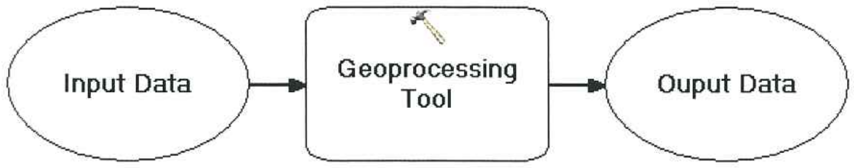
Chapter 2
Geoprocessing in ArcGIS

2.1 Introduction

This chapter introduces the ArcGIS geoprocessing framework, including the use of ArcToolbox, ModelBuilder, and Python. Experienced ArcGIS users will be familiar with most of this material, but a review is beneficial. Understanding the geoprocessing framework is helpful in writing effective geoprocessing scripts. Similarly, Python and ModelBuilder are often used in tandem, so a good knowledge of ModelBuilder is recommended to get the most out of Python scripting.

2.2 What is geoprocessing?

Geoprocessing in ArcGIS allows you to perform spatial analysis and modeling as well as automate GIS tasks. A typical geoprocessing tool takes input data (a feature class, raster, or table), performs a geoprocessing task, and produces output data as a result. ArcGIS contains hundreds of geoprocessing tools. Examples of geoprocessing tools include tools for creating a buffer, for adding a field to a table, and for geocoding a table of addresses.



Geoprocessing supports the automation of workflows by creating a sequence that combines a series of tools. The output of one tool effectively becomes the input of the next tool. Creating these automated workflows combining geoprocessing tools can be accomplished in ArcGIS through the use of models and scripts.

The geoprocessing framework in ArcGIS consists of a set of windows and dialog boxes that are used to organize and execute tools. The geoprocessing framework makes it easy to create, execute, manage, document, and share geoprocessing workflows. Geoprocessing includes a set of tools that operate on data. The basic geoprocessing framework comprises the following:

- A collection of tools, organized in toolboxes and toolsets
- Methods to find and execute tools, including the Search window, the Catalog window, and the ArcToolbox window
- Tool dialog boxes for specifying tool parameters and executing tools
- ModelBuilder for creating models that allow for the sequencing of tools
- A Python window for executing tools using Python
- A Results window that logs the geoprocessing tools being executed
- Methods for creating Python scripts and using them as tools

Each of these components is described in more detail in the sections that follow. A few characteristics of this geoprocessing framework make it possible to work with tools in a consistent yet flexible manner. These characteristics include the following:

- All tools can be accessed from their toolbox, which makes for a consistent procedure for accessing tools, models, and scripts.
- All tools are documented the same way, which allows for consistent cataloging and searching.
- All tools have a similar user interface (the dialog box) for specifying the tool parameters.
- Tools can be shared.

2.3 A note on ArcObjects

You may recall the term "ArcObjects" from chapter 1. This section briefly outlines what ArcObjects are and how they relate to geoprocessing in ArcGIS.

The ArcObjects library consists of basic programming objects Esri has created to develop ArcGIS software. ArcObjects are made available to application developers as part of the ArcObjects .NET Software Development Kit (SDK) and the ArcObjects Java SDK. Application developers can use ArcObjects to create new applications or to enhance the functionality of existing ArcGIS applications. Esri software developers themselves use ArcObjects to create most of the tools in ArcGIS as well as to build the geoprocessing framework.

ArcObjects is intended for use with a system programming language—that is, a language that can access system-level functions to implement complex logic and algorithms. ArcObjects consists of thousands of different objects, which give a programmer a good degree of control over what the application is going to look like and how it is going to work. Examples of system programming languages include C++ and .NET languages, which are some of the most common languages for working with ArcObjects. These languages require substantial programming knowledge, much more than is required for working with models and scripts.

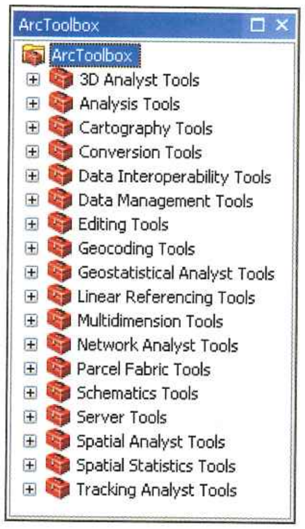
The ArcObjects SDKs and the geoprocessing framework are complementary, yet they accomplish different goals. ArcObjects is used to extend ArcGIS through new behaviors and to write stand-alone applications that build on the functionality of ArcGIS. Examples include creating new user interfaces and adding new behavior to feature classes. The geoprocessing framework is used to run existing tools and to create new tools (models and scripts) that automate tasks within the existing functionality of ArcGIS.

ArcGIS 10 introduced the desktop add-in model, which makes it possible to customize and extend ArcGIS for Desktop applications. In ArcGIS 10.1 for Desktop, Python was added to the list of languages that can be used to author desktop add-ins. Python add-ins can be used for some of the same things that were previously only possible using ArcObjects.

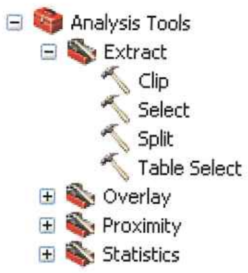
The emphasis in this book is using Python to create geoprocessing tools. ArcObjects is not used directly and is therefore not covered in this book. Desktop add-ins is also not covered. It should be noted that it is possible to work directly with ArcObjects using Python—it is a programming language, after all. However, the real strength of using Python lies in the ability to write powerful scripts, which requires a moderate level of programming skills and effort.

2.4 Using toolboxes and tools

Geoprocessing tools perform operations on datasets. Several hundred tools are available in ArcGIS. Exactly which tools you have available in ArcGIS depends on which product license you have (ArcGIS for Desktop Basic, ArcGIS for Desktop Standard, or ArcGIS for Desktop Advanced, formerly known as the ArcView license level, ArcEditor license level, and ArcInfo license level, respectively) and whether you have extensions installed (such as the ArcGIS 3D Analyst for Desktop extension, ArcGIS Network Analyst for Desktop extension, ArcGIS Spatial Analyst for Desktop extension, and others). The organization of the tools, however, remains the same. ➜


ArcToolbox
- ArcToolbox
  - 3D Analyst Tools
  - Analysis Tools
  - Cartography Tools
  - Conversion Tools
  - Data Interoperability Tools
  - Data Management Tools
  - Editing Tools
  - Geocoding Tools
  - Geostatistical Analyst Tools
  - Linear Referencing Tools
  - Multidimension Tools
  - Network Analyst Tools
  - Parcel Fabric Tools
  - Schematics Tools
  - Server Tools
  - Spatial Analyst Tools
  - Spatial Statistics Tools
  - Tracking Analyst Tools

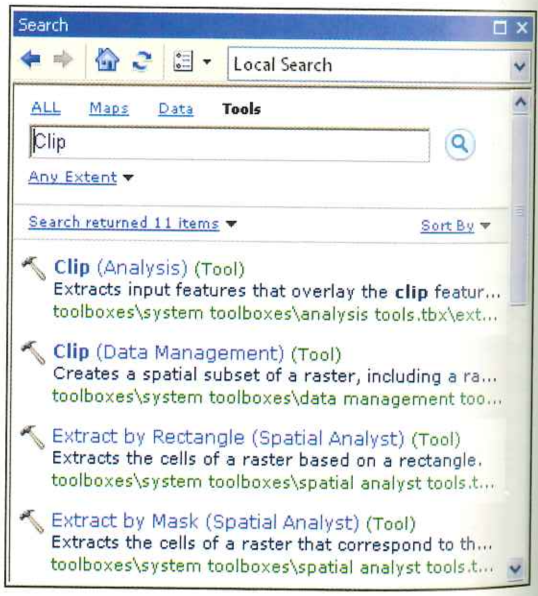
In ArcToolbox, geoprocessing tools are organized into toolboxes—for example Analysis Tools, Cartography Tools, and Conversion Tools, among others. Each toolbox typically contains one or more toolsets, and each toolset contains one or more tools. ➜


- Analysis Tools
  - Extract
    - Clip
    - Select
    - Split
    - Table Select
  - Overlay
  - Proximity
  - Statistics

There are several ways to find the tools you need:

- Some of the most commonly used tools can be accessed directly from the Geoprocessing menu in ArcGIS for Desktop applications. Only a handful of tools are listed there.

- Another approach is to search for tools. In the Search window, you can type a term and search your map documents, data files, and tools. You can also filter your results to search for tools only. ➜


Search

Local Search

ALL Maps Data **Tools**

Clip

Any Extent

Search returned 11 items   Sort By

Clip (Analysis) (Tool)
Extracts input features that overlay the **clip** featur...
toolboxes\system toolboxes\analysis tools.tbx\ext...

Clip (Data Management) (Tool)
Creates a spatial subset of a raster, including a ra...
toolboxes\system toolboxes\data management too...

Extract by Rectangle (Spatial Analyst) (Tool)
Extracts the cells of a raster based on a rectangle.
toolboxes\system toolboxes\spatial analyst tools.t...

Extract by Mask (Spatial Analyst) (Tool)
Extracts the cells of a raster that correspond to th...
toolboxes\system toolboxes\spatial analyst tools.t...

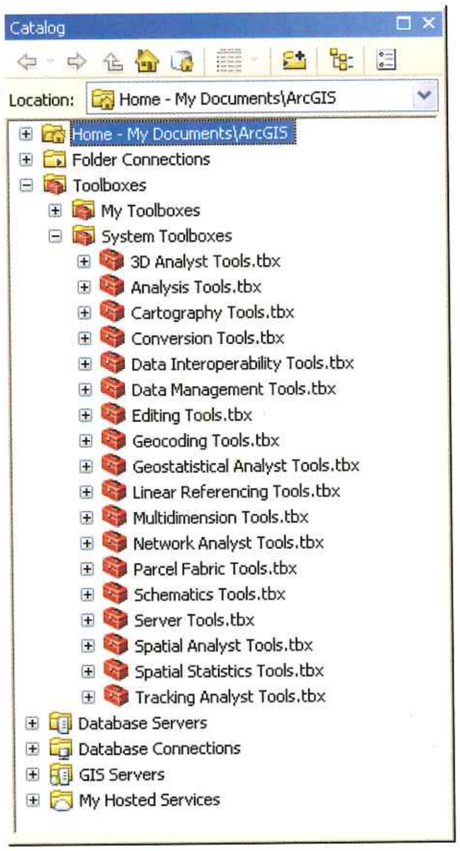
he **clip** featur...
tools.tbx\ext...

including a ra...
nagement too...

yst) (Tool)
n a rectangle.
nalyst tools.t...

