defined parameter, you must define a list of properties when you create the script tool in a toolbox. These properties can be altered but require the juggling of two files and coordinated saves to make them work.

As described in the introduction to this chapter, the Python toolbox tools have all their parameters defined within the Python script. Editing becomes faster and easier, and there are more options than in a script tool. When moving code from a stand-alone script or script tool to the Python toolbox, you should look through the code and find occurrences of the ArcPy parameter commands. These commands will be moved into the parameters module of the Python toolbox code along with any necessary filters or settings.

Scenario

In tutorial 2-5, you wrote an application to use the selected feature in the Fire Department's box zone map to summarize the buildings it contained. In tutorial 2-6, you made the tool interactive by adding a user interface to accept the box number and the building types to summarize. The input for building types had an associated value list to ensure that the user selected an appropriate code, with a chart of the building type descriptions then displayed in the context Help.

For this tutorial, take the code you developed for tutorial 2-6, and turn it into a Python toolbox tool. You may also want to add validation code to ensure that the box number the user enters is valid.

Data

A completed version of the code is in text file Tutorial 4-2 in the Data folder, so your task is to move the code into the Python toolbox structure to make it more portable. The items to be selected are the building footprints for all of Oleander, which are stored in the feature class BldgFootprints in the Planimetrics folder in the City of Oleander geodatabase. The feature class has a field named UseCode, which contains a code for each building type. The following list contains the codes for the building types:

1 = Single Family
2 = Multi-Family
3 = Commercial
4 = Industrial
5 = City Property
6 = Storage Sheds
7 = Schools
8 = Church

The data being used for the selections is a set of polygon feature classes of box zones in the FireBoxMaps feature dataset in the City of Oleander geodatabase. Looking at these feature classes, you will see that there are a large number of these files, one for each box zone. Note also that the numbers are not sequential. Some ranges are skipped, and some box zones appear in two files, which will affect the valid list of choices when you build the data integrity rules.

SCRIPTING TECHNIQUES

You learned in this chapter's introduction how to build a value list for a parameter, and in this tutorial, you will build a value list for the building type codes. Set that parameter's filter type property to Value List, and define a filter list object containing the choices.

You will also create custom Help for this tool. The user will need to know what the building codes mean to be able to make appropriate choices.

Add some code to validate the box number entry. There are too many box numbers to make a usable value list, but it would be a good data integrity practice to make sure the number entered by the user is an actual box number. Verifying the box number is done in a special validation area of the Python toolbox code—the updateParameters module in each tool class. Commands in this area are called anytime an input parameter is changed. The simple validation is to ensure the number entered for the box number is within the allowable range. If the number is not within this range, an error message can be generated, and the script can be held until a proper number is entered.

Set up value validation

1. **Open the map document Tutorial 4-2 and text file Tutorial 4-2 in the Data folder. Examine the script from tutorial 2-6, as shown:**

```
# Import the modules
import arcpy


# Set up the environment
arcpy.env.workspace = r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb"

# Get input from the user
#    The first will be the box number to act upon - index 0
#    The second will be the building type to count - index 1
boxNumber = arcpy.GetParameterAsText(0)
buildingType = arcpy.GetParameterAsText(1)



# Make feature layers from the user input
boxLayer = arcpy.MakeFeatureLayer_management(r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb\FireBoxMaps\FireBoxMap_" \
 + str(boxNumber))
buildLayer = arcpy.MakeFeatureLayer_management(r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb\Planimetrics\BldgFootprints", \
"\"UseCode\" = '" + str(buildingType) + "'")

# Use the specified file of box zone to select specified type of building
arcpy.SelectLayerByLocation_management(buildLayer, "HAVE_THEIR_CENTER_IN", boxLayer,"","SUBSET_SELECTION")

# Count the selected features
bldgCount = int(arcpy.GetCount_management(buildLayer).getOutput(0))

# Display the results in the geoprocessing Results window
arcpy.AddMessage("The count of buildings is " + str(bldgCount) + ".")

# Create a field to store the results

# 1 = Single Family (SFCount)
# 2 = Multi-Family (MFCount)
# 3 = Commercial (ComCount)
# 4 = Industrial (IndCount)
# 5 = City Property (CityCount)
# 6 = Storage Sheds (ShedCount)
# 7 = Schools (SchCount)
# 8 = Church (ChurCount)

if buildingType == 1:
    newField = "SFCount"
elif buildingType == 2:
    newField = "MFCount"
elif buildingType == 3:
    newField = "ComCount"
elif buildingType == 4:
    newField = "IndCount"
elif buildingType == 5:
    newField = "CityCount"
elif buildingType == 6:
    newField = "ShedCount"
elif buildingType == 7:
    newField = "SchCount"
else:
    newField = "ChurCount"

arcpy.AddField_management(boxLayer,newField,"LONG")

# Store the results in the field
arcpy.CalculateField_management(boxLayer,newField,bldgCount)
```

