ChatScript System Variables and Engine-defined Concepts

Copyright Bruce Wilcox, gowilcox@gmail.com www.brilligunderstanding.com
Revision 6/13/2022 cs13.2

- Engine-defined Concepts
- System Variables
- Control over Input
- Interchange Variables

Engine-defined concepts

In addition to concepts defined in script files, the system automatically defines a bunch of dictionary-based sets as well as dynamically computed concept members.

set	description
`~web_url`	word is a web url
`~email_url`	word is an email address
`~kindergarten`	word learned early in life
`~grade1_2`	word learned in these grades
`~grade3_4`	word learned in these grades
`~grade_5-6`	word learned in these grades. Unmarked words are learned even later
`~utf8`	word has nonascii characters
`~daynumber`	word could be a number of a day in a month
`~yearnumber`	word could be the number of a recent year
`~dateinfo`	phrase is month day year of some kind
`~kelvin`	temperature marker
`~celcius`	temperature marker
`~fahrenheit`	temperature marker
`~twitter_name`	twitter user name
`~hashtag_label`	twitter topic reference

Interjections, "discourse acts", and concept sets

Some words and phrases have interpretations based on whether they are at sentence start or not. E.g., *good day, mate* and *It is a good day* are different for *good day.*

Likewise *sure* and *I am sure* are different.

Words that have a different meaning at the start of a sentence are commonly called interjections.

In ChatScript these are defined by the `livedata/interjections.txt` file. In addition, the file augments this concept with “discourse acts”, phrases that are like an interjection. All interjections and discourse acts map to concept sets, which come thru as the user input instead of what they wrote.

For example *yes* and *sure* and *of course* are all treated as meaning the discourse act of agreement in the interjections file. So you don’t see *yes, I will go* coming out of the engine.

The interjections file will remap that to the sentence `~yes`, breaking off that into its own sentence, followed by *I will go* as a new sentence.

These generic interjections (which are open to author control via interjections.txt) are:

interjection	description
`~yes`	
`~no`	
`~emomaybe`	
`~emohello`	
`~emogoodbye`	
`~emohowzit`	
`~emothanks`	
`~emolaugh`	
`~emohappy`	
`~emosad`	
`~emosurprise`	
`~emomisunderstand`	
`~emoskeptic`	
`~emoignorance`	
`~emobeg`	
`~emobored`	
`~emopain`	
`~emoangry`	
`~emocurse`	
`~emodisgust`	
`~emoprotest`	
`~emoapology`	
`~emomutual`	

Because all interjections at the start of a sentence are broken off into their own sentence, this kind of pattern does not work:

```
u: (~yes _*)
```

You cannot capture the rest of the sentence here, because it will be part of the next sentence instead. This means interjections act somewhat differently from other concepts.

If you use a word in a pattern which may get remapped on input, the script compiler will issue a warning. Likely you should use the remapped name instead.

The following concepts are triggered by exactly repeating either the chatbot or oneself (to a repeat count of how often repeated). Repeats are within a recency window of about 20 volleys.

concept	description
`~repeatme`	
`~repeatinput1`	
`~repeatinput2`	
`~repeatinput3`	
`~repeatinput4`	
`~repeatinput5`	
`~repeatinput6`	

POS (Part of Speech) Tags

Words will have pos-tags attached, specifiying both generic and specific tag attributes, eg., `~noun`, `~noun_singular`.

Generic Specifics

nouns	description
`~noun`	
`~noun_singular`	
`~noun_plural`	
`~noun_proper_singular`	
`~noun_proper_plural`	
`~noun_gerund`	
`~noun_number`	
`~noun_infinitive`	
`~noun_omitted_adjective`	

verbs	description
`~verb`	
`~verb_present`	
`~verb_present_3ps`	
`~verb_infinitive`	
`~verb_present_participle`	

verbs	description
`~verb_past`	
`~verb_past_participle`	
`~aux_verb`	
`~aux_verb_present`	
`~aux_verb_past`	
`~aux_verb_future`	
`~aux_verb_tenses`	
`~aux_be`	
`~aux_have`	
`~aux_do`	

Auxilliary verbs are segmented into normal ones and special ones. Normal ones give their tense directly. Special ones give their root word. The tense of the be/have/do verbs can be had via `^properties()` and testing for verb tenses

adjectives	description
`~adjective`	
`~adjective_normal`	
`~adjective_number`	
`~adjective_noun`	
`~adjective_participle`	

adjectives in comparative form	description
`~more_form~most_form`	
`~adverb`	
`~adverb_normal`	

adverbs in comparative form	description
`~more_form~most_form`	
`~pronoun~pronoun_subject~pronoun_object`	
`~conjunction_bits~conjunction_coordinate~conjunction_subordinate`	
`~determiner_bits~determiner~pronoun_possessive~predeterminer`	
`~possessive`	covers *'* and *'s* at end of word
`~to_infinitive`	"to" when used before a noun infinitive
`~preposition~particle`	free-floating preposition tied to idiomatic verb

adverbs in comparative form	description
`~comma`	
`~quote`	covers ’ and “ when not embedded in a word
`~paren`	covers opening and closing parens
`~foreign_word`	some unknown word
`~there_existential`	the word there used existentially

