Getting the Most of ArcView GIS

Performance Timing Tools

By Todd Stellhorn, ArcView GIS Development Lead

Large Avenue applications typically contain many scripts. Multiple scripts are triggered with each update event. Because so many scripts are required by complex applications, it is difficult to identify which scripts are actually being run for a given set of input events, determine the order these scripts execute, and evaluate the performance of individual scripts.

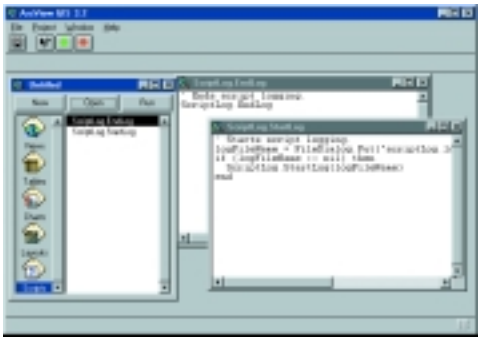
ArcView GIS 3.2 comes with Script Log, a new sample extension that addresses the problem of monitoring multiple scripts. Script Log extends the Avenue development environment with a set of simple performance timing tools. This extension loads a new dynamic linked library (DLL) called ScriptLog.DLL. To make the extension available to ArcView GIS, copy all the scrlog.xxx files from the ArcView/Samples/ext subdirectory to the Arcview/Ext32 subdirectory, choose File > Extensions, and check the Script Log extension in the dialog. Additional information on the Script Log extension and the other sample extensions included with ArcView GIS can be found in the online Help by searching the index for Samples, extensions, and then scrolling through the results for the extension name.

Most developers resort to the tried and true method of using message boxes to identify executing scripts and indicate their sequence. This method is cumbersome at best and falls well short of helping the developer determine the runtime performance of each script. The Script Log extension creates log files containing the names and run times of executing scripts. It adds two buttons to the Project GUI—a Start button (green diamond icon) and a Stop button (red diamond icon). After pressing the Start button, the user is prompted for an input file name and location. After specifying this information, the user runs or debugs an Avenue application and the extension logs information on every script that runs. Pressing the Stop button ends the logging session. The contents of the log file can be viewed using Microsoft Word or another word processing program. Figure 1 shows an example of output from the Script Log extension.

Unloading the Script Log extension will generate an error message stating that ScriptLog.DLL cannot be unloaded. While the DLL will not unload until the user exits ArcView GIS, the Script Log extension unloads correctly.

Figure 1

```
Doc.OpenUpdate: 0 (0)
Doc.ActionUpdate: 0 (0)
Project.UpdateButtons: 0 (0)
View.New: 2 (0)
View.HasThemesUpdate: 2 (0)
View.ActiveDeletableThemesUpdate: 2 (0)
View.ActiveThemesUpdate: 2 (0)
View.ActiveDeletableThemesUpdate: 2 (0)
View.UndoEditUpdate: 2 (0)
View.RedoEditUpdate: 2 (0)
View.CutUpdate: 2 (0)
View.CopyUpdate: 2 (0)
View.DeleteUpdate: 2 (0)
View.CombineUpdate: 2 (0)
View.UnionUpdate: 2 (0)
View.SubtractUpdate: 2 (0)
View.IntersectUpdate: 2 (0)
View.PasteUpdate: 2 (0)
View.HasGraphicsUpdate: 2 (0)
View.GeocodeUpdate: 2 (0)
View.AddEventUpdate: 2 (0)
View.HasThemesUpdate: 2 (0)
View.HasThemesUpdate: 2 (0)
View.HasThemesUpdate: 2 (0)
TocDefs.ShowUpdate: 2 (0)
View.HasDataUpdate: 2 (0)
```



Script Log, an extension that comes with ArcView GIS 3.2, logs every script that runs while it is invoked. The green and red buttons that are added to the Project GUI start and stop logging sessions.

Loading ScriptLog.DLL Directly

The core performance timing tool can also be used independently of the extension. Using ScriptLog.DLL separately gives the developer maximum flexibility when debugging and performance tuning. To use ScriptLog.DLL directly, load it using Avenue request System.LoadLibrary. LoadLibrary takes a FileName as its input argument and points to the ScriptLog.DLL file. Loading the DLL adds the ScriptLog Avenue class with two class requests—ScriptLog.StartLog(aFileName) and ScriptLog.EndLog.

The ScriptLog.StartLog request opens a new log file using the input filename. After this request is called, each time an Avenue script runs, the ScriptLog class writes the name of the script, the cumulative run time (in seconds), and the script run time (in seconds) to the log file. Note that the cumulative run time may include wait time and nonscript invoked run time. The ScriptLog.EndLog request stops the logging of scripts and closes the current log file.

How Does It Work?

The ScriptLog class uses a method employed by other extensions that require the behavior of an existing Avenue request to change dynamically at run time. In the case of the ScriptLog class, the Avenue request av.Run must be modified to add the script logging behavior. As part of the class initialization, the Script Log extension first uses Avenue to find the Run request on the av (Application) class. Once the Run request is found, the DLL stores a local pointer in the request's executable code as a function pointer and replaces the request's executable code with a new version defined by the ScriptLog class. The new version of av.Run contains the code needed to calculate the performance timings and produce the output. The new version of av.Run uses the original av.Run to actually run the script. An example of the Script Log extension's version of av.Run that uses a pseudo C coding style is shown in Figure 2.

Whether used with the extension or independently, the ScriptLog.DLL is a useful tool for developers debugging and fine-tuning Avenue applications.

Figure 2

```
returnObject av.Run(scriptName, arguments)
{
  if (logFileOpen)
    store current time

  returnObject = (old.RunFunction)(scriptName, arguments)

  if (logFileOpen)
  {
    calculate script run time
    calculate duration
    write results to output(script run time, duration)
  }

  return returnObject
}
```