2. **Right-click your MyExercises folder, and create a new Python toolbox named** Fire Department.pyt. **Open the toolbox for editing.**

3. **In the Toolbox class, set the label to** Fire Department Tools **and the alias to** fdept. **Then replace the default value Tool with** BoxBldgCount, **as shown:**

```
class Toolbox(object):
    def __init__(self):
•       """Define the toolbox (the name of the toolbox is the name of the .pyt file)."""
•       self.label = "Fire Department Tools"
•       self.alias = "fdept"

        # List of tool classes associated with this toolbox
•       self.tools = [BoxBldgCount]
```

4. **Change the name of the Tool class to** class BoxBldgCount(object).

5. **In the _init_ module of the BoxBldgCount class, change the label to** Box Zone Building Count **and the description as shown:**

```
class BoxBldgCount(object):
    def __init__(self):
•       """Define the tool (tool name is the name of the class)."""
•       self.label = "Box Zone Building Count"
•       self.description = "The user will enter a box number and a building type." + \
        " The tool will add a field and store the building count for the selected type."
•       self.canRunInBackground = False
```

Next set up the two input parameters. The first is simply a request for the user to enter a number (integer). Look up the keyword for this data type in ArcGIS for Desktop Help.

6. **Add the code to accept the box number from the user, and set up the other properties of the input parameter to match the original application, as shown:**

```
    def getParameterInfo(self):
•       """Define parameter definitions"""
•       param0 = arcpy.Parameter(
•       displayName="Enter the Box Number",
•       name="inBox",
•       datatype="GPLong",
•       parameterType="Required",
•       direction="Input")
```

The numbers entered into this parameter must correspond to the box zone numbers. Without this number validation, the user could enter a number that would cause an error in the script. The validation code is written in the updateMessages module. The updateMessages module is used to provide feedback to the user based on the values they enter as parameters and is displayed to the user before they click OK to run the tool. There is an important distinction here. The updateParameters module validates user input as it is typed, and the updateMessages module can return feedback to the user based on the parameters entered.

The valid box zone numbers are from 100 to 117, 200 to 210, 300 to 309, and 318 to 321. An if statement can check the validity of the box numbers, and if the number is invalid, a script message can be sent to the input dialog box.

The two most common script messages are setWarningMessage(), which displays a yellow triangle icon on the input dialog box, and setErrorMessage(), which displays a red X on the input dialog box. The error message will also stop the script from running until the error is resolved.

7. Scroll to the updateMessages module in the tool's class. Add an if statement to validate that the box number entered is within the acceptable range. Note that the range tool is not inclusive of the second value. Also add a setErrorMessage() command to stop the script if the number fails the validation, as shown:

```
def updateMessages(self, parameters):
    """Modify the messages created by internal validation for each tool
    parameter.  This method is called after internal validation."""
    if parameters[0].value in range(100,118):
        arcpy.AddMessage("Value is OK.")
    else:
        parameters[0].setErrorMessage(str("This number is out of range"))
    return
```

