Documenting Your Code

- Naming Scripts
- Coding Conventions in Scripts:
- Other Documentation
- Written Documentation for Users
Why Document?

• To make it easier to maintain and modify the system.
• To make it easier for users to work with the system.
<table>
<thead>
<tr>
<th><strong>Within Scripts</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Variable Naming Conventions</td>
</tr>
<tr>
<td>• Comment Lines</td>
</tr>
<tr>
<td>• Script Headers</td>
</tr>
<tr>
<td>• Alignment</td>
</tr>
<tr>
<td>• Reserved Words</td>
</tr>
</tbody>
</table>
Global Variables

- Global variables are variables that ArcView remembers even when the script has stopped running. Thus you can access the value stored in a global variable from more than one script.
- Global variable names begin with an “underscore” character:
  `_variableName`
Local Variables

• Local variables can be set and accessed within one script. When the script ends, their values are lost.
  – Always start variable names with a lower-case alphabetic letter (a-z).
  – Use mid-word caps if the variable is a concatenation of more than one word.
  – You can use specific codes at the end of a variable name to indicate classes of variables.
Comment Lines

• Add comment lines in code with a single quote:
  `** This is a comment`

• Sometimes it’s good to add some characters that make the comment line jump out at you when your scanning through a script. (like the asterisks in the example above)
Comment Lines

• Comments should provide clear and concise descriptions of what a block of code accomplishes. They should not simply restate what you can see in the code.
Script Headers

- Script Name
- Main purpose of script
- Author name
- Date created
- Date modified and reason for modification
- List and describe all variables
- SELF object (parameters)
- Return object
Update GUI
Written by Dan Haug 4/12/98
Last Updated 1/20/00 (Updated variable calls for new globals)

This script goes through the system's GUIs and updates them based on the global variables, which indicate what tools are active at any given time.

Self: no params
Return: none

Global Variables:
__crossMode -- (yes or no) tells whether map is bivariate or univariate
__scMode -- (yes or no) tells whether the scatterplot is active
__theClasses -- (2, 5, or 7) the number of classes for the thematic map

Local Variables:
m -- used to represent the menu object being operated on in each of the loops

*******************************************************************
Alignment and Indentation
• Use indentation to keep track of blocks of code:

if (_crossMode) then
  for each aButton in av.GetActiveGUI.GetButtonBar
    oTag = aButton.GetTag
    if (oTag = "@") then
      if (_scMode) then
        aButton.SetEnabled(TRUE)
      else
        aButton.SetEnabled(FALSE)
      end
    end
  elseif (oTag = "focus") then
    aButton.SetEnabled(TRUE)
  end
end
Capitalization

- Capitalize…
  - Classes
  - Requests
- Do not Capitalize…
  - variables
  - reserved words (if, then, for each, … etc.)
- Always use mid-word caps for concatenated words
Documenting the Application

• Naming Files
• Naming Scripts
• Tracking Scripts
• Tracking Global Variables
• Writing User Documentation
File Naming Conventions

- apr – ArcView project
- ave – Avenue script
- avx – ArcView extension
- dbf – Dbase file
- odb – Object database file
- shp – ArcView shape file
- txt – Text file
Script Naming Conventions

• Categorize your scripts, and use hierarchical naming conventions
• Examples:
  – View.Zoom.In.Button.20percent
  – View.Zoom.In.Tool.20percent
  – View.Zoom.In.10percent
  – View.Pan.Tool
Scripts calling other scripts

• In Avenue you can modularize your code by calling one script from another, and returning a value to the first.

• Maintain a list of scripts that are called by other scripts.
  – This will help in maintaining and updating code.
Global Variables

• Compile a list of all global variables. Include:
  – Variable name
  – Short description of the variable
  – List of scripts that set or call the variable
Writing User Documentation

• Provide an introduction to your custom applications, including:
  – The goals of the application
  – A brief discussion of the applications functionality
  – Any specific bits of information that may be useful in getting the application up and running.

• List all modifications to the user interface.
  – Describe the purpose of each interface tool (this should be easy, as you started this in your requirements study).
  – Describe precisely how each tool is used.
NCHS Sample Documentation

- List of global variables called by scripts
- List of scripts called by other scripts
- List of buttons and tools
- User Guide
Conclusion

- Choose coding conventions within your group and stick to them.
- Draw on your requirements study to write the introduction of your user documentation.
- Be sure to list all added functionality in your user documentation.