The JSL Debugger – Basics

Ingredients:

Debugger

Difficulty – Easy

The JSL Debugger allows users to step though program code and observe values of variables. It is particularly useful when looping through a large number of values and if multiple functions are used. This first recipe looks at some basic debugger functionality. To do this, we will use a pre-baked program. The program parses each cell in a table to build vocabulary lists.

1. Blank cells are removed
2. Cells starting with “[v.” are ignored.
3. Commas separate individual words.
4. A slash (/) gives a list of words to combine one or more words after the tilde (~). For example, tener/llevar ~ barba/bigote/gafas is to expanded to six words: tener barba, llevar barba, tener bigote, llevar bigote, tener gafas, llevar gafas. There may be more than two lists in a cell
5. There may be both slashes and commas in a cell. The entry: tronco, extremidad ~ superior/inferior, should be expanded to: tronco, extremidad superior, extremidad inferior.

Steps:

1. Launch the debugger by selecting Run > Debug Script or by clicking the Debugger icon in the menu area.
2. Using the Step Into button at the top, step six times until you reach line 52. If the lines aren’t showing, right click and select Show Line Numbers. The lines from the file in the Include statement will display even if the file is not open.
3. Step once more. All the buttons at the top except Pause are greyed out. Control has been passed to the dialog box created by the program. You may not be able to tell if the dialog is behind the debugger window. If this is the case, move the debugger to select the dialog.
4. Select Section from the list box on the left and move it to Level by dragging or clicking the Level Col button.
5. Control is passed back to the debugger. Click Step Into.
6. Control is passed back to the dialog. We can avoid this back and forth by setting a breakpoint next time we are in the debugger. Select the first column after Section and add it to the Data list box.
7. When control is passed back to the debugger, set a breakpoint on line 7 by clicking to the right of the number. A red dot appears. Breakpoints are persistent across both debugging and JMP session.
8. Close the debugger. Click the Stop button at the top. A warning will appear. It may be behind the debugger. It indicates the operation (stopping the debugger) could not be completed. This is because the dialog box generated by the program is still open. Dismiss the warning and close the dialog. The debugger will close.
9. Relaunch the debugger. The breakpoint on line 7 is still there. Click the Run button at the far left. Fill in the dialog with Section as Level and the three columns below it as Data. Click Run. Control is passed to the debugger which is stopped on line 7.

10. Set a breakpoint on line 26 which references a function from the Include file. Run to this line.
11. Click Step Into. This will take you into the function from the included file. Click Step Into once or twice more. This will step through the current function.
12. Click the Step Out button. Step Out takes you to the next line outside of the function that is currently being stepped through.
13. Click the Run button once more to get back to the line 26 breakpoint. Click Step Over. This allows you to execute the code associated with the function in the current line without having to step into or through the function.
14. Remove the breakpoints on lines 7 and 26 by clicking them. Set breakpoints on lines 31 and 37. For breakpoints to be operational, they need to be set on a line with executable code. Comments or blank lines, or lines consisting solely of a grouping operator (e.g., close parenthesis) or a semicolon will not work.
15. In the lower left, click on the Watch tab. We are going to add several variables to the watch list. This allows us to observe their values at breakpoints. In the debugger code pane, double click `nextSet` (line 15). Right click and select Add Watch. It will appear in the Watch tab. Add `subSet` (line 17), `tildeGroup` (line 18), and `buildCartesian` (line 26). You can also type variable names directly into the watch list. Click Run a few times to observe the watched variable values changing. Watched variables can be deleted by selecting one or more from the Watch list and clicking the single X Delete button at the top of the section. The double X Delete button removes all the watched variables.
16. In the lower right, click on the Breakpoints tab. Any or all breakpoints can be deleted here. Click on the double X Delete button to remove all the breakpoints.
17. Alternatively, we could have clicked the Run without Breakpoints button at the top (second from the left) to bypass all the breakpoints.
18. Run to the end of the program. To close the debugger, dismiss the dialog and click the Stop button at the top of the debugger.

Hints for Success:
- Step Into takes the smallest steps, always stepping into and through functions.
- Use Step Out to get out of the current function. Step Over jumps over a function
- Breakpoints must be set on executable lines.
- Use the Breakpoints tab to manage breakpoints.