In addition to normal generic kinds of pos tags, words which are serving a pos-tag role different from their putative word type are marked as members of the major tag they act as part of. E.g,

	description
`~noun_gerund`	verb used as a ~noun
`~noun_infinitive`	verb used as a ~noun
`~noun_omitted_adjective`	an adjective used as a collective noun (eg *the beautiful are kind*)
`~adjectival_noun`	noun used as adjective like bank “bank teller”
`~adjective_participle`	verb participle used as an adjective

For `~noun_gerund` in *I like swimming* the verb gerund *swimming* is treated as a noun (hence called noun-gerund) but retains verb sense when matching keywords tagged with part-of-speech (i.e., it would match `swim~v` as well as `swim~n`).

Additionally, there is

	description
`~number`	is not a part of speech, but is comprise of `~noun_number` (a normal number value like *17* or *seventeen*)
`~adjective_number`	also a normal numeral value and also `~placenumber`) like *first*.
`~integer`	
`~float`	
`~positiveinteger`	

	description
`~negativeinteger`	
`~modelnumber`	not a true number, but a word with both alpha and numeric
`~filename`	looks like a filename with extension
`~modelnumber`	not a true number, but a word with both alpha and numeric
`~prep_phrase`	chunk of text forming a prepositional phrase
`~verb_phrase`	chunk of text forming a verb phrase
`~noun_phrase`	chunk of text forming a noun phrase

"To" can be a preposition or it can be special. When used in the infinitive phrase To go, it is marked `~to_infinitive` and is followed by `~noun_infinitive`.

	description
`~verb_infinitive`	refers to a match on the infinitive form of the verb (*I hear John sing* or *I will sing*).
`~There_existential`	refers to the use of where not involving location, meaning the existence of, as in There is no future.
`~Particle`	refers to a preposition piece of a compound verb idiom which allows being separated from the verb. If you say *I will call off the meeting*, call_off is the composite verb and is a single token. But if you split it as in *I will call the meeting off*, then there are two tokens. The original form of the verb will be call and the canonical form of the verb will be call_off, while the free-standing off will be labeled `~particle`.

	description
`~verb_present`	will be used for normal present verbs not in third person singular like *I walk* and
`~verb_present_3ps`	will be used for things like *he walks*
`~possesive`	refers to *'s* and *'* that indicate possession, while possessive pronouns get their own labeling `~pronoun_possessive`.
`~pronoun_subject`	is a pronoun used as a subject (like *he*)
`~pronoun_object`	refers to objective form like *him*

Individual words serve roles in the parse of a sentence, which are retrievable. These include:

	description
`~mainsubject`	
`~mainverb`	
`~mainindirect`	
`~maindirect`	
`~subject2`	
`~verb2`	
`~indirectobject2`	
`~object2`	
`~subject_complement`	adjective object of sentence involving linking verb
`~object_complement`	2ndary noun or infinitive verb filling modifying mainobject or object2
`~conjunct_noun~conjunct_verb~conjunct_adjective~conjunct_adverb~conjunct_phrase~conjunct_cla`	
`~postnominalAdjective`	adjective occuring AFTER the noun it modified
`~reflexive`	reflexive pronouns
`~not`	
`~address`	noun used as addressee of sentence
`~appositive`	noun restating and modifying prior noun

	description
`~absolutephrase`	special phrase describing whole sentence
`~omittedtimeprep`	modified time word used as phrase but lacking preposition (*Next tuesday I will go*)
`~phrase`	a prepositional phrase start (except
`~clause`	a subordinate clause start
`~verbal`	a verb phrase

and special concepts: | `~capacronym` | word is in all caps (and &) and is likely an acronym | `~emoji` | word starts and end with : and represents an emoji

Spanish

For Spanish (if you are in spanish language mode) there is ~spanish_he, ~spanish_she, ~spanish_singular, ~spanish_plural for nouns and adjectives and determiner 'the'. Pronouns will be marked with ~pronoun_object_singular or ~pronoun_object_plural or ~pronoun_object_you. Also ~pronoun_indirectobject_singular and ~pronoun_indirectobject_plural and ~pronoun_indirectobject_you. Also ~pronoun_I and ~pronoun_you. And simple future tense verbs will be marked ~spanish_future.