(Tool)
espond to th...
nalyst tools.t...

- To see all the available tools, you can open ArcToolbox and browse through the toolboxes and toolsets until you find the tool you want. Similarly, you can browse through the tools in the Catalog window in the ArcMap application or in the Catalog Tree window in the ArcCatalog application. Given the number of tools available, browsing can be cumbersome if you don't know where to look. Once you gain experience using the tools, however, you will start remembering where the tools are that you use most frequently. ➔



```
⊞ Home - My Documents\ArcGIS
⊞ Folder Connections
⊟ Toolboxes
    ⊞ My Toolboxes
    ⊟ System Toolboxes
        ⊞ 3D Analyst Tools.tbx
        ⊞ Analysis Tools.tbx
        ⊞ Cartography Tools.tbx
        ⊞ Conversion Tools.tbx
        ⊞ Data Interoperability Tools.tbx
        ⊞ Data Management Tools.tbx
        ⊞ Editing Tools.tbx
        ⊞ Geocoding Tools.tbx
        ⊞ Geostatistical Analyst Tools.tbx
        ⊞ Linear Referencing Tools.tbx
        ⊞ Multidimension Tools.tbx
        ⊞ Network Analyst Tools.tbx
        ⊞ Parcel Fabric Tools.tbx
        ⊞ Schematics Tools.tbx
        ⊞ Server Tools.tbx
        ⊞ Spatial Analyst Tools.tbx
        ⊞ Spatial Statistics Tools.tbx
        ⊞ Tracking Analyst Tools.tbx
⊞ Database Servers
⊞ Database Connections
⊞ GIS Servers
⊞ My Hosted Services
```

Once you have found a tool, you can double-click it to open the tool dialog box and fill in its parameters.

2.5 Learning types and categories of tools

There are four types of tools in ArcGIS, and each is designated by a different symbol.

Built-in tools: These tools are built using ArcObjects and a compiled programming language such as C++ or the .NET languages. Esri creates these tools when authoring its software, and most of the tools in ArcGIS look like this.

Model tools: These tools are created using ModelBuilder. A number of tools in ArcGIS are model tools—for example, some of the rendering tools in the Spatial Statistics toolbox.

Script tools: These tools consist of scripts that are accessible using a tool interface. When a tool is executed, a script is run to carry out the geoprocessing operations—for example, a Python script (.py), an AML (Arc Macro Language) file (.aml), or an executable file (.bat or .exe). Most of the script tools in ArcGIS use Python.

Specialized tools: These tools are created by system developers. They can have their own unique interface that is different from a regular tool dialog box. These tools are not very common, although third-party developers may distribute their tools in this manner.

Although these tools are created using different methods, the tool dialog boxes for different types of tools all look the same.

There are also two categories of tools:

1. *System tools:* These are the tools that are created by Esri and installed as part of the regular ArcGIS software. Exactly which system tools are installed depends on the product license level and the number of extensions. Most system tools are built-in tools, but a number of script and model tools are also authored by Esri.

2. *Custom tools:* These tools commonly consist of script and model tools, but built-in and specialized tools are also included. Custom tools can be created by a user, but they can also be obtained from a third party, and then added to ArcGIS.

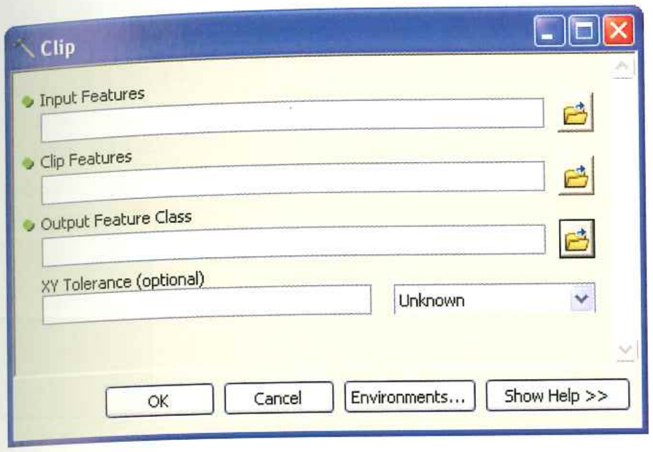
When using geoprocessing tools, you may not notice which tools are system tools and which ones are custom tools because they are designed to work the same way. Once a custom tool is created, it can be added to a geoprocessing workflow the same way as any of the system tools.

2.6 Running tools using tool dialog boxes

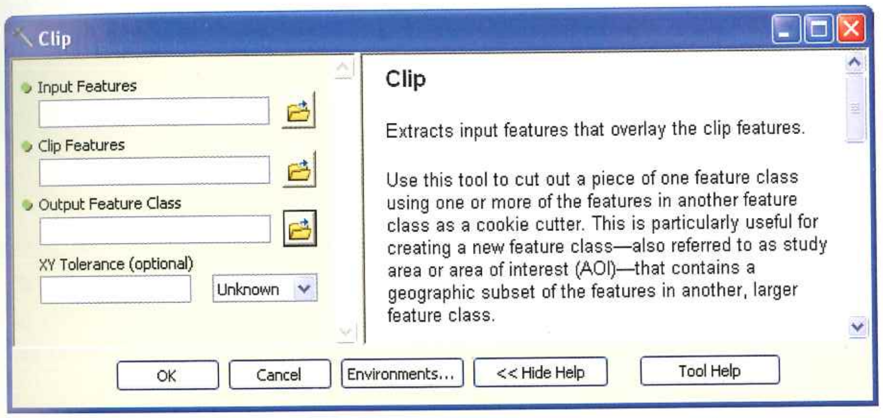
When you find a tool, you can open it by double-clicking it, which brings up the tool dialog box. Each tool has a number of parameters that need to be specified before the tool can be run. A tool parameter is a text string, number, or other entry that tells the tool how it should be executed. The tool dialog box provides an easy-to-use interface for specifying these parameters. This includes browsing for and selecting datasets, selecting options from a list, and entering values.

Most tools require one or more input datasets. Other common parameters are preset text strings called keywords. Although each tool has one or more parameters, not all parameters are required. Optional parameters have default values that are set with the tool. You can accept the default values simply by not changing the parameters or by not specifying a value. Default values for keywords are typically shown on the tool dialog box when it is first opened.

An example of the tool dialog box for the Clip tool is shown in the figure.



Every tool dialog box has a Help panel on the right side that provides useful information about the tool. You can switch the visibility of the Help panel by using the Show Help Show Help >> and Hide Help << Hide Help buttons at the bottom of the tool dialog box. To get a more complete description of the tool, you can click the Tool Help Tool Help button to access the tool's Help page. You will look at the Help pages in more detail in later chapters for examples of how to use a tool in a Python script.



Clip

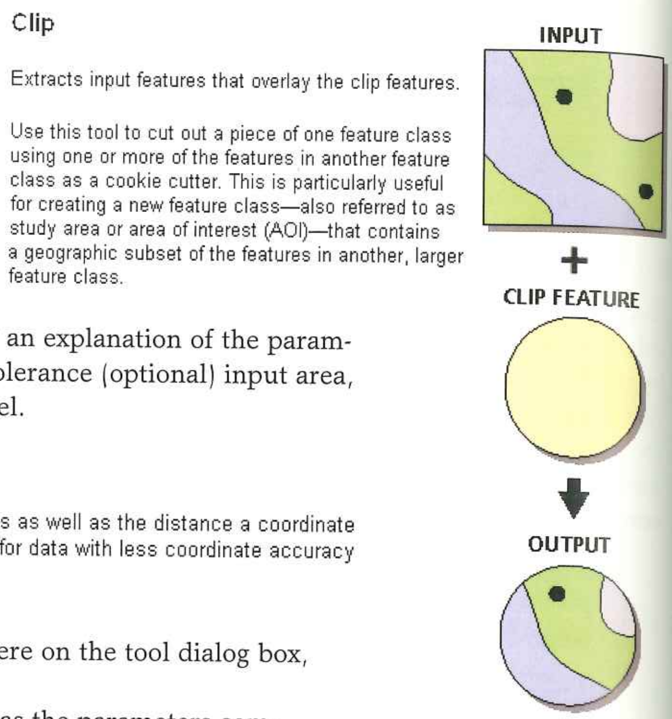
Extracts input features that overlay the clip features.

Use this tool to cut out a piece of one feature class using one or more of the features in another feature class as a cookie cutter. This is particularly useful for creating a new feature class—also referred to as study area or area of interest (AOI)—that contains a geographic subset of the features in another, larger feature class.

The tool dialog box contains the parameters that need to be specified for the tool to run. Notice that there are a total of four parameters in the case of the Clip tool. Three of them are flagged by a small green dot, which indicates the parameter is required and needs a value, and thus there is no default value. These are the Input Features to be clipped, the Clip Features, and the Output Feature Class to store the result. The XY Tolerance parameter is optional.

The tool dialog box has several mechanisms for ensuring proper inputs. For example, you could type the path and file name for Input Features (for example, C:\Data\streams.shp), but you could just as easily end up with a typo. Instead of typing a path and file name, you can click the drop-down arrow ▼ to select from a list of the layers in the table of contents in your current ArcMap document. This arrow is shown only when there are acceptable feature layers in your map document to choose from. You can also use the Browse button to browse to data on disk. These options not only prevent typos, but also check for valid data input. For example, for the Clip tool, the Clip Features parameter has to consist of a polygon feature class, so the selection and browsing options will show only the available polygon feature classes.

