ChatScript Manuel de Déboguage

© Bruce Wilcox, mailto:gowilcox@gmail.com www.brilligunderstanding.com
traduit par Mathieu Rigard pour Utopia-french.tech m.rigard@utopia-french.tech

Révision 11/19/2017 cs7.7

Vous avez rédiger un script. Ça ne marche pas. Maintenant il vous faut savoir pourquoi ? Vous devez le déboguer, le réparer et le recompiler. Le déboguage consiste principalement repérer ce que le système essaie de tester et à découvrir où il ne fait pas ce à quoi vous vous attendiez. Déboguer se fait le plus souvent en émettant des commandes vers moteur, par opposition à celles entrées dans le Chat.

Si le système détecte des bogues pendant son exécution, ils vont dans TMP/bugs.txt Vous pouvez effacer tout le contenu du répertoire TMP à tout moment. Mais la plupart du temps ce n'est pas votre problème.

Le débogage nécessite généralement l'utilisation de commandes :xxxx. Je ne me souviens pas toujours de toutes ces commandes, alors je lance :

`:commands`

pour obtenir une liste des commandes et une description approximative.

`:commands`

La déclaration ci-dessus affiche la liste suivante:

```
---- Debugging commands -
:do         - Execute the arguments as an output stream, e.g., invoke a function, set variables, etc
:silent     - toggle silent - dont show outputs
:log        - dump message into log file
:noreact    - Disable replying to input
:notime     - Toggle notiming during this topic
:notrace    - Toggle notracing during this topic
:redo       - Back up to turn n and try replacement user input
:retry      - Back up and try replacement user input or just redo last sentence
:say        - Make chatbot say this line
:skip       - Erase next n gambits
:show       - All, Input, Mark, Number, Pos, Stats, Topic, Topics, Why, Reject, Newlines
:time       - Set timing variable
              (all none prepare match ruleflow pattern query json macro user usercache sql tcp topic)
:trace      - Set trace variable
              (all none basic prepare match output pattern infer query substitute hierarchy fact
control topic pos)
:why        - Show rules causing most recent output
:authorize - Flip authorization for all debug commands

---- Fact info -
:allfacts  - Write all facts to TMP/facts.tmp
:facts     - Display all facts with given word or meaning or fact set
:userfacts - Display current user facts

---- Topic info -
:gambits    - Show gambit sequence of topic
:pending    - Show current pending topics list
:topicstats - Show stats on all topics or named topic or NORMAL for non-system topics
:topicinfo  - Show information on a topic
:topics     - Show topics that given input resonates with
:where      - What file is the given topic in
```

```
---- System info -
:commands        - Show all :commands
:context         - Display current context labels
:conceptlist     - Show all concepts- or with argument show concepts starting with argument
:definition      - Show code of macro named
:directories     - Show current directories
:functions       - List all defined system ^functions
:identify        - Give version data on this CS
:macros          - List all user-defined ^macros and plans
:memstats        - Show memory allocations
:list            - $ (variables) @ (factsets)  _  (match variables)  ^ (macros)  ~ (topics&concepts)
:queries         - List all defined queries
:timedfunctions  - List all user defined macros currently being timed
:timedtopics     - List all topics currently being timed
:tracedfunctions - List all user defined macros currently being traced
:tracedtopics    - List all topics currently being traced
:variables       - Display current user/sysytem/match/all variables
:who             - show current login/computer pair

---- Word information -
:down      - Show wordnet items inheriting from here or concept members
:concepts  - Show concepts triggered by this word
:findwords - show words matching pattern of letters and *
:hasflag   - List words of given set having or !having some system flag
:nonset    - List words of POS type not encompassed by given set
:overlap   - Direct members of set x that are also in set y somehow
:up        - Display concept structure above a word
:word      - Display information about given word
:dualupper - Display words that have multiple upper-case forms

---- System Control commands -
:build   - Compile a script - filename {nospell,outputspell,reset}
:bot     - Change to this bot
:crash   - Simulate a server crash
:debug   - Initiate debugger
:flush   - Flush server cached user data to files
:quit    - Exit ChatScript
:reset   - Start user all over again, flushing his history
:restart - Restart Chatscript
:user    - Change to named user, not new conversation

---- Script Testing -
:autoreply   - [OK,Why] use one of those as all input.
:common      - What concepts have the two words in common.
:prepare     - Show results of tokenization, tagging, and marking on a sentence
:regress     - create or test a regression file
:source      - Switch input to named file
:testpattern - See if a pattern works with an input.
:testtopic   - Try named topic responders on input
:verify      - Given test type & topic, test that rules are accessible.
               Tests: pattern (default), blocking(default), keyword(default), sample, gambit, all.

---- Document Processing -
:document - Switch input to named file/directory as a document {single, echo}
:wikitext - read wiki xml and write plaintext
:tsv    - convert a tab-delimited spreadsheet into CS table format, with double quotes around any
string needing it

---- Analytics -
:abstract     - Display overview of ChatScript topics
:coverage     - Save execution coverage of ChatScript rules
:showcoverage - Display execution coverage of ChatScript rules
:diff         - match 2 files and report lines that differ
:trim         - Strip excess off chatlog file to make simple file TMP/tmp.txt

---- internal support -
:topicdump         - Dump topic data suitable for inclusion as extra topics into TMP/tmp.txt
                     (:extratopic or PerformChatGivenTopic)
:builddict         -  basic, layer0, layer1, or wordnet are options instead of default full
:buildforeign      - rebuild a foreign dictionary (requires treetagger data)
:clean             - Convert source files to NL instead of CR/LF for unix
:extratopic        - given topic name and file as output from :topicdump,
                     build in core topic and use it thereafter
:pennformat        - rewrite penn tagfile (eg as output from stanford) as one liners
:pennmatch         - FILE {raw ambig} compare penn file against internal result
:pennnoun          - locate mass nouns in pennbank
:pos               - Show results of tokenization and tagging
:sortconcept       - Prepare concept file alphabetically (see writeup)
:translateconcept  - TODO
```

```
:timepos           - compute wps average to prepare inputs
:verifypos         - Regress pos-tagging using default REGRESS/postest.txt file or named file
:verifyspell       - Regress spell checker against file
:verifysubstitutes - Regress test substitutes of all kinds
:worddump          - show words via hardcoded test
:verifySentence    - verification data
```

