ChatScript Dictionary Ontology Pos-Parser Manual
© Bruce Wilcox, gowilcox@gmail.com
Revision 8/17/13 cs3.53

Proper name merging

ChatScript faces uncommon problems for most Natural Language tools. The best parsers are trained on the Wall Street Journal, where everything is properly cased. If you pass "i like you" to the Stanford parser, it will tell you that "i" is a foreign word, "like" is a preposition, and "you" is a pronoun. Chatters often never use upper case and speech recognitions devices don't output it either. So ChatScript works to handle all cases of things. This gets even messier given that ChatScript will attempt named entity extraction – it will decide some sequences of words represent a single upper-case compound name.

By default ChatScript attempts to detect named entities given as multiple words, and merge them into a single token with underscores. "united states of america" becomes "United_States_of_America". This is not without its hazards. The WORLDDATA ships with a large number of names of works like movies, tv shows, books, etc. These can collide. "Don't lie to me" can see "lie to me" as a possible TV show name. The merging code will ALWAYS merge quoted strings and capitalized sequences, so "Lie to Me" will get merged. It will also merge any sequence whose result is marked with the property ALWAYS_PROPER_NAME_MERGE. Currently world data adds this on to all location names (countries, cities, states) so they are always a single token no matter what case is used. The merging code will not automatically merge the names of TV shows. Nevertheless the sequence will be marked with its appropriate concepts because the system checks sequences of words even if they are not merged. So you would get "lie" "to" "me" "Lie to Me" all marked appropriately and it is up to the script to distinguish.

I run script early on that sees if ~NOUN_TITLE_OF_WORK matches. If it does, it determines if it was used in a sentence involving relevant verbs like "I watched lie to me yesterday" or if the user entered it capitalized or quoted or if we were already in the TV topic. If none of those are true, the script erases the ~TV topic mark using ^unmark. This prevents the system from erroneously trying the TV topic. You can also just turn off proper name merging by changing the value of $TOKEN removing the | #DO_PROPERNAME_MERGE value usually applied, at the cost of having to deal with named sequences yourself in script.

ChatScript Parser

The ChatScript parser runs by default on all input, doing what it can to parse it. If it believes it failed to parse correctly then it will set the tokenflags appropriately. You can test for this like this:
u: (%tokenflags&#FAULTY_PARSE)

You can suppress the parser by setting $token to NOT have the request to parse | #DO_PARSE

If the parser is not run, then the pos-tags of all words will be the maximum unrefined set of values. E.g., *dog* will be marked as a noun and a verb. If the parser was run, it will restrict the set of values of all words. If it failed to parse correctly, these may be in error.

The fragments of a successful parse are retrievable using concepts. The main sentence, because it may not contiguous and is special, does not have a composite retrieval. Instead you can retrieve

~mainsubject, ~mainverb, ~mainindirect, ~maindirect. Things which are contiguous chunks are phrases, clauses, and verbals. If

ChatScript has a complexity limit, and cannot handle sentences near 255 words. Also, sentences containing more than 7 of some particular concept will not mark after the first 7. So a sentence with 8 nouns will not have a ~noun after the 7th. If the nouns were a mixture of singular and plural, then it will represent them, up to the 7 limit.

Individual words serve roles, which are retrievable. These include
~mainsubject, ~mainverb, ~mainindirect, ~maindirect
~subject2, ~verb2, ~indirectobject2, ~object2
~conjunct_noun, ~conjunct_verb, ~conjunct_adjective, ~conjunct_adverb, ~conjunct_particle~bit~
~conjunct_clause, ~conjunct_sentence
~appositive – noun restating and modifying prior noun
~adjectival_noun (noun used as adjective like bank "bank teller"
~reflexive - (reflexive pronouns)
~not
~address – noun used as addressee of sentence
~subject_complement – (adjective object of sentence)
~object_complement – (2ndary noun or adjective filling in mainobject or object2)

Because ChatScript cannot store match position information on sequences of words longer than 5, only the leading word of a clause, phrase, or verbal is marked, using it as a role. You can get the entire sequence using ^getparse() passing it a match variable. That function can also return just interesting roles of the sequence. A subordinate clause is ~clause, a verbal is ~verbal, and a prepositional phrase depends on its type.

~phrase – ordinary phrase headed by a prepositional
~absolutephrase – leading or tailing phrase with preposition omitted
~timephrase – time phrase with preposition omitted

Individual words can be retrieved by various pos types, both generic and specific. Only one specific of a generic can be listed on a word at one time.