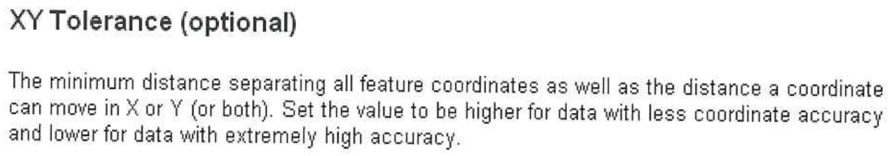
Another feature of the tool dialog box is that the contents of the Help panel change depending on where your pointer is. When you first open the tool, a description of the tool appears in the Help panel. If you are not familiar with the tool, this description is useful for ensuring you have the correct tool. ➔


Clip

Extracts input features that overlay the clip features.

Use this tool to cut out a piece of one feature class using one or more of the features in another feature class as a cookie cutter. This is particularly useful for creating a new feature class—also referred to as study area or area of interest (AOI)—that contains a geographic subset of the features in another, larger feature class.

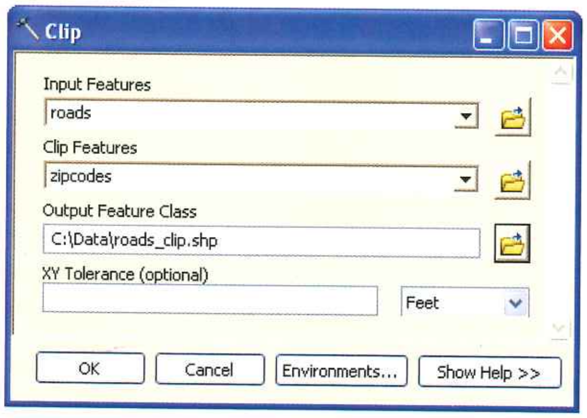
Once you click inside the input area for a particular parameter on the tool dialog box, the content of the Help panel changes to show an explanation of the parameter. For example, when you click in the XY Tolerance (optional) input area, a brief description is provided in the Help panel.


XY Tolerance (optional)

The minimum distance separating all feature coordinates as well as the distance a coordinate can move in X or Y (or both). Set the value to be higher for data with less coordinate accuracy and lower for data with extremely high accuracy.

To get back to the overview Help, click anywhere on the tool dialog box, but not inside any parameter input areas.

Now consider an example dialog box that has the parameters completed. The input feature class is a shapefile (.shp) called roads.shp and is being clipped by a shapefile called zipcodes.shp. The output feature class is a shapefile called roads_clip.shp. The XY tolerance is left blank, which means the default value is used (which is 0.001 meters or its equivalent in map units for this parameter).


Clip

Input Features
roads

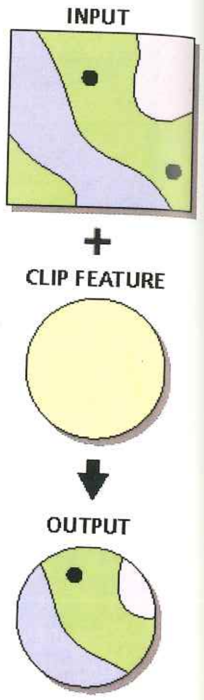
Clip Features
zipcodes

Output Feature Class
C:\Data\roads_clip.shp

XY Tolerance (optional)

Feet

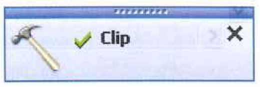
Note: The full path to the output features is shown because the feature class was selected from disk. When selecting layers in the current ArcMap document using a drop-down list, only the name of a feature layer is shown and not its full path. In the latter case, the file extension, such as .shp, is not shown because the parameter is specified as a feature layer, not as a feature class on disk.



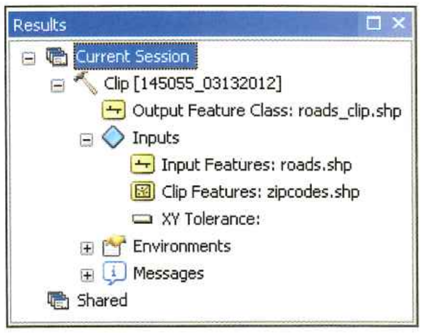
res is shown because
. When selecting
using a drop-down
hown and not its
nsion, such as .shp,
pecified as a feature

When you click OK, the Clip tool runs. At the bottom of the ArcMap interface, a status bar displays the name of the tool that is being executed. By default, tool execution occurs in the background. This means you can continue working in ArcMap as the tool runs.

When the tool is finished running, a pop-up notification appears in the notification area, at the far-right corner of the taskbar. ➔



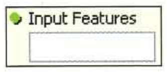
When a tool is finished running, the resulting feature class is added by default as a layer to the ArcMap table of contents (when the tool is run from within ArcMap). An entry is also posted to the Results window (on the menu bar, click Geoprocessing > Results). This entry includes all the input and output parameters, as well as tool execution messages. ➔



```
Results
⊟ Current Session
  ⊟ Clip [145055_03132012]
      Output Feature Class: roads_clip.shp
    ⊟ Inputs
        Input Features: roads.shp
        Clip Features: zipcodes.shp
        XY Tolerance:
    ⊞ Environments
    ⊞ Messages
  Shared
```

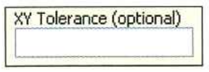
The entries in the Results window can be valuable in a number of ways. First, you can review the parameters that were used to run a particular tool. Second, you can run the same tool again directly from the Results window. The tool dialog box will be populated with the same parameters as before, and you can use these same parameters or change selected ones. Finally, you can review any error messages in the Results window.

Take a moment to review the tool parameters on a tool dialog box. Parameters that are required and need a value on the tool dialog box have a small green dot next to them. ➔

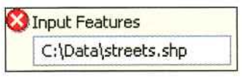


You can click the green dot to see more detailed information about the required parameter.

Optional parameters have no icon in front of them, and if they are left blank, the default values will be used when the tool runs. ➔

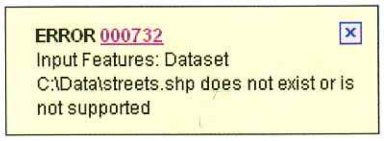


If an incorrect parameter is specified, an error warning appears. ➔



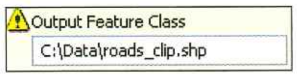
Input Features
C:\Data\streets.shp

Pausing your pointer over the icon shows a brief description, while clicking the icon brings up a more detailed error message. In this case, the input dataset does not exist. ➔



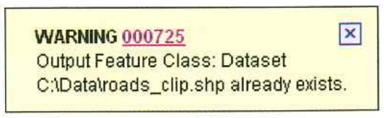
ERROR 000732
Input Features: Dataset C:\Data\streets.shp does not exist or is not supported

Sometimes a warning message appears, indicating that running the tool may lead to undesired results. ➔



Output Feature Class
C:\Data\roads_clip.shp

A warning message does not prevent the tool from running, but it may warrant taking a look at the warning prior to running the tool. In this case, the output dataset already exists and will be overwritten if the tool is executed. ➔

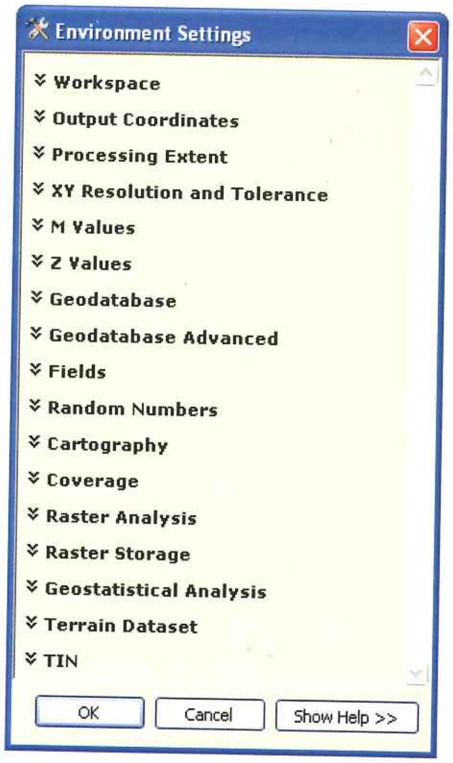


WARNING 000725
Output Feature Class: Dataset C:\Data\roads_clip.shp already exists.

Note: Overwriting geoprocessing results is an option under Geoprocessing > Geoprocessing Options. The default is turned off, meaning an attempt to overwrite existing datasets will result in an error. When the option is turned on, only a warning message is provided and the tool will run, overwriting the existing datasets.

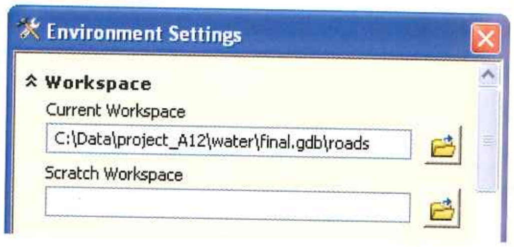
2.7 Specifying environment settings

Geoprocessing operations are influenced by *environment settings*. These settings are like additional hidden parameters that affect how a tool is run. The Environment Settings dialog box (Geoprocessing > Environments) allows you to view and set the environments for geoprocessing. ➔



There are a number of settings, but one of the most important is the current workspace. Most geoprocessing tools use datasets as inputs, and then output new datasets. A workspace consists of a path to where these datasets are located. Complete path and file names can get quite long—for example, C:\Data\project_A12\water\final.gdb\roads\streets. To avoid having to type these lengthy names every single time (and possibly introduce typos), you can use the tool dialog box to select layers from the current ArcMap document or to browse to the location of a dataset. You can also drag files from the ArcMap table of contents to your map. In addition, a workspace can be set to make specifying input and output datasets easier. After your workspace is set, you need to specify only the base name. In the preceding example, you would set the workspace to C:\Data\project_A12\water\final.gdb\roads, and then enter only the base name "streets" when specifying tool parameters.

