to review your procedures. Sharing procedures usually means that you are inviting feedback and criticism, which is used to improve your analysis.

### *Sharing the tools*

You may share your scripts, script tools, and models so that others can run the tools either with your data or adapt the tools to their own projects if possible.

For your projects, there may be an audience for each of these categories and you might have to decide on several methods for sharing your work. This short topic offers some basic guidelines you should follow when distributing geoprocessing tools and data.

## Considerations for sharing your work

To share your work, you must decide on one or more delivery methods. For example, you can simply deliver a presentation, either live or over the Web; make the tools and data accessible through your organization's network; download the project files and folders to a CD and distribute it; or archive the project in a compressed file that people can download.

The following table outlines some guidelines you should consider for each method. The guidelines are described below the table.

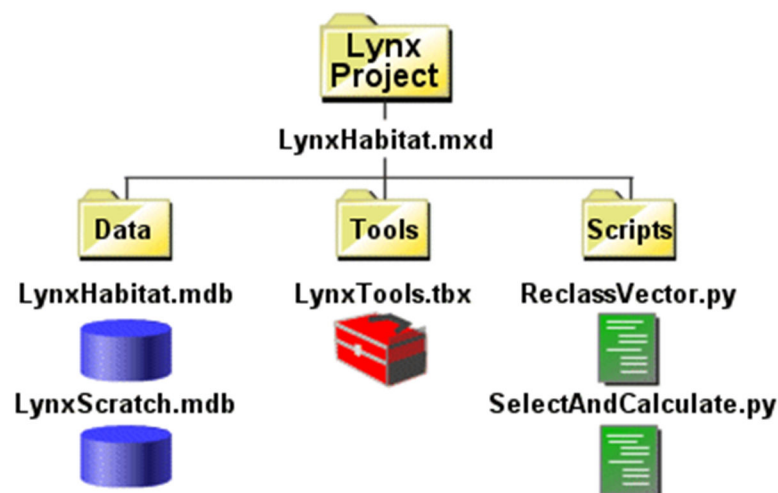
Delivery method	Organization	Relative path	File permissions	Documentation	License issues
Live presentation	✓			✓	
Web presentation	✓		✓	✓	
Network access	✓	✓	✓	✓	✓
CD or DVD	✓	✓		✓	✓
Archive (ZIP, TAR or CAB)	✓	✓		✓	✓

- Keep your project data and tools **organized** in a geodatabase or logical file structure. You learned how to do this earlier in this course.
- Setting [relative paths](#) enables others to use your custom tools. If absolute paths are saved, the model or script will look for its inputs in the wrong locations—in the folders on the model creator's computer or network where the inputs were stored when the custom tool was created. You learned how to store relative paths for custom tools the previous modules.
- Setting **file permissions** lets you protect folders or files that you do not want changed. You have not done this in the course, but you can use your file manager to enforce different permissions for folders and files.
- **Document** your custom tools with labels, metadata, and help. You learned how to do this in the previous topic.
- Be aware of ArcGIS **licensing** issues. Others will not be able to run models or scripts that contain geoprocessing tools that are unavailable at their license level. Consult the *Geoprocessing Commands Quick Reference Guide* located in the Geoprocessing tool reference book of ArcGIS Desktop Help for more information.

## Sharing the lynx project

In the graphic below, pause your mouse pointer over the items on the left to see how some basic sharing guidelines might be applied to a GIS project such as the lynx project that you have worked on throughout this course.

- [Organization](#)
- [Relative paths](#)
- [Permissions](#)
- [Documentation](#)
- [Licensing](#)



In this course, the data, tools, scripts, map documents, and other related files for the lynx project were organized using system folders.

Intermediate and final data was stored in geodatabases located in the Data folder. Custom tools, like models and script tools, were stored in a custom toolbox in the Tools folder, while script files were stored in a Scripts folder. A logical organization schema makes it easy to find what you are looking for.

**GO TO EXAM 2**