ChatScript Analytics Manual

© Bruce Wilcox, gowilcox@gmail.com

Revision 11-16-2012 cs2.97

Your bot has been written and debugged and released. You are getting log files from users. What can you learn from them? That's the job of the analytics tools.

Components of a log file

A typical log file will have one or more “Start” lines and some number of “Responding to” lines.
A “start” line shows conversation initiation. It also provides a bunch of identifying information.

```
Start:  => Good morning.  prior:0 Jun  9 09:17:27 2012|.|| Sat Jun  9 09:17:27 2012 Script0:Tue May 29 12:12:32 2012 Script1:  209-138
```

After Start, the line may contain the user name, the bot name, the IP address (using a default configuration). After that you see the startup message it issued (eg “good morning”, the input volley count coming into this start (how many volleys from previous conversations have happened). Then a datestamp of when this user first ever chatted with the bot. Even in the absences of unique user names and/or IP addresses, this alone will be close to unique and can serve as a user id for tracking. Following the initiation date are the identifying topic and rule offsets of the 15th and 40th volley responses by the system. Combine THAT with the initiation date and you are probably totally unique per user. Of course above does not show those values since the volley counts haven't been reached yet. Then you get the datestamp of this specific conversation start. Followed by Script0: xxx and Script1: xxxx. Those are the datestamps of when those levels were compiled by the system, telling you which version of your data was being used.

```
Responding to: (~love_tom) 2   I .  ==>  Be careful up here.  Sat Jun  9 09:19:06 2012 114-315
```

In contrast, the responding to log entry is much simpler. It shows you the topic the bot was in when it finished responding, the volley count. What the user said, what the chatbot said, when it said it, and the topic and rule offset (s) that generated the output (the topic id is before the -, the rule offset is after).

:where

You can ask the system to decode a topic-offset pair using :where xxx-yyyy. This will display the rule identified.

The Abstract

Looking at your entire source script is tedious. It's extensive and hard to read. If you want to see what you've got in a reasonable overview, you need :abstract. It can do a variety of tasks.

:abstract

This prints out a view of the entire topic system showing its structure (gambits, rejoinders, responders) as well as conditions on gambits and normal text content of everything, but omits actual code complexity. It is useful for seeing what will be said in response to sample input (if you use the #! and #! x commands on rules. E.g.

```
****** Topic: ~introductions[]
t: ( $old   %input=0   %hour<12 $name ) Good morning, .
t: ( $old   %input=0   %hour>11   %hour<18 $name ) Good afternoon, .
t: ( $old   %input=0   %hour>18 $name ) Good evening, .
t: Where do you live?
    a: "Fukushima" =>   I've heard of _0. Were you born there?
    a: "I live in Japan" =>   I've visited Japan.
    a: "I live in California" =>   That's where I live!
    a: "Libya" =>   I would have thought you lived in Japan, not _0 .
    a: "Earth" =>   Yes, we all live on Earth.
    a: "Mars" =>   I don't believe you.
t: What do you do for a living?
u: "Am I welcome here?" =>   Of course you are welcome.
s: "I'm back" =>
   [ Where did you go?  ]
   [ Where have you been?  ]
   [ I'm glad.  ]
s: "Knock" =>   Who's there?
```

While :abstract primary does topics, you can get it to do fact data as well. It will attempt to call a function ^abstract_facts(), so if you define that, you can do whatever you want for abstracting facts.

:abstract ~topicname

Calling :abstract with a topic will limit it to doing just that topic.

:abstract 140 ~topicname

If you want to adjust output of yours that would be too long for something like a phone screen, you can ask :abstract to show you all rules whose output would likely exceed some limit (here 140). Again with a topic name restricts it to that topic and without the name it does the entire system.

:abstract why will look at your topics and tell you which gambits like a rejoinder with a why keyword.

:abstract censor ~bad_words will note all output which contains any words considered "bad" by many. Of course regular uses will also appear, like "juggling 4 balls at once" (balls being a bad word in some contexts). The censor command can look for any words referred to by the concept given.