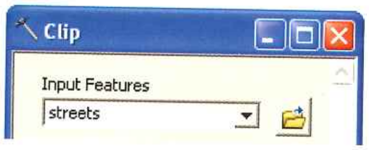
For example, how you would set the current workspace is shown in the figure. ➔



Current Workspace: C:\Data\project_A12\water\final.gdb\roads

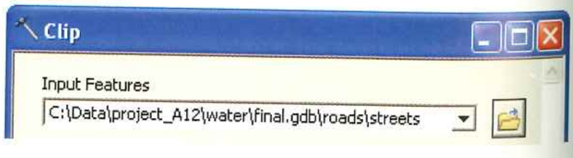
Scratch Workspace:

On a tool dialog box then, you could specify a feature class inside this workspace by typing only its base name. ➔



Input Features: streets

When you click anywhere else on the tool dialog box, the parameter is automatically completed using the current workspace, as shown in the figure. ➔



Input Features: C:\Data\project_A12\water\final.gdb\roads\streets

Output datasets are also created by default in the current workspace.

There are two types of workspaces: (1) the current workspace, which specifies, by default, where inputs are taken from and where outputs are placed; and (2) the scratch workspace, which is primarily used by model tools to write intermediate data.

There are also settings for specific data types (such as a geodatabase, raster, or TIN (triangulated irregular network)) and for specific types of functions (such as random numbers). Typically, you need to set only a few of these environments for a particular workflow because many of them do not apply to the data and tools you are using.

Environments are always at work. In other words, even if you don't specify them, they have default values that are used when a tool is run. For example, the default Output Coordinate System is the same one as the input. So when you are running a tool, the coordinate system is not changed, unless you specify otherwise on the Environment Settings dialog box.

Environment settings can be specified at a number of different levels, and there is a specific hierarchy to this process:

1. The first level is the application. You can right-click in the ArcToolbox window and click Environments. This brings up the Environment Settings dialog box. Any settings created here are passed to the tools that are called by the application.

2. The second level is the individual tool. Every tool dialog box has an Environments button Environments... . When you click the button, the Environment Settings dialog box opens. Any settings created here are applied only to the current running of the tool, and these settings override the settings passed by the application. These settings are not saved to the tool but apply only to a single execution of the tool.

3. The third level is a model. Environment settings can be created as part of the model properties, which is separate from the settings you create on the tool dialog box. Any settings created in the model override the settings passed by the application or the tool dialog box settings. Model environment settings are saved as part of the model properties.

4. The fourth and final level is a script. Environment settings can be coded into a Python script and these settings override the settings passed by the application or the tool dialog box. These settings are saved as part of the Python code in the script file.

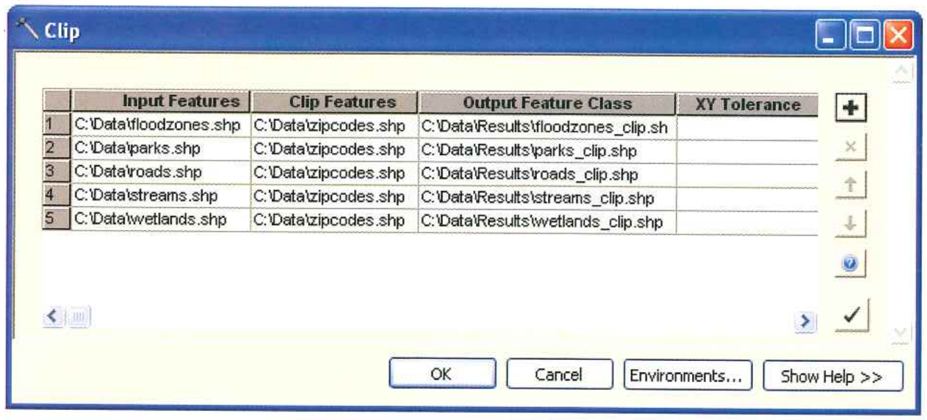
In general, environment settings are passed down in this hierarchical system, but you can override these settings at each level.

2.8 Using batch processing

Most tools use a limited number of input datasets. For example, the Clip tool uses only a single input feature class and clips it using a single clip feature class. What if you wanted to run the same tool using similar settings on many different input datasets? This is where batch processing comes in. In the context of geoprocessing in ArcGIS, batch processing means executing a single tool multiple times using different inputs without further intervention.

All geoprocessing tools can be run in batch mode. Right-clicking a tool and clicking Batch brings up the Batch window of the particular tool. The Batch window shows a grid of rows and columns. The columns are the parameters of the tool and each row represents one execution of the tool. Rows can be added and the tool parameters can be specified for each run.

For example, in the case of the Clip tool, the Batch window looks like the example in the figure.

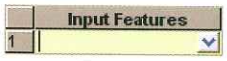


	Input Features	Clip Features	Output Feature Class	XY Tolerance
1	C:\Data\floodzones.shp	C:\Data\zipcodes.shp	C:\Data\Results\floodzones_clip.sh	
2	C:\Data\parks.shp	C:\Data\zipcodes.shp	C:\Data\Results\parks_clip.shp	
3	C:\Data\roads.shp	C:\Data\zipcodes.shp	C:\Data\Results\roads_clip.shp	
4	C:\Data\streams.shp	C:\Data\zipcodes.shp	C:\Data\Results\streams_clip.shp	
5	C:\Data\wetlands.shp	C:\Data\zipcodes.shp	C:\Data\Results\wetlands_clip.shp	

The batch grid contains five rows, indicating the Clip tool will run five times based on different inputs. The cells in each row represent the tool parameters. The buttons on the right allow for adding and deleting rows, changing the order of the rows, and examining the cells in the batch grid for valid values.

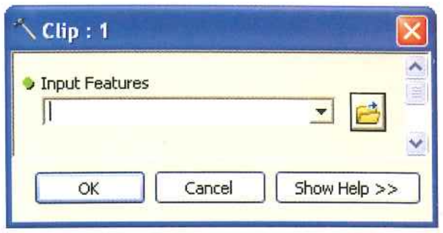
Entering parameters for the batch grid is similar to working with a regular tool dialog box, as follows:

- Click inside a cell and a drop-down arrow appears. This allows you to select from the layers in the table of contents of the current ArcMap document.



- You can drag layers from the table of contents into the dialog box.

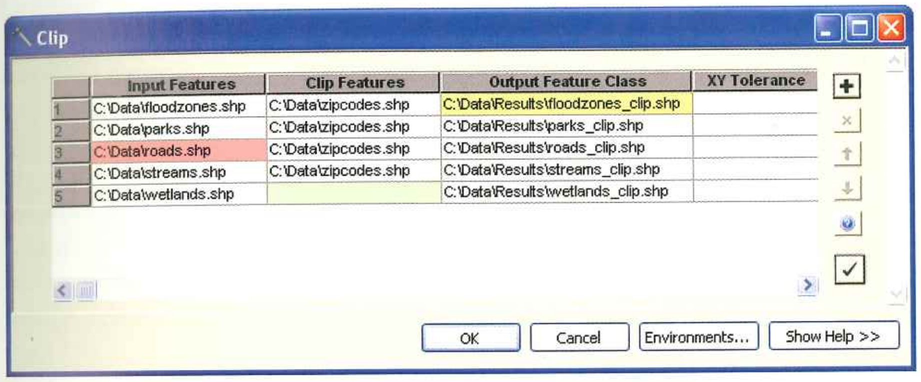
- Right-click in a cell and click Open. This brings up a separate dialog box that has the familiar drop-down arrow and Browse option. ➔



- Right-click in a cell and click Browse. This is a shortcut to the Browse option.

In addition to completing the batch grid cell by cell, there are a number of ways to quickly enter the parameter values for multiple cells, including (1) copying and pasting cell values and (2) using the Fill option to fill cells below a clicked cell with that cell's value.

One important feature of the Batch window is the Check Values button ✓. When you enter values into the batch grid, there is no automatic error checking. This is in contrast to regular tool dialog boxes, which automatically validate parameters—for example, checking whether a particular input dataset exists. When the Check Values button is clicked, all rows are scanned for errors and output dataset names are created if needed. If errors are found, the color of the cells changes.



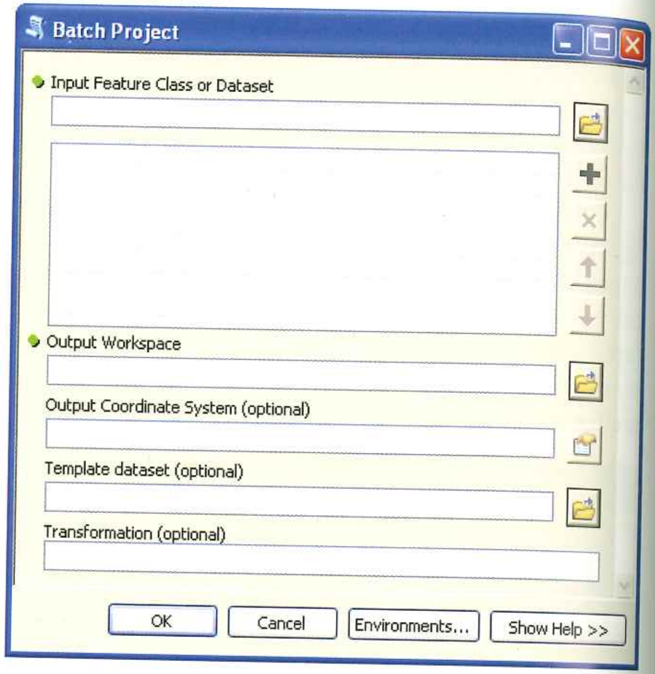
	Input Features	Clip Features	Output Feature Class	XY Tolerance
1	C:\Data\floodzones.shp	C:\Data\zipcodes.shp	C:\Data\Results\floodzones_clip.shp	
2	C:\Data\parks.shp	C:\Data\zipcodes.shp	C:\Data\Results\parks_clip.shp	
3	C:\Data\roads.shp	C:\Data\zipcodes.shp	C:\Data\Results\roads_clip.shp	
4	C:\Data\streams.shp	C:\Data\zipcodes.shp	C:\Data\Results\streams_clip.shp	
5	C:\Data\wetlands.shp		C:\Data\Results\wetlands_clip.shp	

Some common errors include the following:

- Green cells mean that a required parameter has not been specified.
- Red cells indicate an error was found and the tool will not run. The most common reason is that the input dataset does not exist.
- Yellow cells indicate a warning. The most common reason is that the output may not be what you expect.