System Variables

The system has some predefined variables which you can generally test and use but not normally assign to. These all begin with `%` . Ones that are reasonable to set are written in bold underline. Boolean values are always `1` or `null` on returns. `1` or `0` if you are setting them.

Date & Time & Numbers

variable	description
`%date`	one or two digit day of the month
`%day`	Sunday, etc
`%daynumber`	1-7 where 1 = Sunday
`%fulltime`	seconds representing the current time and date (Unix epoch time)
`%fullmstime`	Numeric full time/date in milliseconds (Unix epoch time)
`%hour`	0-23

variable	description
`%timenumbers`	completely consistent full time info in numbers that you can do `_0 = ^burst(%timenumbers)`to get `_0` =seconds (2digit) `_1`=minutes (2digit) `_2`=hours (2digit) `_3`=dayinweek(0-6 Sunday=0) `_4`=dateinmonth (1-31) `_5`=month(0-11 January=0) `_6`=year.You need to get it simultaneously if you want to do accurate things with current time, since retrieving %hour %minute separately allows time to change between calls
`%leapyear`	boolean if current year is a leap year
`%daylightsavings`	boolean if current within daylight savings
`%minute`	0-59
`%month`	1-12 (January = 1)
`%monthname`	January, etc
`%second`	0-59
`%volleytime`	number of seconds of computation since volley input started
`%time`	hh:mm in military 24-hour time
`%zulutime`	2016-07-27T11:38:35.253Z
`%week`	1-5 (week of the month)
`%year`	e.g., 2011
`%rand`	get a random number from 1 to 100 inclusive

Time and date information are normally local, relative to the system clock of the machine CS is running on. See $cs_utcoffset for adjusting time based on relationship to utc (e.g your server is in Virginia and you are in Colorado).

%rand is only pseudo-random. A specific username is assigned a seed based on their name. Thereafter the seed evolves by the dialog but it is repeatable when the same user starts over again. If you want truly random, use %fullmstime % $howmany to get range 0 .. $howmany-1

User Input

variable	description
`%bot`	current bot responding
`%revisedinput`	Boolean is current input from `^input` not direct from user
`%command`	Boolean was the user input a command

variable	description
`%foreign`	Boolean is bulk of the sentence composed of foreign words
`%impliedyou`	Boolean was the user input having you as implied subject
`%impliedsubject`	Boolean was the user input having an implied subject (not you, usually I)
`%input`	the count of the number of volleys this user has made ever
`%volley`	sae as %input, the count of the number of volleys this user has made ever
`%ip`	ip address supplied
`%myip`	ip address of cs server responding
`%language`	current dictionary language
`%length`	the length in tokens of the current sentence
`%more`	Boolean is there another sentence after this
`%morequestion`	Boolean is there a `?` or question word in the pending sentences
`%originalinput`	all sentences user passed into volley, before adjusted in any way except OOB data is stripped off

variable	description
`%originalsentence`	the current sentence after tokenization but before any adjustments
`%parsed`	Boolean was current input parsed successfully
`%question`	Boolean was the user input a question - same as `?` in a pattern
`%quotation`	Boolean is current input a quotation
`%sentence`	Boolean does it seem like a sentence (subject/verb or command)
`%tableinput`	current line being executed in a table expansion during script compilation
`%tense`	past , present, or future simple tense (present perfect is a past tense)
`%user`	user login name supplied
`%userfirstline`	value of `%input` that is at the start of this conversation start
`%speaker`	value of `speaker` from a conversation involving :tsvsource
`%userinput`	Boolean is the current input from the user (vs the chatbot)
`%voice`	active or passive on current input
`%trace_on`	Fake empty variable used to turn on tracing (see Debugging commands)

variable	description
`%trace_off`	Fake empty variable used to turn off tracing (see Debugging commands)
`%starttimems`	Start of user request time/date in milliseconds
`%inputsize`	gives how many characters were passed in input
`%inputlimited`	1 if too many characters were given (relative to fullinputlimit)
`%tsvsource`	1 if in progress Null otherwise
`%heapsize`	how many bytes of heap are left

Chatbot Output

variable	description
`%inputrejoinder`	rule tag of any pending rejoinder for input or null if none pending
`%lastoutput`	the text of the last generated response for the current volley - always null across volleys
`%lastquestion`	Boolean did last output end in a ?
`%outputrejoinder`	rule tag if system set a rejoinder for its current output or 0
`%response`	number of committed responses that have been generated for this sentence (see Advanced User- Advanced Output: Committed Responses