Your turn

The last set of code validates the box maps in District 1. Using the ranges stated previously, add the code to validate the other acceptable ranges and to provide an applicable error message, as shown:

```
def updateMessages(self, parameters):
    """Modify the messages created by internal validation for each tool
    parameter.  This method is called after internal validation."""
    if parameters[0].value in range(100,118):
        arcpy.AddMessage("Value is OK.")
    elif parameters[0].value in range(200,211):
        arcpy.AddMessage("Value is OK.")
    elif parameters[0].value in range(300,310):
        arcpy.AddMessage("Value is OK.")
    elif parameters[0].value in range(318,322):
        arcpy.AddMessage("Value is OK.")
    else:
        parameters[0].setErrorMessage(str("This number is out of range"))
    return
```

The entry and validation of the first parameter is complete. Next, add and configure a value list parameter for the building type. The entry portion of the parameter is basically the same as the first parameter. Give the user a list of choices using the descriptions by setting the filter type and the filter list. Then include code to set the return value based on the user's choice.

8. **Set up a second parameter (param1) with the display name** Select a Building Type **and the name** bldgType**. The data type is a string, and the keyword for that data type is found in ArcGIS for Desktop Help.**

```
•        param1 = arcpy.Parameter(
•        displayName="Select a Building Type",
•        name="bldgType",
•        datatype="GPString",
•        parameterType="Required",
•        direction="Input")
```

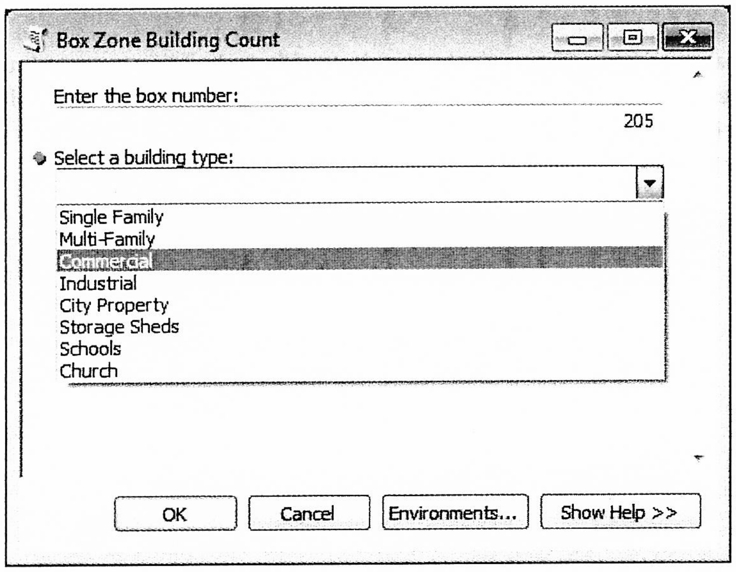
9. **Now add a filter to the parameter to make it a value list with the text descriptions of the building types, as shown:**

```
•        param1.filter.type = "ValueList"
•        param1.filter.list = ["Single Family","Multi-Family","Commercial","Industrial","City Property", \
•                              "Storage Sheds","Schools","Church"]
```

10. **After defining the two parameters, add them to the parameters list returned by the getParameterInfo method, as shown:**

```
•        params = [param0,param1]
•        return params
```

11. **Save and close the Python script. Right-click your MyExercises folder, click Refresh to ensure that the Python toolbox loads the most recent code, and then double-click the Box Zone Building Count tool to run it. Note the inputs and validations. Try several box numbers (press Tab after typing each one), and examine the drop-down lists, as shown. When you are finished, click OK or Cancel. The script has no code in the execute module, so nothing will happen, but you can see how the interface operates.**


Box Zone Building Count

Enter the box number:
205

Select a building type:

Single Family
Multi-Family
Commercial
Industrial
City Property
Storage Sheds
Schools
Church

Now add all the executable code to run the application.