Several other cell colors are also possible. White cells mean the parameters are correct. Gray cells mean the parameter is unavailable because it has no use, given the values of the other parameters. Finally, blue cells indicate a row is selected.

Running a tool in batch mode can be useful for setting up a large number of geoprocessing tasks. Although filling out the batch grid takes time, once the cell values are filled in, the tool runs in batch mode multiple times without additional user input. Running a tool in batch mode, however, does not reduce the time it takes for a tool to run. For example, running the Clip tool in batch mode using 20 rows takes the same amount of time as running the stand-alone Clip tool 20 times with the identical parameters. Time is saved by the quicker setup, not by faster tool execution.


Batch Project
Input Feature Class or Dataset
Output Workspace
Output Coordinate System (optional)
Template dataset (optional)
Transformation (optional)
OK
Cancel
Environments...
Show Help >>

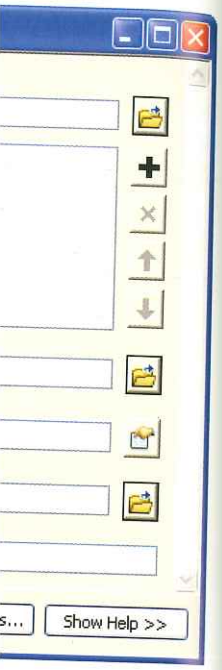
In addition to batch mode, there are a handful of specific batch system tools. For example, the Data Management toolbox contains the Project tool, which creates a new feature class with a different coordinate system from the input feature class. The Project tool uses only a single input feature class. The Batch Project tool, on the other hand, is the batch version of this tool and allows for multiple input feature classes. The same can be accomplished by running the Project tool in batch mode, but the parameter controls vary slightly. ➔

Both models and scripts provide additional ways to run batch processing, which is discussed in later chapters.

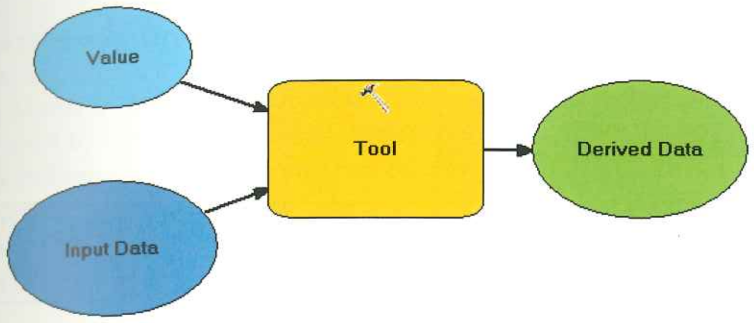
2.9 Using models and ModelBuilder

The execution of single tools is a practical way to accomplish certain GIS tasks. In a typical GIS workflow, however, you'll often need to run a sequence of tools to obtain the desired result. You could simply run through the sequence by running one tool at a time, but this has limitations, especially if your workflows are long and repetitive. ModelBuilder is one approach to creating this sequence of tools, whereby the output of one tool becomes the input to another tool. ModelBuilder is like a visual programming language—rather than using text-based instructions, it uses a visual flowchart to sequence geoprocessing tasks. A model in this context is a visual representation of a sequence of geoprocessing tasks. Within ArcGIS, models are tools, and once they are created, they work just like any other ArcGIS tool.

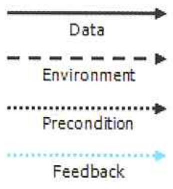
You can use any system or custom tool in your model, and there is no limit to how many tools you can use in a single model. Models can also include other models (since models are tools), and you can use iteration loops and conditions to control the flow of a model.



Before looking at how to create and run a model, first familiarize yourself with the basic elements of a model. Elements are the building blocks of a model. There are several types of elements: tools, data variables, value variables, and connectors.



Geoprocessing *tools* are the basic building blocks of a model. Tools perform geoprocessing operations on geographic data. *Data variables* reference data on disk or a layer in the ArcMap table of contents. *Value variables* are items such as strings, numbers, Boolean values (True or False), spatial references, linear units, and extents. In short, value variables contain anything but references to data on disk. Variables are used as the input and output parameters of tools. Derived data, or the output variable of one tool, becomes the input variable of another tool. Data and values are connected to tools by *connectors*. The connector arrows show the direction of geoprocessing tasks. There are four types of connectors: (1) data connectors, which connect data and value variables to tools; (2) environment connectors, which connect a variable containing an environment setting to a tool; (3) precondition connectors, which connect a variable to a tool; and (4) feedback connectors, which connect the output of a tool back into the same tool as input. ➔

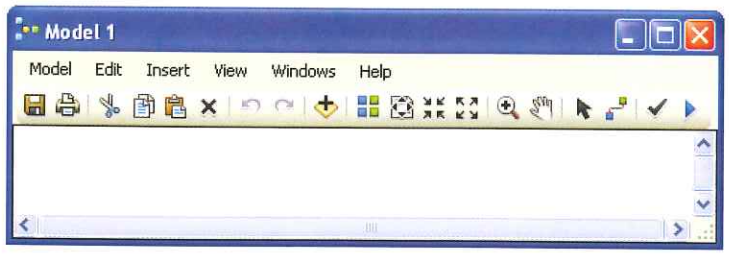


Connectors create model processes. A model process comprises a tool and the variables connected to the tool. The connector arrows specify the sequence of processing. A typical model contains a number of processes connected together. Complex models can contain hundreds of processes.

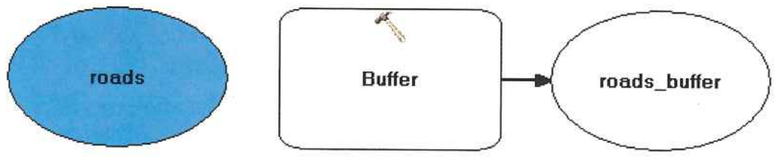
Creating a model and running tools in ModelBuilder consists of a number of steps:

1. Create a new model.
2. Add data and tools to the model.
3. Create connectors and fill tool parameters.
4. Save the model.
5. Run the model.
6. Examine the model results.

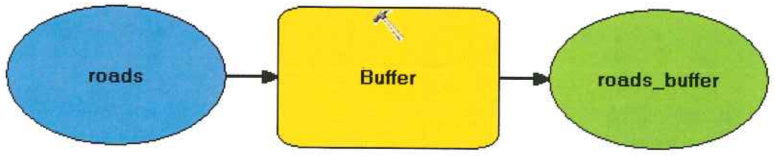
Create a new model: There are two main ways to create a new model: (1) on the Standard toolbar in an ArcGIS for Desktop application, you can click the ModelBuilder button ; or (2) you can right-click a toolbox or toolset in ArcToolbox, and then click New > Model. This creates a new blank model in ModelBuilder.



Add data and tools to the model: You can add data and tools to the model either by dragging them from the ArcMap table of contents or ArcToolbox into the Model window or by using the Add Data or Tool button on the Model toolbar. In the model in the figure, a feature class called "roads" has been added as a data element and the Buffer tool has been added as a tool. Because the output of the Buffer tool is a new feature class, it has been automatically added as a derived-data element.

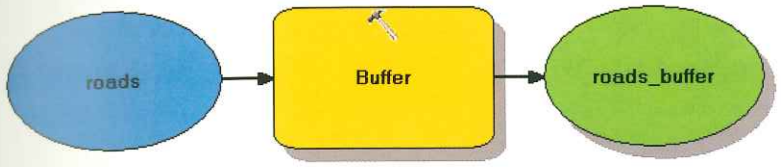


Create connectors and fill tool parameters: When you initially drag tools and data to a model, the process is not ready to run yet, because the required parameter values have not been specified. When any part of a process appears white in the model, it means that parameters are still missing. Parameters can be specified by opening the tool dialog box for each tool, and then specifying tool parameters as you would for any tool. Setting parameters creates connecting arrows, or connectors, between datasets and tools. You can also create these connectors using the Connect button , which lets you select the specific parameter to be set for a particular tool. Once all the required parameters for a process are set, all the model process elements are turned into a specific color to show they're ready to run.



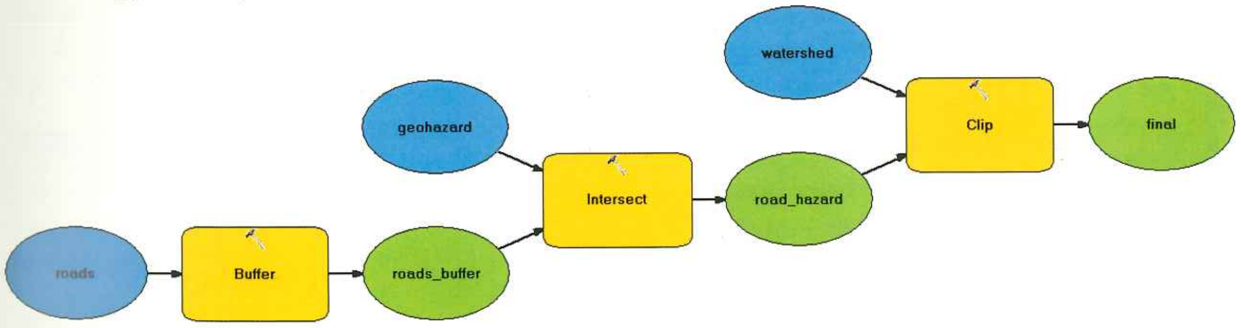
Save the model: You can save your model by clicking the Save button or by clicking Model > Save on the Model toolbar. Model properties, including the name of the model and its display name, can be set by clicking Model > Model Properties.