System variables

Note for all time variables, they normally use local machine time. If you have a $cs_utcoffset variable with a value, then all time will be relative to GMT/UTC/Zulu (which means it doesn't pay attention to daylight savings and you have to do that yourself with the answer).

variable	description
`%all`	Boolean is the :all flag on? (:all to set)
`%document`	Boolean is :document running
`%fact`	Numeric value most recent fact id
`%freetext`	kb of available text space
`%freedict`	number of unused dictionary words
`%freefact`	number of unused facts
`%maxmatchvariables`	highest number of match variables, currently 20
`%maxfactsets`	highest number of @factsets, currently 20
`%host`	name of the current host machine or "local"
`%regression`	Boolean is the regression flag on
`%server`	Boolean is the system running in server mode
`%rule`	get a tag to the current executing rule. Can be used in place of a label
`%topic`	name of the current "real" topic . if control is currently in a topic or called from a topic which is not system or nostay, then that is the topic. Otherwise the most recent pending topic is found
`%actualtopic`	literally the current topic being processed (system or not)
`%trace`	Numeric value of the trace flag (:trace to set)
`%httpresponse`	return code of most recent ^jsonopen call (see below)
`%pid`	Linux process id or 0 for other systems
`%restart`	You can set and retrieve a value here across a system restart.
`%timeout`	Boolean tells if a timeout has happened, based on the timelimit command line parameter
`%lastcurltime`	Time Analysis: Name Look up: Host/proxy connect: App(SSL) connect: Pretransfer: Total Transfer:
`%crosstalk`	4k buffer in server visible between users to pass data back and forth

variable	description
`%crosstalk1`	4k buffer in server visible between users to pass data back and forth
`%crosstalk2`	4k buffer in server visible between users to pass data back and forth
`%crosstalk3`	4k buffer in server visible between users to pass data back and forth
`%logging`	bit status of serverLog, userLog, and host name - 0=off 1=file 2= stdout 4=stderr 8=prelog)
`%forkcount`	number of forks requested in linux evserver environment
`%dbparams`	copy of the server params given to db used as fileserver (pg or mysql or mssql or mongo)
`%botid`	bot id number in use
`%curlversion`	curl version information
`%dbversion`	db version information
`%testpattern`	The index number in the array of patterns of current pattern being matched in ^testpattern

%httpresponse returns the official http response codes when it succeeds in connecting to a server. When it fails, it returns various negative codes that are specific to curl.

```
-1 timeout - connection attempt was canceled or no time was allowed (instant fail)
-2 couldn't connect or not resolve host or proxy
-3 unsupported protocol
-4 curl got nothing (typically sent http to https site)
-5 malformed url
-6 other
-7 curl operation timeout
```

^testpattern control variables

`%testpattern-prescan` | execute this pattern on all sentences before doing other patterns one-by-one
`%trace_on` | starting here, do :trace pattern in ^testpattern
`%trace_on all` | starting here, do :trace all in ^testpattern
`%trace_off` | turn off tracing (also turns off at end of cs call)

Build data

variable	description
`%dict`	date/time the dictionary was built
`%engine`	date/time the engine was compiled
`%os`	os invovled (linux windows mac ios)
`%script`	date/time build1 was compiled
`%version`	engine version number

You actually can assign to any of them. This will override them and make them return what you tell them to and is a particularly BAD thing to do if this is running on a server since it affects all users (unless you reset the variable at the end of the volley. Assigning a period to a variable resets it).

Typically one does this as a temporary assignment in a `#!` comment line to set up conditions for testing using `:verify`.

Making them return a new value is NOT the same thing as making the engine have a different value. Unless the variable is marked as settable, setting a value affects only the value returned by a future call to the system variable. It does not change engine values the variable is meant to reflect.

Control Over Input

The system can do a number of standard processing on user input, including spell correction, proper-name merging, expanding contractions etc. This is managed by setting the user variable `$cs_token`.

The default $cs_token that comes with Harry is:

```
$cs_token = #DO_INTERJECTION_SPLITTING |
            #DO_SUBSTITUTE_SYSTEM |
            #DO_NUMBER_MERGE |
            #DO_PROPERNAME_MERGE |
            #DO_SPELLCHECK |
            #DO_PARSE
```

The `#`signals a named constant from the `dictionarySystem.h` file. One can set the following:

