ChatScript Debugger Manual

Copyright Bruce Wilcox, mailto:gowilcox@gmail.com www.brilligunderstanding.com
Revision 2/18/2018 cs8.1

You can run an interactive debugger environment (IDE) if you have Windows. You launch it by going to BINARIES and double clicking on ChatScript IDE. It will load your bot, just as running ChatScript will. And you can type input into the input window and see output in the output window just like normal ChatScript. But you can do so much more...

You can set breakpoints on lines of code, be they actions in a macro, rules in a topic, or the topic itself. You can break on when CS will output a message to the user (to see where it comes from). You can break on change in value of a variable. You can single step in, over, out. You can display values of user, system, and match variables.

Running the IDE

Before you can run the IDE you should compile your scripts under CS 8.0 or higher. You can actually do that from the IDE itself.

The IDE is a wrapper around the CS engine. Instead of launching `ChatScript.exe` from BINARIES you launch `ChatScriptIDE.exe`. This will load your project just as the main engine does and then come to rest in the IDE, awaiting your commands.

The display

Script window

The script window displays your source file, information on the debugger commands, debugging commands (:xxx), IDE commands, and system variables (and important CS interchange variables).

Your script files will also be displayed when you request it (:i filename) or when some breakpoint requires it to show you where you are.

The script window has your code. To the left of that are line numbers from the file. And to the left of the line numbers are CS tag id's of rules. IF you have a breakpoint set, you will see a B between the line number and your script. The entire area of of tags and line numbers is clickable to enable and disable a breakpoint at that line.

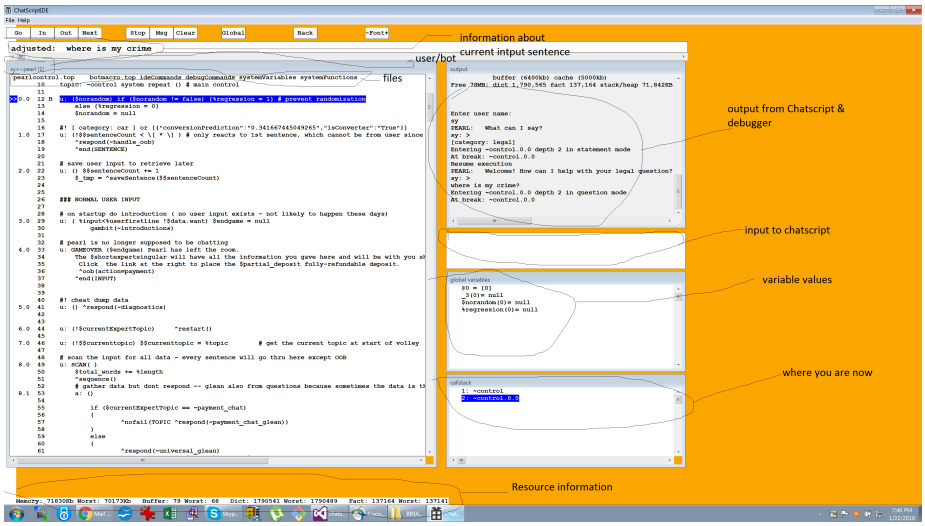

Figure 1: Debugger Screen Image

The window is also sensitive to having parts of your script clicked upon. If you click on a variable, be it $var, _var, %var, ^var, or @nvar, the current value will get added to the variables window and the value constantly displayed there. An @n variable shows the count of how many facts it has. You can remove an entry from the variables window by clicking on it.

Clicking on a function name will change the script window to display that function. The debugger will highlight where it is in blue. When you are executing your code and the debugger enters a breakpoint of some kind, the script window will show you where you currently are, displaying your script and highlighting in blue where you are. It will also put << to the left of rule tags/line numbers to indicate this is where you actually are. The callstack of where you are will be displayed in the stack window.

If you click on a rule label (like that in a ^reuse) then the system will take you to that label.

Clicking on a concept or topic name will change the script window to that's definition.

File names title bar

The first line is a list of "windows" you can see. The first name in the list is always the current windows, and if you left-click on something else in that list, it is moved to first position and displayed.

If the bar gets too crowded, right-click on a name and the name is removed from

the bar.

Who talking to whom

Above the file names title bar is text that shows you the current username and the current bot name and botid. ‘bob=>rose(0)

Breakpoints

Left-click before a script line (in the number or tag area) and the system will establish a breakpoint (and put a B after the line number). If your code executes to that point, it will stop and give you control. Click again in the same area and the breakpoint is removed.

If you Right-click before a script line, the system create a transient breakpoint there and run code. The breakpoint will be removed when execution is stopped by some breakpoint (that one or a different one).

Input window

Here is where you type input for CS or for the IDE.

