MAXScript

The Autodesk® 3ds Max® platform offers a full-featured scripting language called MAXScript. This scripting capability enhances the functionality of the software beyond what can be accomplished with the predefined user interface. Although it is beyond the scope of this appendix to teach MAXScript in depth, you will get a high-level overview of the software's terminology as well as a hands-on taste of using a script.

3ds Max Scripting with MAXScript

MAXScript is a powerful scripting language designed for nonprogrammers that gives you the ability to create, manipulate, animate, and control objects in your scene. Nearly everything you can do in 3ds Max can be performed or controlled from a script, and most menu and toolbar buttons you use in 3ds Max run scripts in the background to perform their functions. Learning how to leverage MAXScript is a great way to reduce the need to perform highly repetitive and standardized processes manually, greatly improving your productivity.

In this section, you will learn the basics of MAXScript and create and modify a simple script for creating an array of objects.

Understanding MAXScript

One of the easiest ways to understand how a script command is formatted is to open the MAXScript listener by choosing MAXScript ➔ MAXScript Listener (or pressing F11 on the keyboard), and choose MacroRecorder ➔ Enable (see Figure BA2.1).

**Figure BA2.1**
Enable the MacroRecorder.
With the listener enabled, all scripts triggered or commands you would use to create objects will display in the upper view of the listener. For instance, when you create an object such as a box in a viewport, the MAXScript command may look like this:

```
Box lengthsegs:1 widthsegs:1 heightsegs:1 length:4 width:10
   height:25 mapcords:on realWorldMapSize:on pos:[0,0,0] isSelected:on
```

Copying this command into the lower window of the MAXScript Listener will run that script and create the box. Any command entered in the lower window will run instantly; however, a script program is typically created and edited from the MAXScript editor (MAXScript Editor) and can be as simple as the command to create a box or as complicated as a full application with a user interface (UI) to read and write custom geometry, access databases, or perform other unique and complex functions within 3ds Max, including being used as a calculator. For example, in the lower MAXScript window, type `3 + 3` and press Enter. The MAXScript Listener will evaluate what you have entered and return the value 6, as shown in Figure BA2.2.

**Figure BA2.2**
The MAXScript Listener can evaluate mathematical equations.

Simple scripts for performing repetitive commands can be created by performing the actions in 3ds Max and then copying and pasting the scripts into the MAXScript editor. Figure BA2.3 shows a sample script.

**Figure BA2.3**
MAXScript Editor sample script
In the MAXScript Editor, you press Ctrl+E to evaluate (run) the script program. Scripts can be saved to a file and assigned to menu buttons to be run quickly at any time. Scripts can be automatically evaluated when 3ds Max is first run by placing the script into the C:\Program Files\Autodesk\3ds Max 2013\scripts\Startup folder. Scripts can be encrypted to protect intellectual property and prevent reverse engineering with the MAXScript command encryptScript. If you want to learn about encrypting your scripts so that they can be run on any 3ds Max installation, but the source code cannot be viewed or copied, you need to read that section in the General MAXScript Topics of the MAXScript Help.

When working with scripts, you are often creating and accessing objects; these objects may be renderable geometry, arrays of numbers, variables, renderers, and so on. Accessing and controlling the object typically requires knowing the name of the object, adding a dot after the name, and then a command or parameter name and optionally new values to pass to the object. To change the height of the box created earlier, you could run the command $Box001.height = 5 to explicitly set the height of an object named Box001. To add 5 units to the existing height, you would use a command like $Box001.height = 5 + $Box001.height. The MAXScript listener will echo the new height in the view and the object will reflect the new value.

MAXScript is a complete programming language that includes the ability to loop operations, subdivide sections of programs into functions, and react to user interface events such as pressing a button or changing a value or option. A user interface (UI) is created by manually defining objects or by using the MAXScript Editor and the Tools > New Rollout menu option to graphically create and edit the UI elements and the events that are triggered.

**MAXScript Resources**

The Help files for MAXScript are easy to find by selecting Help > MAXScript Help from the Main menu.

www.scriptspot.com is a user community on the Internet for those interested in writing MAXScript tools for 3ds Max. The site has tutorials, user-submitted scripts, plug-ins, forums, and blogs.