These enable various LIVEDATA files to perform substitutions on input:

flag	description
`#DO_ESSENTIALS`	perform `LIVEDATA/systemessentials` which mostly strips off trailing punctuation and sets corresponding flags instead
`#DO_SUBSTITUTES`	perform `LIVEDATA/substitutes`

flag	description
`#DO_CONTRACTIONS`	perform `LIVEDATA/contractions`, expanding contractions
`#DO_INTERJECTIONS`	perform `LIVEDATA/interjections`, changing phrases to interjections
`#DO_BRITISH`	perform `LIVEDATA/british`, respelling brit words to American
`#DO_SPELLING`	performs the `LIVEDATA/spelling` file (manual spell correction)
`#DO_TEXTING`	performs the `LIVEDATA/texting` file (expand texting notation)
`#DO_SUBSTITUTE_SYSTEM`	do all LIVEDATA file expansions

The contents of the files above are pairs of tokens per line. Left is the word to replace and right is the replacement. When multiple words are involved, the left side uses underscores to represent this and the right side uses `+`. If the right side is missing, it means just delete. | `#DO_INTERJECTION_SPLITTING` | break off leading interjections into own sentence | `#$DO_NUMBER_MERGE` | merge multiple word numbers into one (*four and twenty*)
| `#$DO_PROPERNAME_MERGE` | merge multiple proper name into one (*George Harrison*) | `#DO_DATE_MERGE` | merge month day and/or year sequences (*January 2, 1993*) | `#JSON_DIRECT_FROM_OOB` | asking the tokenizer to directly process OOB data. See `^jsonparse` in JSON manual. | `#NO_FIX_UTF` | do not adjust inputs with html or utf8 encodings to simple ascii.
| `#TOKENIZE_BY_CHARACTER` | Every non-whitespace character becomes its own token and canonical form. (good for Japanese)

If any of the above items affect the input (except TOKENIZE_BY_CHARACTER), they will be echoed as values into `%tokenFlags` so you can detect they happened. The next changes do not echo into %tokenFlags and relate to grammar of input:

flag	description
`DO_POSTAG`	allow pos-tagging (labels like ~noun ~verb become marked)
`DO_PARSE`	allow parser (labels for word roles like ~main_subject)
`DO_CONDITIONAL_POSTAG`	perform pos-tagging only if all words are known. Avoids wasting time on foreign sentences in particular
`NO_CONDITIONAL_IDIOM`	will not perform substitutions in the dictionary which are considered conditional idioms
`NO_ERASE`	where a substitution would delete a word entirely as junk, don't

flag	description
`DO_SPLIT_UNDERSCORES`	happens after all other input tokenization and adjustments except number merge, and separates words that have been conjoined either because the dictionary has them (*credit_card*) or because they were merged by proper name merging, or by substitution. The result is only words without underscores (excluding number words like *five_thousand_and_four*
`MARK_LOWER`	if a word is considered a proper name in CS and is marked as an upper case word, this will force it to perform any markings for its lower case form as well. Sometimes users type stuff in upper case that really should be lower

Normally the system tries to outguess the user, who cannot be trusted to use correct punctuation or casing or spelling. These block that:

flag	description
`STRICT_CASING`	except for 1st word of a sentence, assume user uses correct casing on words
`NO_INFER_QUESTION`	the system will not try to set the QUESTIONMARK flag if the user didn't input a ? and the structure of the input looks like a question
`DO_SPELLCHECK`	perform internal spell checking
`ONLY_LOWERCASE`	force all input (except "I") to be lower case, refuse to recognize uppercase forms of anything
`NO_IMPERATIVE`	
`NO_WITHIN`	don't match fragments within a composite word
`NO_SENTENCE_END`	do not break input into sentences

Normally the tokenizer breaks apart some kinds of sentences into two. These prevent that:

flag	description
`NO_COLON_END`	don't break apart a sentence after a colon
`NO_SEMICOLON_END`	don't break apart a sentence after a semi-colon
`UNTOUCHED_INPUT`	if set to this alone, will tokenize only on spaces, leaving everything but spacing untouched

flag	description
`LEAVE_QUOTE`	if input is found within ” ” it will become a single token exactly as it is seen. W/o Leave_Quote, it is converted into a word without quotes and using underscores instead of spaces. So “My Fair Lady” becomes My_Fair_Lady, which would match a movie title if you had one, unlike *My Fair Lady* becoming the resulting token and unrecognized
`SPLIT_QUOTE`	if input is found within ” ” the quotes will be removed.

Note

you can change `$cs_token` on the fly and force input to be reanalyzed via `^retry(SENTENCE)`. I do this when I detect the user is trying to give his name, and many foreign names might be spell-corrected into something wrong and the user is unlikely to misspell his own name.