IDE input is prefixed with `:i`. Naming a function or variable or concept set is just like clicking on it in the script window. It will put the variable in the variables window or display the code in the script window. You can also name a rule label and code display will jump to there. Not the full topic label, just what you put as the label in `u: MYLABEL () ...`, i.e., MYLABEL.

```
:i ~control
```

Above is the classic way to bring up the control script whereby you can then set a breakpoint. And if you type in a rule label as argument, it will take you to that label. Handy when you know where you want to debug in particular.

If the engine is currently running, you can give IDE commands but the IDE will refuse input for the engine.

Output window

Messages from the IDE as well as results or tracing from the engine are displayed here. While it is scrollable, if too much information is in the buffer it will truncate the older output.

Callstack window

When you are in the IDE at a breakpoint, this shows you the calling hierarchy of where you are. Each scope is numbered (low is deepest, high is most recent). The possible scopes are a topic, a rule, a function, an if, or a while. Scopes that have output actions (rules, if, while, function), will display that you are at that scope by displaying it with `()`. You cannot walk in and execute code within a pattern or an if test or a loop count evaluation, so when the scope is within executable output, it will display it with `{n}`, where `n` is the internal code offset of where you are, and it will show you the upcoming code it is about to execute. There is also `lvl:n` data, which is internal data about what recursive level the output interpreter is at (mostly irrelevant to you).

If you click on one of these scope lines, the system will display the code of that scope in the script window and any local variables ($_xxx and ^xxx) in the variables window. It will have a title of `local variables`.

Variables window

This window shows current values of global variables you have requested (updated whenever you end up in the IDE). If you clicked on a scope line, it shows the values of local variables for that scope. You can flip back to viewing globals by pressing the `globals` button.

When variables are displayed, after the variable it tells you how many characters the value has.

If you click to the left of a global variable, it will set a breakpoint for when the value of that variable changes and will put a B before it. Click again in that area and it removes the breakpoint. You can request a break when the value becomes =, <, or > some value using

```
:i varname = test
:i varname > 5    -- integer values only
:i varname < 5    -- integer values only
```

You can also use various engine debug commands while at a break, including `:trace` and `:word`. Just dont try to use things that have massive impact, like

```
:document, :source, :reset, :bot, :user, :restart
```

System resource information

At the bottom of the main window the system prints out useful statistics about resource usage. It shows you how much of a resource is currently in use, and

what the least amount available has ever been seen.

'Memory' is the available space for stack + heap. If this gets down toward 10M, you might want to allocate more text space to the program.

'Buffer' is how many preallocated large buffers (typically 80K) are in use. If it gets to be <10, you might want to allocate more.

`DICT` is how many dictionary entries are available. Less than 5K probably indicates you want to allocate more.

`FACT` is how many facts are available to be created. Less than 5K probably indicates you want to allocate more.

Sentence window

The window above the script window tells you information about the current user input. By default it shows you the `adjusted` input, what the tokenizer and various NLP corrections have been made to result in the input the system is actually processing at present.

L-click anywhere in the sentence box and it changes to show you the `raw` input, what the user provided (after tokenization).

L-click again and it shows you the `canonical` form of the input.

L-click again and it returns to the `adjusted` form of the input.

If you R-click on a word, it will show you the concepts it is involved in in the output window.

Buttons

Go

Resume execution until complete or a new breakpoint is hit.

In

Try to go in deeper into the current thing. For a topic that will be its first rule. For a rule, if the pattern matches, it will be the output of it. For a function it will be the code of the function.

If the system cannot go in, it will stop at the next available opportunity. Ie., the next action, rule, pass out of the function or topic.

Out

Leave the current level of behavior and stop as soon thereafter. For a topic, it leaves the topic, for a rule it leaves the topic. If within actions of a rule, it leaves that rule. For a function it leaves that function.

Next

Stay at the current level of behavior. If at a topic, move on to the next topic (having executed all its rules). If a rule, move on to the next rule (possibly having executed all its actions). If a function, step over the function (executing all its actions) and return to caller. If withing actions of a function then execute the current action and move to the next one. If inside a rule, do the current action and move to the next.

Stop

If the engine is currently executing script, stop that execution as soon as possible and enter the IDE. Otherwise note that you want to stop, so that on the next user input the IDE takes control immediately.

Msg

Breakpoint when the system wants to print output to the user. This is useful to see why you are getting the output you get.

Fail

Breakpoint if any system function return a failure code. These typically suggest you passed it bad arguments. With this, you can see when this happens.

Cntrl-lclick tells the debugger to terminate current input and return to top level.

Clear

L-click removes all breakpoints currently set (not including variable breaks). R-click temporarily disables all breakpoints set (including variable breaks and msg break and fail break). R-click again to re-enable.

Global

Restore variable window to showing global variables if it is currently showing locals of some callframe.

Back

As you move from location to location in the stack window, you create a breadcrumb trail, and `Back` will take you back from where you were. If you need to rapidly return to where you are currently executing code, just click on the last callframe in the stack window.

-Font+

Changes the font size. Left click for smaller. Right click for bigger.