Run the model: Once all the parameters are specified, the model is ready to run. You can run the entire model by clicking the Run button or by clicking Model > Run on the Model toolbar. You can also run a specific process by itself by right-clicking a tool and clicking Run. A Model progress dialog box indicates the progress made in running the tools included in the model. When the model run is completed, the model elements (other than the data inputs) have drop shadows to indicate the tools have been run and the output datasets have been created.



Examine the model results: By default, output datasets created by a model are considered intermediate, which means they are saved on disk but not automatically added to the ArcMap table of contents. To examine the model result, you can right-click the model element that contains the output dataset and click Add To Display. This adds the dataset to the ArcMap table of contents so you can examine the result.

Additional tools and data can be added to the model using the same steps. In the example model in the figure, the road layer is buffered and then intersected with a layer of geological hazard areas. The intersect result is then clipped using a watershed layer.



A model in ModelBuilder serves as a visual flowchart of a sequence of geoprocessing tools. The ModelBuilder interface provides an intuitive way to create this sequence. The key is that a model is a tool in a toolbox, which makes it possible to save the model for future use and to share it with others.

The model developed so far is relatively simple. However, there is no limit to the number of datasets and tools that can be used in a single model. Sophisticated models can contain a large number of geoprocessing tasks.

Among the advantages of using models to create geoprocessing workflows:

- ModelBuilder provides an intuitive interface to create workflows.
- Models provide an efficient mechanism to document workflows.
- Models can be organized in toolboxes and shared with others.

There are many more details to learn about ModelBuilder, which is beyond the scope of this book. ArcGIS Desktop Help provides extensive documentation on ModelBuilder. From within an ArcGIS for Desktop application, click Help > ArcGIS Desktop Help > Professional Library > Geoprocessing > Geoprocessing with ModelBuilder.

2.10 Using scripting

Just as ModelBuilder can be used to create models that run a sequence of tools, a scripting language can be used to create and run this sequence. Scripting languages are relatively easy to learn, and the primary scripting language used in ArcGIS is Python.

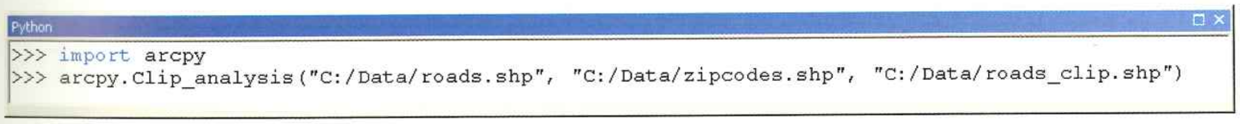
Scripts are analogous to models: ModelBuilder is used to create models, and Python is used to create scripts. ModelBuilder is a visual programming language, and Python is a text-based programming language. And just as models are tools within ArcGIS, scripts are also tools. So once a script is created, it becomes another tool you can run on its own or use in a model or in another script. Scripts can be run as stand-alone scripts on disk, in which case they are not a tool, but it is relatively easy to add a script as a script tool to a toolbox. Models can also be converted to scripts, but not vice versa. Converting a model to a script is covered in a later section of this chapter.

If models and scripts are so similar, why use a script instead of a model? ModelBuilder is a very intuitive way to create tools and relatively easy to learn for the beginning ArcGIS user. It requires no programming experience and there is no syntax to learn. Many geoprocessing workflows can be accomplished using models created in ModelBuilder. These models can be shared and modified. ModelBuilder, however, has certain limitations, and some of the more complex geoprocessing operations cannot be accomplished by a model alone. Some specific things you can do with a script that are not possible with a model include the following:

- Some lower-level geoprocessing tasks are possible only in scripts. For example, script cursors let you loop through records in a table, reading the existing rows and inserting new rows.
- Scripting allows for more advanced programming logic, such as advanced error handling and the use of more advanced data structures. Many scripting languages, including Python, have been extended with additional libraries offering more advanced functions.
- Scripting can be used to wrap other software—that is, to glue together applications. This facilitates the integration of various software applications. For example, Python can be used to access functions in Microsoft Excel or in the statistical package R.
- A script can be run as a stand-alone script on disk outside of ArcGIS. In most cases, you still need to have ArcGIS installed on the computer, but ArcMap or ArcCatalog do not need to be running for the script to work.
- Stand-alone scripts can be scheduled to run at a specific time without user intervention.

Python scripts can be created and run using a Python editor, such as PythonWin. You can also run Python code in the Python window of ArcGIS for Desktop. The Python window works like an interactive interpreter and code is executed immediately. The functionality of the Python window is discussed in more detail in chapter 3.

To run a tool in Python, type the tool name followed by its parameters. For example, the Python code shown in the figure runs the Clip tool.



```
>>> import arcpy
>>> arcpy.Clip_analysis("C:/Data/roads.shp", "C:/Data/zipcodes.shp", "C:/Data/roads_clip.shp")
```

The result, which follows, is printed in the Python window and the resulting shapefile is added to the ArcMap table of contents.

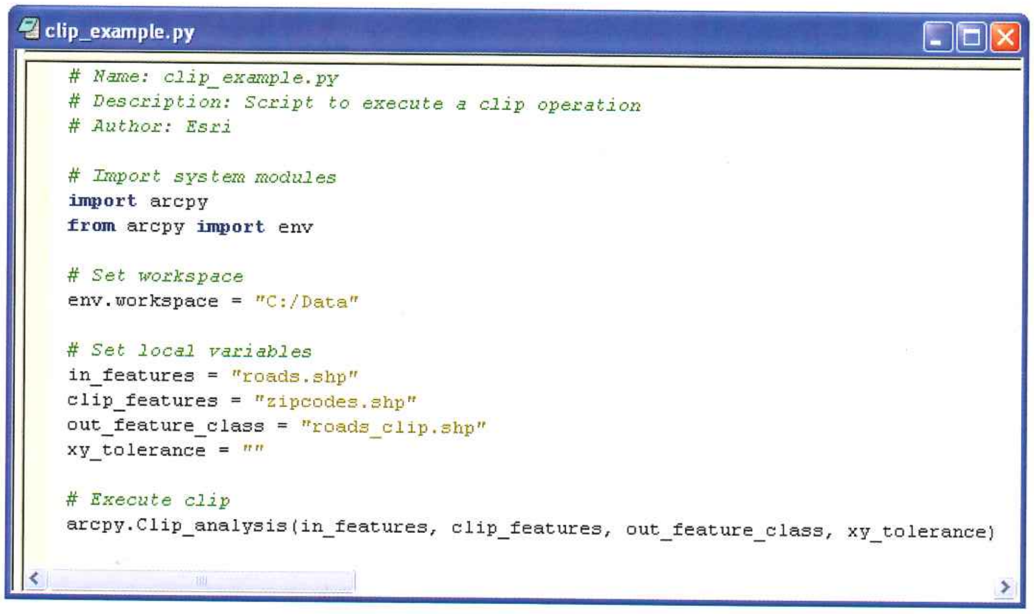
```
<Result 'C:\\Data\\roads_clip.shp'>
```

Notice that the Python code in the figure (see preceding page) uses "arcpy" in the code. This is the ArcPy site package. The first line of code is `import arcpy`, which makes it possible to access all ArcGIS geoprocessing tools and other functionality in Python. ArcPy is described in detail in chapter 5. Don't worry too much about the exact syntax of the code for now. The basic idea is that you run a tool by typing the tool name followed by its parameters.

> ***Note:*** *ArcGIS 9 contained a Command Line window, which also allowed for the running of a tool by typing the tool name followed by the tool parameters. The text you entered was called a "command." The syntax of these commands, however, was specific to the ArcGIS environment—in fact, it relied heavily on the syntax of the old-style ArcInfo command line. You could not use Python in the Command Line window. In ArcGIS 10, the Command Line window has been replaced by the Python window. Those working with Python typically refer to "code" rather than "commands," but occasionally you will see references to Python "commands."*

Python code can be entered in the Python window and run immediately. You can also use a text editor or a Python editor to create and run Python files on disk. Python files have a .py extension and are known as scripts. Scripts are programs that can be run from the operating system, from a Python editor, or by using a script tool that runs the script. Following is an example of each.

The code contained in a script called clip_example.py is shown in the PythonWin editor. This is a modified version of the script that is provided within the Help page for the Clip tool. Again, don't worry too much about the exact syntax for now.

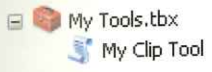


```
# Name: clip_example.py
# Description: Script to execute a clip operation
# Author: Esri

# Import system modules
import arcpy
from arcpy import env

# Set workspace
env.workspace = "C:/Data"

# Set local variables
in_features = "roads.shp"
clip_features = "zipcodes.shp"
out_feature_class = "roads_clip.shp"
xy_tolerance = ""

# Execute clip
arcpy.Clip_analysis(in_features, clip_features, out_feature_class, xy_tolerance)
```

You can navigate to the location of this script and double-click the file to run it. You do not need to have an ArcGIS for Desktop application open and you don't need to open the script in a Python editor. You can confirm the results of the script execution by examining the data in an ArcGIS for Desktop application. There are several benefits to running a script directly—most notably, you can set a script to run at a specific time without user intervention.

Another way to run a script is to use a Python editor like PythonWin. You can open a script in the editor, verify its content, and then execute the script. Similar to running a script directly from the operating system, you do not need to have an ArcGIS for Desktop application open for the script to run, although you need to have ArcGIS installed on your computer to be able to use the geoprocessing functions. One of the benefits of running a script using a Python editor is that messages are printed to the interactive window, including any error messages.

The third way to run a script is to create a script tool that runs the script. For example, you can create your own toolbox, create a new script tool (for example, My Clip Tool), and then add the clip_example.py script to this tool.