Just remember to reset `$cs_token` back to normal after you are done. Here is one such way, assuming `$stdtoken` is set to your normal tokenflags in your bot definition outputmacro:

```
#! my name is Rogr
s: (name is _*)

    if ($cs_token == $stdtoken)
        {
        $cs_token = #DO_INTERJECTION_SPLITTING |
                    #DO_SUBSTITUTE_SYSTEM | #DO_NUMBER_MERGE |
                    #DO_PARSE
        retry(SENTENCE)
        }
    _0 is the name.
    $cs_token = $stdtoken
```

If you type *my name is Rogr* into a topic with this, the original input is spell-corrected to *my name is Roger*, but this will change the `$cs_token` over to one without spell correction and redo the sentence, which will now come back with *my name is Rogr* and be echoed correctly, and `$cs_token reset`.

That’s assuming nothing else would run differently and trap the response elsewhere. If you were worried about that, it would be possible for the script to save where it is using `^getrule(tag)` and modify your control script to return immediate control to here after input processing if you had changed `$cs_token`.

%tokenflags

These are the values that %tokenflags may have after analysis of a sentence...
#define PRESENT 0x0000000000002000ULL
#define PAST 0x0000000000004000ULL // basic tense- both present perfect and past perfect map to it #define FUTURE 0x0000000000008000ULL
#define PRESENT_PERFECT 0x0000000000010000ULL // distinguish PAST PERFECT from PAST PRESENT_PERFECT #define CONTINUOUS 0x0000000000020000ULL
#define PERFECT 0x0000000000040000ULL
#define PASSIVE 0x0000000000080000ULL

#define IMPLIED_SUBJECT
#define QUESTIONMARK
#define EXCLAMATIONMARK
#define PERIODMARK
#define USERINPUT
#define COMMANDMARK
#define IMPLIED_YOU
#FOREIGN_TOKENS
#FAULTY_PARSE
#QUOTATION
#NOT_SENTENCE

One or more of these will be set if input was changed do to use of these files

```
#DO_ESSENTIALS
#DO_SUBSTITUTES
#DO_CONTRACTIONS
#DO_INTERJECTIONS
#DO_BRITISH
#DO_SPELLING
#DO_TEXTING
#DO_NOISE
#DO_PRIVATE
#DO_NUMBER_MERGE
#DO_PROPERNAME_MERGE
#DO_SPELLCHECK
#DO_INTERJECTION_SPLITTING
```

Private Substitutions

While in general, substitutions are defined in the LIVEDATA folder, you can define private substititions for your specific bot using the scripting language. You can say

```
replace: xxx yyyyy
```

which defines a substitution just like a livedata substitution file. It actually creates a substitution file called `private0.txt` or `private1.txt` in your TOPIC folder.

Even then, those substitutions will not be enacted unless you explicitly add to the `$cs_token` value `#DO_PRIVATE`, eg

```
$cs_token = #DO_INTERJECTION_SPLITTING |
            #DO_SUBSTITUTE_SYSTEM |
            #DO_NUMBER_MERGE |
            #DO_PROPERNAME_MERGE |
            #DO_SPELLCHECK |
            #DO_PARSE |
            #DO_PRIVATE
```

The left side of the substitution pair is case insensitive (matches either case on input) and can be placed in double-quotes (which converts spaces to underscores internally).

The right side of the substitution pair is case sensitive and can be placed in double-quotes (which converts spaces to plus signs internally).

Note: if you privately define a substitution that leads to a known interjection, it will be treated as an interjection, marked as DO_INTERJECTIONS rather than DO_PRIVATE. Interjections do not perform an actual substitution, does not replace the words on the left with the interjection concept name on the right. Instead interjections merely mark the phrase as being a member of that concept, leaving the actual words unchanged.

Similarly while canonical values of words can be defined in `LIVEDATA/SYSTEM/canonical.txt`, you can define private canonical values for your bots by using the scripting language. You can say:

```
canon: oh 0
canon: faster fast
```

which defines new canonical values for things and creates a file `canon0.txt` or `canon1.txt` in your TOPIC folder.

You can optionally add MORE_FORM or MOST_FORM as a 3rd argument, to set those flags for adjectives and adverbs.

If you want to set a canonical pair from a table during compilation, you can use a function to do the same thing (but only 1 pair at a time).

```
^canon(word canonicalform)
```

Numeric Substitutions

A special kind of private substitution (equally applicable in regular substitution files) is the numeric substitution.

```
replace: ?_km kilometers
```

The ?_ matches a digit number followed immediately by km, like `1.2km` and will separate the number and replace the units with the given replacement. The input can be singular or have an 's' like `10.5dollars`. And it can be with or without abbreviation periods, like `10kps` or `10k.p.s`

Apostrophe Substitutions replace

```
replace: 'xxx  yyy
```

allows you to split during tokenization any word followed by 'xxx into two words, original sans 'xxx and yyy. eg

```
replace: 've have
```

gives "companies've =>"companies have".

