ChatScript System Variables and Engine-defined Concepts

© Bruce Wilcox, gowilcox@gmail.com brilligunderstanding.com

Revision 4/10/2016 cs6.3

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.

~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:

~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.

~repeatme, ~repeatinput1, ~repeatinput2, ~repeatinput3, ~repeatinput4, ~repeatinput5, ~repeatinput6,

POS (Part of Speech) Tags

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

<u>Genric Specifics</u>

~noun ~noun_singular, ~noun_plural, ~noun_proper_singular, ~noun_proper_plural, ~noun_gerund, ~noun_number, ~noun_infinitive. ~noun_omitted_adjective

~verb ~verb_present, ~verb_present_3ps, ~verb_infinitive, ~verb_present_participle, ~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

~adjective ~adjective_normal, ~adjective_number, ~adjective_noun, ~adjective_participle

Adjectives in comparative form will also have ~more_form or ~most_form.

~adverb ~adverb_normal

Adverbs in comparative form will also have ~more_form or ~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)

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

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.

~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.

~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)* while pronoun_object refers to objective form like (*him)*

Individual words serve roles in the parse of a sentence, which are retrievable. These include
~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_clause, ~conjunct_sentence
~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
~absolutephrase – special phrase describing whole sentence
~omittedtimeprep – modified time word used as phrase but lacking preposition (*Next t**uesday* I will go)
~phrase – a prepositional phrase start (except
~clause – a subordinate clause start
~verbal – a verb phrase

%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

%date – one or two digit day of the month
%day -Sunday, etc
%daynumber – 0-6 where 0 = Sunday

%fulltime – seconds representing the current time and date (Unix epoch time)
%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
%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).

User Input

%bot – current bot responding
%revisedinput – Boolean is current input from ^input not direct from user
%command – Boolean was the user input a command
%foreign – Boolean is bulk of the sentence composed of foreign words
%impliedyou – Boolean was the user input having you as implied subject
%input – the count of the number of volleys this user has made ever
%ip – ip address supplied
%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
%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)
%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
%userinput – Boolean is the current input from the user (vs the chatbot)
%voice – active or passive on current input

Chatbot Output

%inputrejoinder- rule tag of any pending rejoinder for input or 0 if none
%lastoutput – the text of the last generated response for the current volley
%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 responses that have been generated for this sentence

System variables

%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)

Build data:

%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 one 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:
DO_ESSENTIALS - perform LIVEDATA/systemessentials which mostly strips off trailing punctuation and sets corresponding flags instead.
DO_SUBSTITUTES – perform LIVEDATA/substitutes
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
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”)

If any of the above items affect the input, 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:
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_ERASE – where a substitution would delete a word entirely as junk, don't.

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

Normally the tokenizer breaks apart some kinds of sentences into two. These prevent that:
NO_HYPHEN_END – don't break apart a sentence after a hyphen
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.

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.

Private Substitutions

While in general, substitutions are defined in the LIVEDATA folder, you can define private substitutions 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

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 faster fast

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

Interchange Variables

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

$cs_token – described extensively above

$cs_response – controls automatic handling of outputs to user. By default it consists of
$cs_response = #Response_upperstart | #response_removespacebeforecomma |
#response_alterunderscores | #response_removetilde
#response_none is all flags turned off. See ^print for explanation of flags
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)

$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_looplimit – loop() defaults to 1000 iterations before stopping. You can change this default with this.

$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 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.

$cs_prepass – name of a topic to run in responder mode on main volleys, which runs before $cs_control_main and after all of the above and pos-parsing is done. Used to amend preparation data coming from the engine. You can use it to add your own spin on input processing before going to your main control. I use it to, for example, label commands as questions, standardize sentence construction (like "if you see me what will you think" to "assume you see me. What will you think?").

$cs_control_main – name of topic to run in responder mode on main volleys, set by author.

$cs_control_post – name of topic to run in gambit mode on post-pass, set by author

$botprompt – message for console window to label bot output

$userprompt – message for console window to label user input line

$cs_crashmsg – message to use if a server crash occurs.

$cs_language – if spanish, will adjust spell checking for spanish colloquial.

$cs_token- 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 – topic used by :abstract to display facts if you want them displayed

$cs_prepass – topic used between parsing and running user control script. Useful to supplement parsing, setting the question value, and revising input idioms.

$cs_wildcardseparator – when a match variable covers multiple words, what should separate them- by default it's a space, but underscore is handy too. Initial system character is space, creating fidelity with what was typed. Useful if _ can be recognized in input (web addresses). Changing to _ is consistent with multi-word representation and keyword recognition for concepts. CS automatically converts _ to space on output, so internal use of _ is normal.

$cs_userfactlimit – how many of the most recent permanent facts created by the script in response to user inputs are kept for each user. Std default is 100.

$cs_response – controls some characteristics of how responses are formatted

$cs_randIndex – the random seed for this volley

$cs_utcoffset – if defined, then %time returns current utc time + timezone offset. The offset is usually a simple number, meaning hours, and can have + or – in front of it. It can also be a normal time reference like 02:30 which means plus 2 hours and 30 minutes beyond utc, or -01:30:20 which means 1 hour, 30 minutes, and 20 seconds before utc (as if anyone would use that).

The following variables are generated by the system on behalf of scripts.
$$db_error – error message from a postgres failure
$$findtext_start - ^findtext return the end normally, this is where it puts the start.
$$tcpopen_error – error message from a tcpopen error
$$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