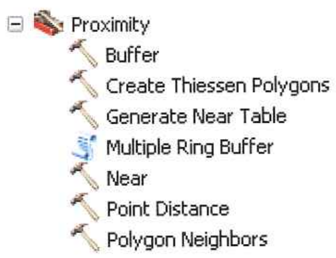


You can then run the script as you would any other geoprocessing tool. The benefit of running a script as a script tool from within ArcGIS is that you can integrate the script tool with other tools and models. The tool has its own dialog box, and the tool can be added to a model in ModelBuilder or called by another script.

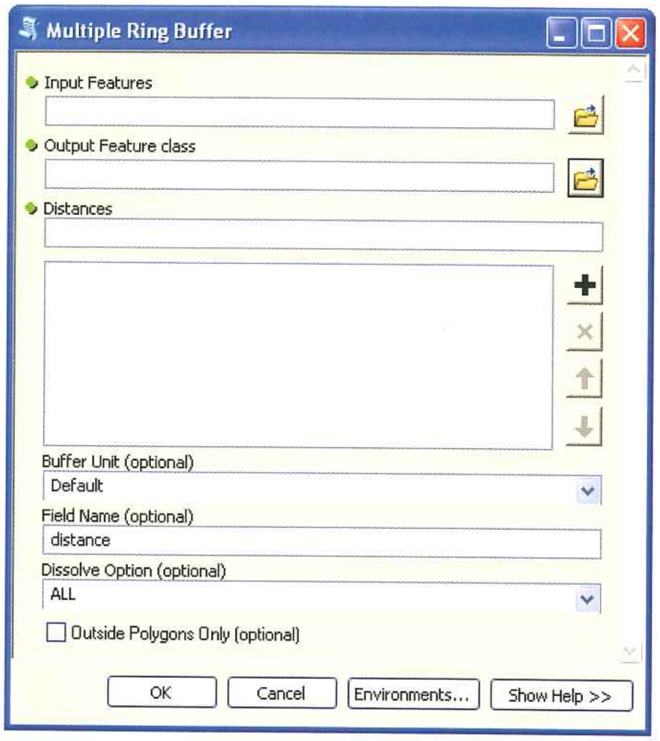
The script used here so far is relatively simple and in fact does nothing more than the regular Clip tool. However, it is relatively easy to create scripts whose functionality exceeds that of existing tools—these scripts are covered in later chapters.

2.11 Running scripts as tools

As discussed in the previous section, scripts can be run in various ways. Running a script as a tool is a great way to integrate Python scripts into ArcGIS workflows. In fact, many scripts written by Esri are made available as tools in ArcToolbox. For example, take a look at the Proximity toolset within the Analysis toolbox. Notice that the Multiple Ring Buffer tool is a script tool, as shown in the figure. ➔



When you open the tool dialog box, it looks like a regular tool with several required and optional parameters. So from the perspective of a regular ArcGIS user, all tools in ArcToolbox look the same.



Multiple Ring Buffer

Input Features

Output Feature class

Distances

Buffer Unit (optional)
Default

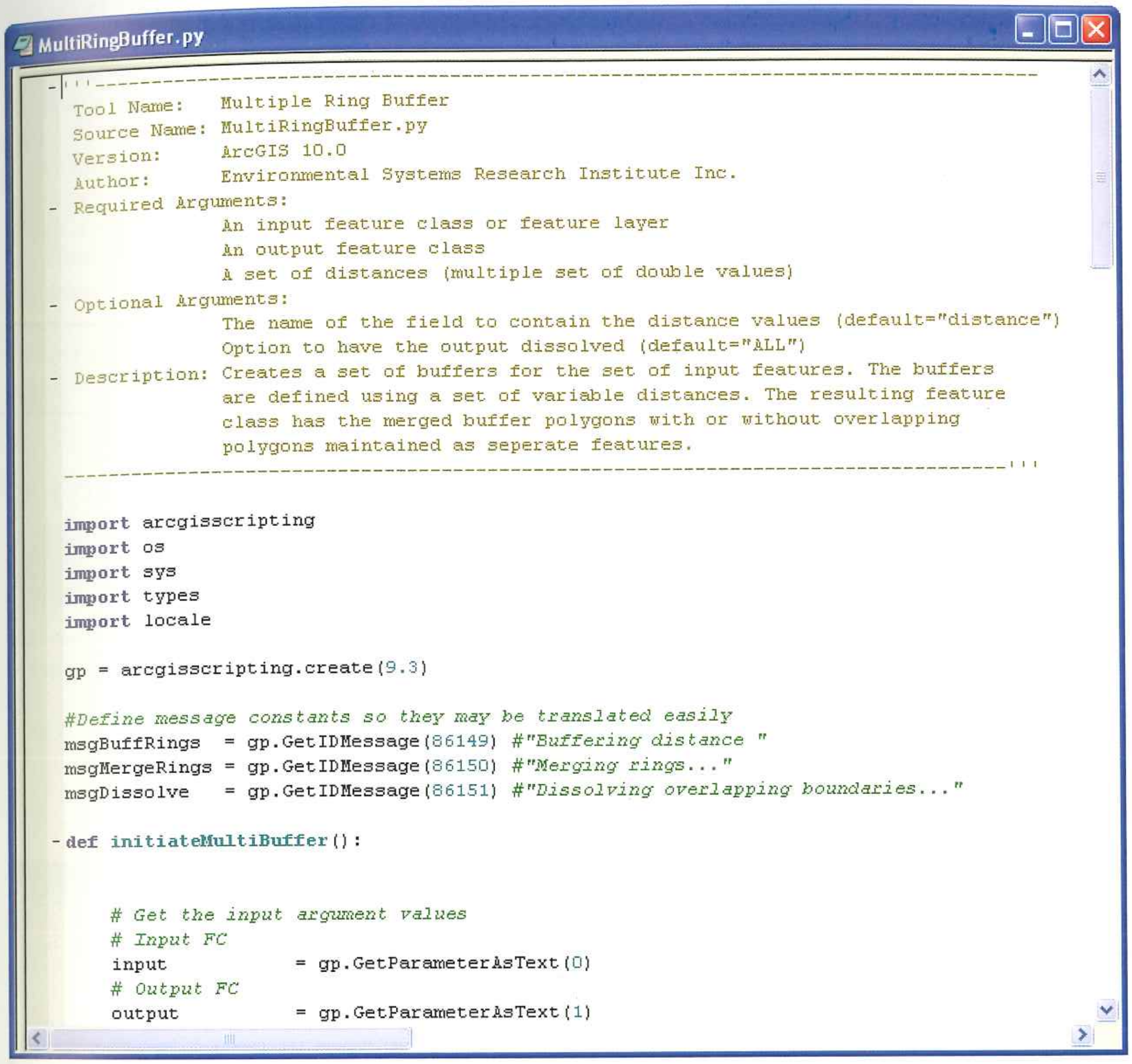
Field Name (optional)
distance

Dissolve Option (optional)
ALL

☐ Outside Polygons Only (optional)

For most system tools in ArcGIS for Desktop, the underlying code cannot be viewed. However, for script tools, you can look "under the hood" by opening the script. To view the contents of a script, right-click the script tool and click Edit. This shows that the tool calls a script called MultiRingBuffer.py. These scripts are typically located in C:\Program Files\ArcGIS\Desktop10.1\ArcToolbox\Scripts. Several dozen of the system tools that come with ArcGIS for Desktop are script tools, and their content can be viewed in this manner.

The MultiRingbuffer.py script that is attached to the Multiple Ring Buffer tool is shown in the figure (see facing page).



```
'''-------------------------------------------------------------------------------
 Tool Name:   Multiple Ring Buffer
 Source Name: MultiRingBuffer.py
 Version:     ArcGIS 10.0
 Author:      Environmental Systems Research Institute Inc.
 Required Arguments:
              An input feature class or feature layer
              An output feature class
              A set of distances (multiple set of double values)
 Optional Arguments:
              The name of the field to contain the distance values (default="distance")
              Option to have the output dissolved (default="ALL")
 Description: Creates a set of buffers for the set of input features. The buffers
              are defined using a set of variable distances. The resulting feature
              class has the merged buffer polygons with or without overlapping
              polygons maintained as seperate features.
-------------------------------------------------------------------------------'''

import arcgisscripting
import os
import sys
import types
import locale

gp = arcgisscripting.create(9.3)

#Define message constants so they may be translated easily
msgBuffRings  = gp.GetIDMessage(86149) #"Buffering distance "
msgMergeRings = gp.GetIDMessage(86150) #"Merging rings..."
msgDissolve   = gp.GetIDMessage(86151) #"Dissolving overlapping boundaries..."

def initiateMultiBuffer():


    # Get the input argument values
    # Input FC
    input          = gp.GetParameterAsText(0)
    # Output FC
    output         = gp.GetParameterAsText(1)
```

Reading through this code can give you ideas for writing your own code. The script is too lengthy to discuss in detail here. However, the basic idea is that you can create a script and add it as a tool to a toolbox so that it becomes a script tool a user can use without having to work directly with the Python code.

Keep in mind that not all system tools are necessarily updated to reflect the latest possibilities in Python scripting. For example, the Multiple Ring Buffer script still uses the `ArcGISscripting` module, the predecessor to the ArcPy site package. That is why you see the line `gp = ArcGISscripting.create(9.3)`, followed by frequent references to this geoprocessing object. Esri does not necessarily update all its code since the tool works fine using the older `ArcGISscripting` module.