Replacing to a word with + in it

Normally `replace:  x  y+z` will generate 2 words, y and z. If you need a plus in your word, you can escape your 2nd word:

```
    replace: "black and decker" \BLACK+DECKER
```

Advanced replace substitution

You can name a pattern (which can extend over multiple lines) that can conditionally change the matched word into any other word or remove it or do nothing. Matching starts with _0 having been assigned to the location of the word/phrase to replace.

```
    replace: bubble_tea  ([
        (is $$cs_replace:=2)
        (has $$cs_replace:=null)
        (@_0- *~2 my  $$cs_replace:=1)
    ])
    "bubble tea is" -> 2 is
    "bubble tea has" -> has
    "my green bubble tea loves" -> my green 1 loves
```

You cannot use concepts in these patterns, nor the canonical forms of words. Your replacement data must be only tokens potentially separated by +, and potentially having _ in them. Do not use double quotes.

Interchange Variables

The following variables can be defined in a script and the engine will react to their contents.

interchange variable	description
`$cs_token`	described extensively above
`$cs_response`	controls automatic handling of outputs to user. see flags list below.
`$cs_crashmsg`	in server mode, what to say if the server crashes and we return a message to the user. By default the message is *Hey, sorry. I forgot what I was thinking about.*
`$cs_abstract`	used with :abstract
`$cs_trace`	if this variable is defined, then whenever the user's volley is finished, the value of this variable is set to that of :trace and :trace is cleared to 0, but when the user is read back in, the :trace is set to this value. For a server, this means you can perform tracing on a user w/o making all user transactions dump trace data
`$cs_control_pre`	name of topic (flag it SYSTEM) to run in gambit mode on pre-pass, set by author. Runs before any sentences of the input volley are analyzed. Good for setting up initial values

interchange variable	description
`$cs_usermessagelimit`	max number of message pairs (user input & bot output) saved in topic file
`$cs_externaltag`	name of a topic to use to replace existing internal English pos-parser. See bottom of ChatScript PosParser manual for details

interchange variable	description
`$cs_prepass`	name of a topic (mark it SYSTEM) to run in responder mode on main volleys, which runs before $cs_control_mainandafteralloftheaboveandpos-$ $parsingisdone.Usedtoamendpreparationdatacomin$ `name of topic (flag it SYSTEM) to run in responder mode on main volleys, set by author |` |$cs_control_post`\|nameoftopic(flagitSYSTEM)toru$ $pass,setbyauthor\|\|$`botprompt|` `message for console window to label bot output |` |$userprompt`\|messageforconsolewindowtolabeluse$ `message to use if a crash occurs. see also $cs_crash\|` \|$cs_crash| `topic to execute in gambit mode if a crash occurs. see also $cs_crashmsg\|` \|$cs_language`|ifspanish,willadjustspellcheckingfo$ `bits controlling how the tokenizer works. By default when null, you get all bits assumed on. The possible values are in src/dictionarySystem.h (hunt for $token) and you put a # in front of them to generate that named numeric constant \|` \|$cs_abstract`|topicusedby:$ $abstracttodisplayfactsifyouwantthemdisplayed\|\|$`c `topic used between parsing and running user control script. Useful to`

interchange variable	description
`$$tcpopen_error`	error message from a tcpopen error
`$$document`	name of the document being read in document mode
`$cs_randindex`	current value of the random generator value
`$cs_bot`	name of the bot currently in use
`$cs_login`	login name of the user
`$$csmatch_start`	start of found words from ˆmatch
`$$csmatch_end`	end of found words from ˆmatch
`$cs_fullfloat`	if defined, causes the system to generate full float 64-bit precision on outputs, otherwise you get 2 digit precision by default
`$cs_botid`	when non-zero creates facts and functions restricted by this bitmask so facts and functions created by other masks cannot be seen. allows you to separate facts and functions per bot in a multi-bot environment. During compilation if this is set by a bot: command, then functions created and facts created by tables will be restricted to that owner.

interchange variable	description
`$cs_numbers`	if defined, causes the system to output numbers in a different language style: french, indian. All other values are english.
`%trace_on and %trace_off`	Pseudo system variable used by the ˆtestpattern and ˆtestoutput call to let code request a trace be returned.
`$cs_indentlevel`	controls indenting when tracing in ˆtestpattern. 3 is a good number usually
`$indentlevel`	deprecated form of $cs_indentlevel$||‘cs_tracetestoutput`| set to 1 to force tracing in ˆtestoutput| |`$cs_outputlimit$‘|$Generatingmoreoutputthanthiswil$ `After volley prints to terminal milliseconds of time used in preparation, rules, postprocessing | |`$cs_showtime$‘|$Aftervolleyprintstoterminalmillise$ `set to 1, treat user as always new (don't try to read topic file) | |`cs_jid‘|$numbertostartwithwhenstartingindexingo$