12. Start editing the Python toolbox. Copy the code from the text file Tutorial 4-2 in the Data folder, starting with the line "# Get input from the user" through to the end. Paste the code to the execute module above the return statement, and correct the indent levels as necessary. Note the code to accept user input, as shown:

```
def execute(self, parameters, messages):
    """The source code of the tool."""
    # Import the modules
    import arcpy


    # Set up the environment
    arcpy.env.workspace = r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb"

    # Get input from the user
    #    The first will be the box number to act upon - index 0
    #    The second will be the building type to count - index 1
    boxNumber = arcpy.GetParameterAsText(0)
    buildingType = arcpy.GetParameterAsText(1)
```

The original script used arcpy.GetParameterAsText() to gain input from the user, but the Python toolbox uses parameters from the getParameterInfo method. To add the correct input for the executable code, change the arcpy.GetParameterAsText() code to parameters[] code, making sure to keep the same index numbers.

13. Change the data acceptance code to use the parameters from the toolbox script, as shown:

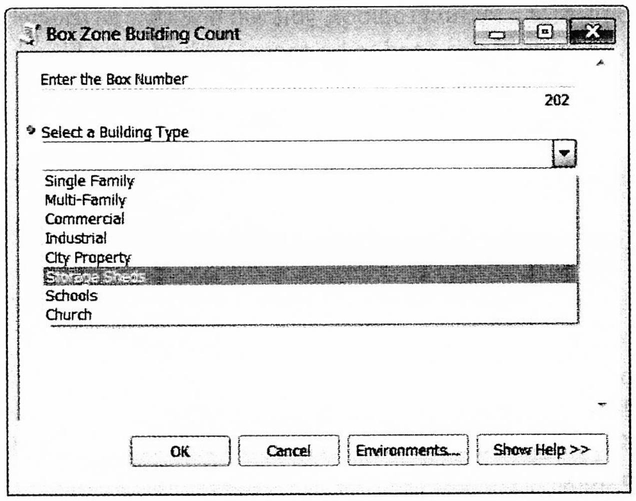
```
# Get input from the user
#    The first will be the box number to act upon - index 0
#    The second will be the building type to count - index 1
boxNumber = parameters[0].value
buildingType = parameters[1].value
```

There is one other change to make. The original script dealt with the building types as integers, and now they are being selected as strings. This problem is easily repaired by changing the values to integers based on their text description. However, when you make this change, you must cast this variable to a string before you can use it in any message.

14. In the executable code just above the line "# Make feature layers . . . ," add a set of if-elif-else statements to convert the building type description into an integer using the values from the list provided in the data description at the start of this tutorial, as shown:

```
# Convert the string descriptions into integers
# "Single Family","Multi-Family","Commercial","Industrial","City Property",
# "Storage Sheds","Schools","Church"
if buildingType == "Single Family":
    buildingType = 1
    arcpy.AddMessage("You selected Single Family.")
elif buildingType == "Multi-Family":
    buildingType = 2
    arcpy.AddMessage("You selected Multi-Family.")
elif buildingType == "Commercial":
    buildingType = 3
    arcpy.AddMessage("You selected Commercial.")
elif buildingType == "Industrial":
    buildingType = 4
    arcpy.AddMessage("You selected Industrial.")
elif buildingType == "City Property":
    buildingType = 5
    arcpy.AddMessage("You selected City Property.")
elif buildingType == "Storage Sheds":
    buildingType = 6
    arcpy.AddMessage("You selected Storage Sheds.")
elif buildingType == "Schools":
    buildingType = 7
    arcpy.AddMessage("You selected Schools.")
else:
    buildingType = 8
    arcpy.AddMessage("You selected Church.")
# Make feature layers from the user input
```

15. Save and close the script. Right-click your MyExercises folder, and click Refresh to update the code. Double-click the Box Zone Building Count tool to run it. Get a count of storage sheds in box 202, as shown:



Enter the Box Number
202

Select a Building Type

Single Family
Multi-Family
Commercial
Industrial
City Property
Storage Sheds
Schools
Church

Try this with other combinations of box numbers and building types. Afterward, inspect the attribute tables for the input files, and note the addition of the fields and counts to the tables.