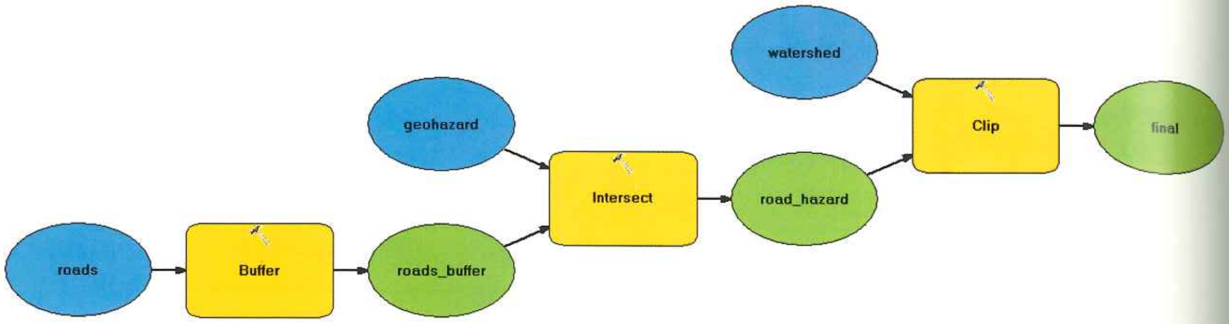
Tools that are altered substantially from earlier versions, however, are more likely to be rewritten specifically using ArcPy.

The script tools that are part of the system tools are read-only and cannot be edited. However, you can copy parts of the script code, or you can copy the script file itself to a different location and make edits to the script.

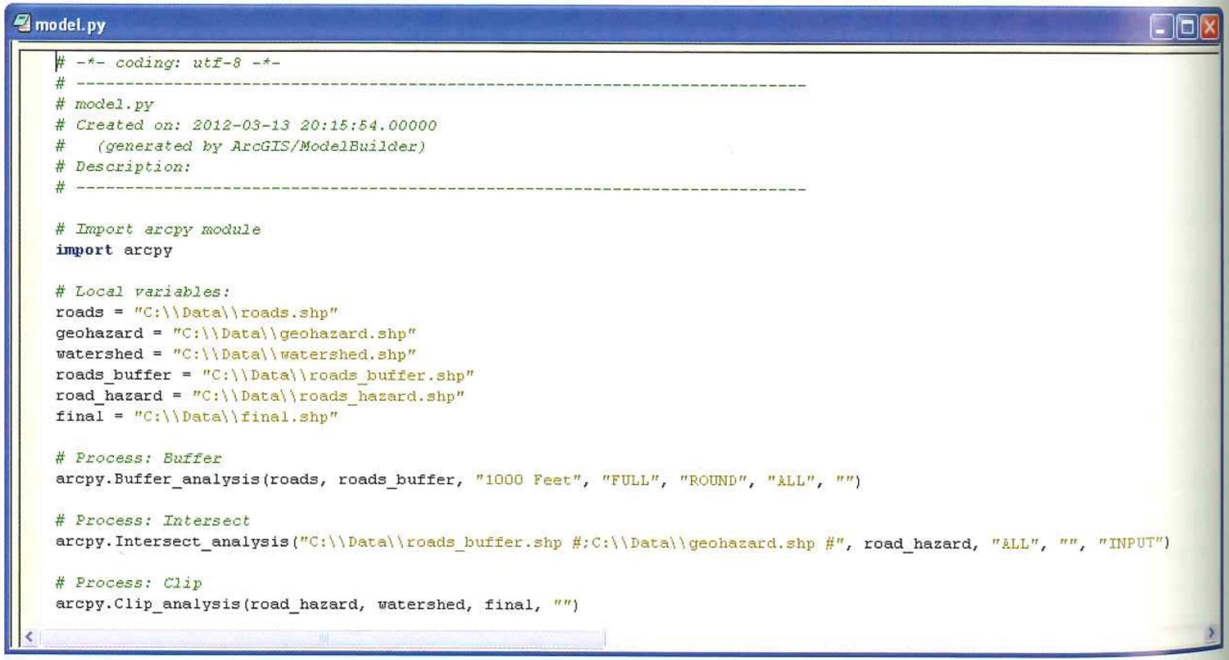
2.12 Converting a model to a script

Models and scripts are analogous in that they are both used to create a sequence of geoprocessing tasks. Models can be converted to Python scripts. On the ModelBuilder menu bar, click Model > Export > To Python Script.

Take a look at the model created previously, as shown in the figure.



After exporting the model to a script, the Python code looks like the example in the figure.



```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# model.py
# Created on: 2012-03-13 20:15:54.00000
#   (generated by ArcGIS/ModelBuilder)
# Description:
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy

# Local variables:
roads = "C:\\Data\\roads.shp"
geohazard = "C:\\Data\\geohazard.shp"
watershed = "C:\\Data\\watershed.shp"
roads_buffer = "C:\\Data\\roads_buffer.shp"
road_hazard = "C:\\Data\\roads_hazard.shp"
final = "C:\\Data\\final.shp"

# Process: Buffer
arcpy.Buffer_analysis(roads, roads_buffer, "1000 Feet", "FULL", "ROUND", "ALL", "")

# Process: Intersect
arcpy.Intersect_analysis("C:\\Data\\roads_buffer.shp #;C:\\Data\\geohazard.shp #", road_hazard, "ALL", "", "INPUT")

# Process: Clip
arcpy.Clip_analysis(road_hazard, watershed, final, "")
```

The script contains all the elements from the model: the data inputs (roads, geohazard, and watershed) and the tools (Buffer, Intersect, and Clip).

Although a model can be exported to a Python script, however, the reverse is not true. Python scripts are more versatile than ModelBuilder, so that a Python script cannot be exported to a model.

Creating a model and converting it to a script is a good way to be introduced to what scripts look like and to get familiar with Python syntax. When a model is converted to a script, the Python code very closely follows the elements in the model, without adding anything extra. For example, the resulting script does not contain any specific validation or error-checking procedures. In general, converting a model provides a starting point for writing a script, including very specific code blocks, but it rarely results in a finished script.

2.13 Scheduling a Python script to run at prescribed times

Stand-alone scripts can be set to run at prescribed times. This can be useful for such things as carrying out routine data maintenance tasks. The steps for accomplishing this depend on the operating system.

Step 1: Access the scheduled tasks.

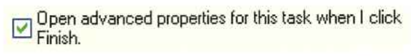
- For Windows XP: On the taskbar, click the Start button, and then, on the Start menu, click Control Panel > Scheduled Tasks. If the Control Panel is in category view, select Performance > Maintenance > Scheduled Tasks.
- For Windows Vista: On the taskbar, click the Start button, and then, on the Start menu, click Settings > Control Panel > System and Maintenance. Then click Administrative Tools > Schedule Tasks.
- For Windows 7: On the taskbar, click the Start button, and then, on the Start menu, click Control Panel > Administrative Tools > Task Scheduler. If the Control Panel is in category view, click System and Security > Administrative Tools > Task Scheduler.

Step 2. Double-click Add Scheduled Task (or Create Basic Task).

Step 3. Complete the options on the wizard.

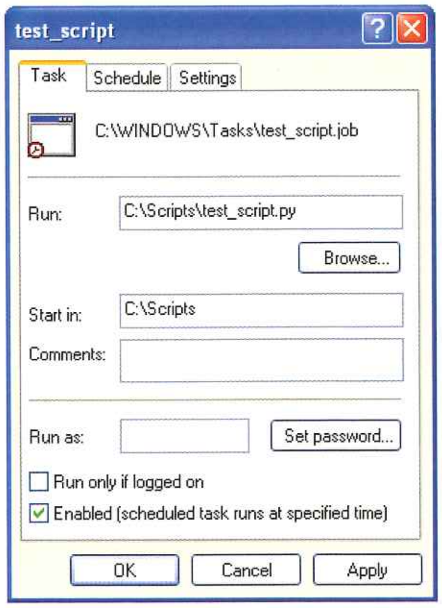
When asked to click the program you want Windows to run, click the Browse button to select the Python script.

Many Python scripts require arguments to run. These can be set as part of the scheduled task. On the last dialog box of the wizard, select the "Open advanced properties" check box:



✓ Open advanced properties for this task when I click Finish.

On the dialog box that opens, the script to be run is shown in the Run box.



test_script

Task | Schedule | Settings

C:\WINDOWS\Tasks\test_script.job

Run: C:\Scripts\test_script.py

Start in: C:\Scripts

Comments:

Run as:

☐ Run only if logged on

✓ Enabled (scheduled task runs at specified time)

For a script to run with arguments, the Run box needs to be changed to a string that contains the Python executable file, the script, and the arguments to be passed to the script. For example, this would look like the following code:

```
c:\python27\python.exe c:\data\testscript.py c:\data\streams.shp
```

These arguments are similar to the way parameters are passed to a script from a script tool. If all information needed to run the script is hard-coded in the script itself, no arguments are needed.

Scheduling a Python script to run at prescribed times appears relatively simple, but there are some potential obstacles. First, the computer needs to be turned on for a scheduled task to be executed. Second, scheduled tasks typically require administrative access, and login information needs to be provided when the task is set up. Finally, many Windows-based PCs are configured to be locked or to log off current users after a certain period of inactivity, which can interfere with running scheduled tasks. So before you can rely on scripts being run in this manner, it is worthwhile to test your computer configuration to ensure scheduled tasks are run properly.

Points to remember

- The geoprocessing framework in ArcGIS provides a powerful yet flexible system for organizing and running tools.
- ArcGIS has a large number of tools, organized in toolboxes and toolsets within ArcToolbox. The different types of tools include built-in tools, script tools, model tools, and custom tools.
- A tool is run by specifying tool parameters, including input and output datasets, and other parameters that control how a tool is executed.
- Environment settings also control how tools are executed and can be set at different levels.
- You can create your own tools using models and scripts. Once you have created your own tools, they work exactly like regular tools.
- ModelBuilder provides a visual programming language for creating a sequence of geoprocessing tasks. Models act like a flowchart.
- Python provides a text-based programming language for creating a sequence of geoprocessing tasks. Python code can be run in the Python window directly within ArcGIS. Python scripts (.py files) typically consist of more complex code and can be executed in various ways: directly from within the operating system, using a Python editor like PythonWin, or from a script tool within ArcGIS.
- Both models and scripts work like tools within the ArcGIS geoprocessing framework. Models can be converted to a script, but scripts cannot be converted to a model. Converting a model to a script is a good way to get started writing scripts in Python.