Toutes les commandes peuvent être écrites en entier, ou la plupart peuvent être abrégées en utilisant: premièrelettredernièrelettre, ex :

:build

peut devenir

: bd

:commands off désactive localement les commandes jusqu'à ce qu'une commande :commands on soit donnée.

Avant qu’il y ait un problème lors de l'exécution

Avant de discuter avec votre Chatbot et de découvrir que les choses ne fonctionnent pas, il y a quelques choses à faire pour être averti des problèmes.

La compilation (`:build`)

Bien sûr, vous avez commencé avec :build pour compiler votre script. Il n'aurait pas laisser passé un script complètement faux, mais il a pu émettre des avertissements.

Ce n'est probablement pas votre problème, mais regardons ce qu'il pourrait vous avoir dit comme avertissement. Le système vous avertira pendant la compilation et résumera ses résultats à la fin de la compilation. Les messages de compilation se produisent à l'écran et dans le fichier journal.

Les avertissements les plus importants sont une référence à un ensemble ou à une étiquette ^reuse non définies. Par exemple.,

```
*** Warning- missing set definition ~car_names
*** Warning- Missing cross-topic label ~allergy.JOHN for reuse
```

Vous devriez les corriger parce qu'ils sont clairement des erreurs, bien que le script s'exécute bien partout sauf dans ces endroits.

```
*** Warning- flowglow is unknown as a word in pattern
```

Les avertissements concernant des mots dans les Modèles qu'il ne reconnaît pas ou qu’il connaît en minuscules alors que vous avez utilisé des majuscules peuvent être important ou pas. Aucun d'eux n'est particulièrement faux, à moins que vous ne l'ayez pas fait exprès. Les mots qu'il ne reconnaît pas apparaissent soit parce que vous avez fait une faute de frappe (vous avez besoin de le corriger) ou simplement parce que le mot n'est pas dans le dictionnaire.

Les mots en majuscules sont à nouveau des mots qu'il connaît en minuscules, mais vous les avez utilisés en majuscules. Peut-être juste ou faux.

L'édition du dictionnaire principal n'est pas une tâche pour les timides. Mais ChatScript maintient des dictionnaires secondaires dans le dossier TOPIC et ceux-ci sont faciles à ajuster. Pour les modifier, vous pouvez définir des concepts qui ajoutent des entrées locales de dictionnaire. Les mots importants comme bits sont dans src/dictionarySystem.h la bases c'est NOUN, VERB, ADJECTIVE et ADVERB.

```
concept: ~morenoun NOUN (fludge flwoa boara)
```

Un concept comme celui-ci, déclaré avant l'utilisation de ces mots, les définira dans le dictionnaire local en tant que noms et supprimera le message d'avertissement. Vous pouvez également être plus précis avec plusieurs indicateurs comme celui-ci:

```
concept: ~myverbs VERB VERB_INFINITIVE (spluat babata)
```

Vous pouvez définir des concepts au niveau 0 et/ou au niveau 1 du :build, de sorte que vous pouvez définir de nouveaux mots chaque fois que vous en avez besoin.