John Wainwright, the creator of MAXScript, created a training DVD in 2002 called MAXScript 101. He has now made this resource and a Zip file of all the covered scripts available for free on vimeo.com. The five-part course is broken down into easy-to-follow clips on the site.

**USING MAXSCRIPT**

The easiest way to learn a new subject like scripting is to take something that works and modify it to see how the parts function. Troubleshooting is an essential skill in programming. If the program stops working correctly, you’ll know where to look to troubleshoot the problem more easily. The following simple script will create a one-dimensional array of teapots in your scene:

```plaintext
myX = 0    -- Initial Teapot Position
myY = 0
myZ = 0
myXOffset = 10      -- X Offset for each teapot
myQuantity = 5    -- How many teapots
myRadius = 3      -- Radius of the teapots
```
delete $myObj* -- Delete all scripted objects

for i = 1 to myQuantity do
{
    obj = Teapot radius:myRadius smooth:on segs:4 handle:on \ 
         spout: on lid:on \ 
         mapcoords:on realWorldMapSize:on \ 
         pos:[myX, myY, myZ] isSelected:on

    obj.name = uniquename "myObj"    -- Add unique numbers to name
    myX += myXOffset    -- Increment the X location
}

Type the code into the MAXScript Editor, and press Ctrl+E to run the script. As with all programming, syntax and spelling are crucial for code to work properly.

As shown in the first line of code, placing two minus signs (--) in a line causes MAXScript to ignore the remainder of the line, allowing you to add descriptive comments, called remarks in programming parlance, to the code. The first three lines of this script create the variables myX, myY, and myZ that set the initial position of the first teapot. As teapots are created, the number in myXOffset is added to myX to create the next teapot in a new X location. The total number of teapots is defined by myQuantity, and the teapot radius is in the variable myRadius.

In this example, after variables are declared; that is, given a type and value, the next line deletes any objects in the scene with a name that starts with myObj. The * is a wildcard that means “any additional characters,” and any object starting with that name will be deleted. This has an effect only after the code is run once and teapots exist in your scene.

The next section is a for loop that creates the array of teapots; the code between the parentheses will be run repeatedly until the desired number of loops is reached, as defined by the myQuantity variable. The i = 1 code creates an incrementing variable i with an initial value of 1, and 1 will be used to keep track of the loops. Within the loop parentheses are three commands: the command to make a teapot, a command to change the name of the new teapot, and code to increment the myX variable for the next loop.

A teapot is created in the first four lines of code within the loop. At the end of the first three lines is a \ (backslash) that means the current line is continued on the next line and the next line is seen as part of the first line. These four lines could be on a single line but are split for clarity. The code that starts with obj = Teapot creates a new variable called obj that is a reference to the new Teapot object, and it allows you to perform additional operations later on the teapot. Each teapot parameter is then listed with a colon and then a value or variable, for example, radius:myRadius and handle:on. For the position of the new teapot, the code pos:[myX, myY, myZ] controls the location. The three values for X, Y, and Z must be placed between the brackets and separated by commas.

The next line, obj.name = uniquename "myObj", accesses the name parameter of the new object and replaces it with a new name myObj concatenated with the next in a series of numbers determined by the uniquename command. Commands and parameters for objects are accessed by placing a dot after the object name and then the parameter you want to control. For instance, you can change the number of segments of the Teapot object by using the command obj.segs = 12.
The final line of code takes the current teapot position `myX` and increments it by the number in the variable `myXOffset`. The code `+=` is a shortcut for incrementing the variable on the left with the variable or value on the right. With each loop, the `myX` position is changed so that the teapots are offset from one another.

When you have finished working with MAXScript, make sure to close all the scripts you have been working on, saving the ones you want to keep. Otherwise, they will automatically be reopened in the MAXScript Editor the next time you start 3ds Max, and you won’t know it unless you open the editor.

---

**GARbage Collection**

Entering `gc()` into the Mini-Listener at the bottom left corner of the 3ds Max interface and pressing Enter will cause 3ds Max to evaluate the scene and perform an operation called *garbage collection*. This can clean up your scene and improve the performance of 3ds Max. Garbage collection purges the 3ds Max Undo Scene Operation buffer, clears the MAXScript heap, closes unreferenced open files, and displays the amount of free memory in the MAXScript heap after the operation.