Views over User Logs

If you get a lot of user logs (say thousands), reading through them becomes a chore. The logs have a bunch of excess information and are in a bunch of different files. This is where :trim comes in, making it easier to see things. :trim assumes all files in the LOGS directory are user logs and will process them in some manner. It will normally put its output in TMP/tmp.txt . At the end it will put how many users it saw, and who had the longest conversation with their volley count.

Some of these outputs can be usefully sorted and aggregated as described later.

:trim n

Trim will read every file and generate output depending on the integer code given it. The codes are:

0 - puts the what the user said, followed by what the chatbot said, on single lines, removing all the excess junk.

1 – similar to 0, but puts what the chatbot said first, and what the user said after. This is useful for seeing all the responses users have made and can be aggregated to figure out what clever rejoinders you might want.

2 – similar to 0 (user first), but puts the name of the topic the chatbot ended in before either. You can see the flow of topics better with this view.

3 – similar to 2 (topic shown), but puts what the chatbot said before the user.

4 – puts the user and chatbot on separate lines, indenting the chatbots line. Easier to read.

5 – similar to 3, but indents the user instead of the chatbot.

6 – only lists the users inputs. This is good for creating a file that can recreate a user's experience, if you want to recreate it for debugging or regression.

7 – shows topic, user and then bot, but only shows what the system thinks were questions by the user. Useful for seeing what users ask and what the chatbot responded. This can be aggregated to find out what the most common questions are, to decide which you might want to answer that you're not.

8 - reads top level file testinput.txt which is a list of words, phrases, or concepts one per line. It will then find all user inputs that match those and put out the topic, botline, userline. Concepts only match literally, not cannonically, so if swim is in a concept and swimming is in the input, they don't match. Good for seeing what bot response was to certain user inputs.

9 – like 8, reads matchers but applies them to bot output instead of user input. Outputs topic, then user input, then matching bot output. Good for seeing what user input caused matching bot output.

10- like 9, matching on bot output, but prints out the topic, prior bot output, user input, matching bot output. Good for seeing what bot said that triggered user into triggering bot.

+8 +9 +10   - similar to 8,9,10 but show the matching word in parens at the front of the output.

12 – the system reads all the user input words and counts how many times each word is seen. These are NOT the words the user typed, they are the words seen after input adjustments (including spell-checking, proper name merging, multiple-word substitutions, etc). Having done this, you can then use :worduse to get some statistics. This 12 command also outputs a map of the distribution of sentence lengths of all the inputs.

You can name the directory to use before the number code action. By default the system uses LOGS, but you can do otherwise. E.g.,

:trim c:\FULLLOGS 8

Hotspot Analysis

:trim 15

The interesting analysis is the done by :trim 15. This reads log files, noting what users have entered what topics and what top level rules were hit. This information is then transferred into an abstract (automatically running :abstract) so you can see the hit counts on every rule and topic. You can actually see the decay rate as users enter a topic and some number of them abandon it after each gambit. You can determine what are common responders or rejoinders, and which topics are heavily trafficked or simply never seen.

Aggregation

You can take various forms of the outputs from :trim and run a DOS sort on the file. If you use the outputs the prefix with topic names, you can get a sorted copy of what the bot said by topic and how users responded. Of course some users are going to say what others have said, and you don't want to read each of those. Hence, once you have sorted the file, you can run :aggregate on it.

:aggregate filename

will read the file and remove lines that immediately repeat. Instead of showing all the repetitions, it will put a repeat count on a line. This simplified your view significantly. The output goes into TMP/tmp.txt.

Word Frequency

:worduse word

You prepare for this by using :trim 12.

When word is an ordinary word or ~name (interjection) the system will tell you how many times that word showed up in input.

If word is >n where n is a number, the system will display the top n most common known words and their frequency.

If word is ?n where n is a number, the system will display the top n most common unrecognized words and their frequency. This is useful for seeing words you might want to add into substitutions to augment spelling correction.

Analyzing topics

You can learn how many gambits and responders each topic has, as well as get a histogram of how many topics are what gambit length by using:
:topicstats

Long topics (like beyond 80 gambits) may be unproductive because users won't get to end. Better to see if it can be split into subtopics that can be called from your original topic but which can stand on their own and be accessed independently.