Lorsque le :build est terminé, la converstion reprendra là où elle s'était arrêtée avec l'utilisateur (ses fichiers de données sont inchangés). Si vous voulez effacer automatiquement l'utilisateur et recommencer, utilisez

```
:build xxx reset
```

Commandes de débogage les plus utiles

```
:prepare ceci est une entrée simple
```

Cela vous montre comment le système va marquer une phrase et à quels concepts elle correspond. C'est ce que le système fait pour se préparer à faire correspondre les sujets à vos commentaires. À partir de là, vous pouvez déduire quels modèles correspondraient et quels sujets pourraient être invoqués.

Si vous ne donnez pas d'arguments à préparer, cela active simplement une trace de préparation pour toutes les entrées futures qui désactive réellement la réponse. Ça n'est pas ce qu'on veut faire habituellement.

Il y a un premier argument facultatif à :prepare, qui est une variable. Si elle est donné, il indique la valeur $cs_token à utiliser pour la préparation. Par exemple

```
:prepare $mytoken this is a sentence.
```

La variable $cs_prepass contient un script optionnel qui sera exécuté avant l'exécution du script de contrôle principal.

L'avantage de $cs_prepass est qu'il est également exécuté par :prepare, vous pouvez donc en suivre les traces. Si vous voulez voir la préparation avec ou sans ce Topic (préparation brute par le moteur), vous pouvez ajouter l'argument optionnel NOPREPASS ou PREPASS. Par exemple :

```
:prepare NOPREPASS This is a sentence
:prepare $newtoken NOPREPASS This is a sentence.
```

L'utilisation du PREPASS ou NOPREPASS est mémorisée pendant l'exécution de CS, de sorte que la prochaine fois que vous appelez : prepare vous pouvez l'omettre et le même réglage sera utilisé. Si tout ce que vous voulez savoir est dans quel concepts tel mot est impliqué, vous utiliserez :common.

`:why`

Quand je teste mes bots (en supposant qu'ils passent la vérification), je discute avec eux jusqu'à ce que j'obtienne une réponse que je n'aime pas. Je lui demande alors pourquoi il a généré la réponse qu'il a faite.

Ceci spécifie les règles qui ont généré cette réponse et de quels Topic elle provient. La pluspart du temps je peux voir, en regardant la règle, pourquoi je ne voudrais pas que cela corresponde et ce que je dois faire pour corriger cette règle. Par contre cela ne résout pas la raison pour laquelle une règle que je veux voir correspondre a échoué, donc pour cela, j'ai besoin de traçage.

Trace

`:trace all` / `:trace always` / `:trace none`

La commande de débogage ultime restitue une trace de tout ce qui se passe pendant l'exécution sur l'écran et dans le fichier journal. Après l'avoir entré, vous tapez dans votre chat et regardez ce qui se passe (qui est également enregistré dans le fichier journal en cours). Le problème étant que c'est potentiellement une grande trace. Vous voudrez sans doute être plus précis dans votre entreprise.

Sinon,
`:trace all`
entraine tout le traçage. Il peut être supprimé dans les zones de code où vous avez spécifié `^NOTRACE()`.

`:trace always`
ignore la protection `^NOTRACE()` et trace tout.

`:trace ignorenotrace`
vous permet d'utiliser les traces limitées ci-dessous, tout en ignorant les appels couverts par `^NOTRACE()`.

`:trace ^myfunction`

Permet le traçage pour la fonction. Appelez-la à nouveau pour revenir à la normale.

`:trace $var`

cela permet le suivi lorsque cette variable change. Les variables locales ne montrent qu'une trace pendant la volée en cours. Les variables globales montrent une trace à travers les volées successives.

`:trace ~education`

Permet le suivi de ce Topic et tous les Topic qu'il appelle. Appelez-le à nouveau pour revenir à la normale. C'est probablement ce dont vous avez besoin habituellement pour voir ce qui s'est mal passé. Vous connaissez le code que vous souhaitez exécuter, surveillez donc le sujet dans lequel il se trouve.

`:trace !~education`

cela désactive le suivi actuel pour ce sujet et tous les sujets qu'il appelle quand `:trace all` est exécuté. Appelez-le à nouveau pour revenir à la normale.

De même, vous pouvez faire

`:trace ~education.4.5`

qui dit de tracer juste la 4ème règle de niveau supérieur dans l'éducation, le répondeur numéro 5. Les valeurs par défaut ci-dessus tracent tout tant qu'il reste dans la Topic. Vous pouvez spécifier ce que vous voulez tracer en nommant des valeurs de trace, puis le sujet.

`:trace prepare basic facts ~education`

Ce qui précède active un certain nombre de valeurs de trace, puis les affecte au sujet, en effaçant les valeurs de trace.

: trace input: trace prepare peuvent générer beaucoup de données sur les concepts. Si vous voulez juste une simple vérification de ce à quoi ressemble l'entrée correcte, vous pouvez faire :trace input`