By default $cs_response consists of the equivalent of:

```
$cs_response = #Response_upperstart | #response_removespacebeforecomma | #response_alterunde
```

If you want none of theses, use $cs_response = 0 (all flags turned off). See

^print for explanation of flags. `#response_noconvertspecial` - leave escaped n r and t alone in output and ^log, `#response_upperstart` - makes the first letter of an output sentence capitalized, `#Response_removespacebeforecomma` - does the obvious, `#Response_alterunderscores` - converts single underscores to spaces and double underscores to singles (eg for a web url) |

hook functions

`$cs_beforereset` | if set to a topic, will be executed before :reset is executed |
`$cs_addresponse` | provides a function name hook onto the output q to the user. |
`$testpatternpretopic` | execute this topic to preprocess input before matchines |
`$$cs_testpatterninput` | a copy of user input created by engine for $testpatternpretopic to change if it wants |
`$testpattern_posttopic` | can name a topic to be executed after ^testpattern to alter returned new variables |

variables to limit effort

`$cs_topicretrylimit` | if defined changes how many times you can pass back RETRY_TOPIC before it fails (current limit is 30) |
`$$topic_retry_limit_exceeded` | set if topic retry limit is encountered |
`$cs_userhistorylimit` | if not null, indicates how many volleys back are tracked as what was said by both parties |
`$cs_sentences_limit` | after this many sentences in volley, cs ignores the rest (default 50) |
`$cs_inputlimit` | Restrict user input size (excluding oob) |
`$cs_looplimit` | loop() defaults to 1000 iterations before stopping. You can change this default with this |
`$cs_analyzelimit` | in non-standalone mode, after this millisecond limit, cs stops NL analysis of more sentences |
`$cs_analyzelimitlog` | if analyzelimit triggers, report this fact in bug log |
`$FakeTimeOffset` | For testing analyzelimit, pretend this much ms has already lapsed on start |
`$cs_badspellLimit` | x-y format. After x many spelling corrections or x/y ratio of badspells to words seen, stop spellchecking |
`$cs_sequence` | How many words in sequence to check as a composite (default: 5) |

JSON variables

`$cs_jsontimeout` | seconds before JsonOpen declares a time out failure. If unspecified the default is 300 |

`$cs_saveusedJson` | if not null, the only JSON facts CS will write into the user's topic files that are referred to (directly or indirectly) from user variables being saved. (see below) |
`$cs_proxycredentials` | See ^JSONOPEN in JSON manual|
`$cs_proxyserver` | See ^JSONOPEN in JSON manual|
`$cs_proxymethod` | See ^JSONOPEN in JSON manual|
`$correlation_id` | See ^JSONOPEN in JSON manual|

Mongo variables

`$cs_mongoqueryparams` | set as a json structure of move its fields to a mongo query |
`$mongo_enable_ssl` | if set to true, will use ssl |
`$mongosslcafile` | data for ssl |
`$mongosslpemfile` | data for ssl |
`$mongosslpempwd` | data for ssl |
`$mongovalidatessl` | data for ssl |
`$mongo_timeexcess` | if certain operations exceed this ms, log entry is created |
`$$mongo_error` | error message if db not openable |

Note for %trace_on and %trace_off - you can use the command line parameter `blockapitrace` to prevent tracing in any code you accidentally leave in place.

`$cs_saveusedJson` exists as a kind of garbage collection. Nowadays most facts will come from JSON data either from a website or created in script. But keeping on top of deleting obsolete JSON may be overlooked. When this variable is non-null, ChatScript will automatically destroy any JSON fact that cannot trace a JSON fact path back to some user variable. Variables that have as values the name of a JSON object or array automatically protect all JSON facts underneath. JSON references merely within some text string will not protect anything, nor will references from some other non-JSON fact.

`$cs_inputlimit=x:y` for excessively long user input (excluding oob portion), the input will be truncated by keeping the first x characters and the last y characters.

`$cs_crash` - This topic can generate an appropriate dummy output and CS completes that volley but does not save an updated user file. The NEXT volley coming in will force cs to completely reload itself before processing. Making a dummy output hopefully means the same fatal input will not be sent back into CS to crash it again (due to external retry when no answer is received from CS). E.g.,

```
topic: ~crashtopic system ()
    t: Huh?
```

`$cs_addresponse` names a function of 2 arguments that will be called when CS wants put text into the output queue of the user. The first argument will be what CS wants to output. The second is the rule tag that generated this output. If the function returns a failure code, the message will be aborted and not put into the queue. If the function returns a text value (not null) then that message will replace what was intended to go to the user.