Debugging an Avenue Script

This is a short tutorial on debugging your scripts in Avenue. It briefly introduces an approach that you can use to debug your code, and then describes three tools that ArcView provides to help in this process.

Types of Problems

There are three basic types of problems that you will encounter with your scripts. They are:

1. Your script will not compile.
2. Your script compiles, but gives you an error message when it runs (the worst case scenario here is that you get a “segmentation violation” and ArcView crashes).
3. Your script runs, but doesn’t do what you want it to do.

I put these three types of problems in order of difficulty from easiest to solve to hardest to solve.

Finding the problem

The hardest part of solving any of these types of problems is finding where in your script the error is occurring. What specific line (or section) of code is creating the problem?

In the case of type 1 problems, the compiler will place the cursor on a line where it thinks the problem is. This is usually very helpful, but can sometimes be misleading. If you cannot find any problem with the line that the compiler points to, look for other lines of code earlier in the script that use the same variables. You can also look earlier in the script for errors that do not properly close off sections of code (such as missing “end” statements, or closing parentheses, and infix operators without a second argument).

Type two problems usually arise because a variable is not referencing what you think it should be referencing. For instance, you may have a variable called \textit{myTheme} that really references a list of themes. Therefore the compiler sees a variable with a reasonable request (such as \textit{myTheme.SetVisible(true)}), but when you run the code you will get an error saying that a list object does not recognize the request \textit{SetVisible}. Similar to compile errors, this type of error will also place your cursor at the point in the code where it believes the error is occurring. In this case, however, the error is rarely with that line of code. It is in an earlier line where the particular variable is being set. To help debug your code, you can find out what type of object the variable is actually referencing, and then try to figure out how it got set to that object. I’ll explain how you find out what type of object the variable references in a section below.

Type three problems are often the most difficult to solve because ArcView doesn’t give you any help in discovering where the problem is occurring. In order to solve this type of problem, you first need to make a guess at what object isn’t doing what it’s supposed to do, and then find places in your script that work with that object. Then try to narrow down the potential problem areas by guessing which lines of code could be executing incorrectly. Sometimes you will find that errors occur because \textit{if .. then} statements are
not testing for the right conditions, and therefore certain sections of code never execute. Below are some strategies that you can use to help locate problems.

**Strategies for Finding Problems**

1. Comment out lines of code, and see what effect it has on the script. This is especially useful for type 1 and type 2 problems.

2. Insert message boxes that popup and tell you what the variable values are at different points in the script. This is especially helpful with type 2 and type 3 problems. With type 2 problems, this will only work if the message box is somewhere before the problem occurs (of course).

**ArcView’s Debugging tools**

ArcView provides you with a few simple tools that help you pinpoint where the problem in your script is occurring. They are (from left to right in the graphic below):

1. **Step:** Step allows you to move through a script one operation at a time.

2. **Breakpoints:** Allows you to set points in your script that will pause the execution. Once the execution is paused, you can use the **step** tool to move one operation at a time, and use the **examine variables** tool to see what the types and values of your variables are.

3. **Examine Variables:** This tool pops up a dialog that lists all variables that are used in the script, tells you what type of object each variable references, and (where possible) lists the value of the variable.

Using these tools you can stop your script at a specific location, move slowly through the script, and constantly monitor the value of your variables. Even if you are not running into any problems, these tools can also help you understand how your script is working. Sometimes it’s fun (in a painful sort of way) to just step through a simple script, and see what it does. Try this out sometime.