Generic	**Specific**
~noun	~noun_singular, ~noun_plural, ~noun_proper_singular, ~noun_proper_plural, ~noun_gerund, ~noun_cardinal, ~noun_ordinal, ~noun_infinitive
~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_be, ~aux_have, ~aux_do The tense of the be/have/do verbs can be had via ^properties() and testing for verb tenses
~adjective	~adjective_normal, ~adjective_ordinal, ~adjective_cardinal, ~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, ~pronoun_possessive

~determiner
~predeterminer
~preposition
~to_infinitive (“to” when used before an infinitive)
~conjunction_coordinate
~conjunction_subordinate
~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)
~possessive (covers ' and 's at end of word)
~there_existential (the word there used existentially)

Some of those are obvious and some aren’t.
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_cardinal (a normal number value like 17 or seventeen) and ~adjective_cardinal (also a normal numeral value) and ~adjective_ordinal (which is also ~placenumber) like *first.*

To can be many things. When used in the infinitive phrase *To go*, it is marked ~to_infinitive. ~verb_infinitive gets special handling. It refers both to a match on the infinitive form of the verb, as well as the phrase with to attached—i.e., *I like to swim* will match (I like to ~infinitive) as will (I like ~infinitive). This allows you to use a pattern like (I like [~infinitive ~participle]) treating *I like running* and *I like to run* as being the same meaning. This fails if the to is separated by an adverb as in *To boldly go*.

~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)*

This part of the manual is not for users of ChatScript; it is for developers of ChatScript, particularly in other languages than English.

Dictionary Rationale

It is entirely feasible to write a chatbot without a dictionary, without pos tagging, and without parsing. The use of ChatScript's concepts can enough ontology. So why is a dictionary useful?

First, it supports pos-tagging and parsing by specifying what parts of speech a word might be. Second it supports the internal spelling correction system, which will only try to fix words it does not recognize as real words, where words in the dictionary with a proper part of speech are considered real. Third, it supports an initial ontology for nouns, such that a collie is a dog is an animal is an entity, etc.

ChatScript ships with an English dictionary derived primarily from WordNet, which has some 150,000 words and which creates a hierarchy (ontology) of words. The noun hierarchy of WordNet is somewhat useful, but the ones for verbs, adjectives, and adverbs are useless from a chat point of view. An example of a lack in WordNet on nouns is the word “cat”, which is a feline, which is a carnivore, a mammal, etc. But nothing connects cat as a submember of pet. So ChatScript concepts supplement WordNet to expand the ontology available.

WordNet's verb hierarchy is minimal and useless. Take three verbs like: clean, decorate, and paint. What do they have in common? Start with clean. The first verb meaning is to make clean by removing dirt, a submember of “change, alter, modify”. The second meaning is remove unwanted substances from and is a submember of “remove, take away, withdraw”. Decorate meaning to adorn or ornament is also from “change, alter, modify”. Paint, meaning to coat with paint, is from “coat, surface”, which is from “cover”. So in WordNet there is no obvious connection on these three words. Yet to me, they are all ways of improving appearance. So ChatScript has its own verb ontology.

Sumo (Suggested Upper Merged Ontology) is not much better and doesn't even pretend to handle adjective and adverb ontologies.

PosTagging Rationale

While you can survive in ChatScript without knowing the part of speech of something in a sentence, you can do more precise matches if you do.

```
u: ( hit * me)  Don't hit me.
```

Without pos-tagging, this reacts to “why do you hit me” and “you are a hit with me”. With pos-tagging you can limit hit to being used as a verb via:

```
u: (hit~v * me) Don't hit me.
```

Similarly a concept like ~strike (hit~v strike~v) can be triggered only by verb uses of the words and not react to “The party was a hit”.

Different systems use different collections of pos-tag labels, some with few and some with many. The classic is the Penn Treebank collection of 36 tags. Fewer tags means it is easier to statically pos-tag something but provides less information about what role the word is actually serving. In TreeBank, for example, they don't distinguish between a preposition and a conjunction. Another system has over 100 tags, separately noting tense on the verbs to “be” and to “have” from other verb tenses. Generally speaking you can easily transcribe from something with more tags to something with fewer tags but not visa versa. Because my parser and pos-tagger interact, I have my own tags, which indicate something

about how the word is used. They are listed in a table at the end of this doc.The pos-tagger ignores the general label (NOUN) and uses specific format labels like:
NOUN_SINGULAR, NOUN_PLURAL, NOUN_PROPER_SINGULAR,
NOUN_PROPER_PLURAL, NOUN_CARDINAL, NOUN_ORDINAL,
and NOUN_GERUND (a verb in ing form being used as a noun)

For the word “swiming” this means it gets NOUN_SINGULAR from WordNet and gets NOUN_GERUND inferred from my code. Some rule later will have to decide which form of noun it is (NOUN_GERUND will overrule NOUN_SINGULAR).

Likewise a verb used as an ADJECTIVE will be an ADJECTIVE_PARTICIPLE and a noun used like an adjective (“bank teller”) will be an ADJECTIVE_NOUN. Labeling “bank” as an adjective means that when you pattern match in ChatScript searching for a noun, you won't stop at “bank” but proceed to a true noun (“teller”)

One can refer to single bits like NOUN_SINGULAR in a rule, as well as collections of bits like NOUN_BITS or NORMAL_NOUN_BITS. These are all listed in dictionarySystm.h

Parsing Rationale

While you can write patterns of words, sometimes writing patterns based on the sentence parse is better.

```
s: ( << ~mainsubject?~pet ~mainverb?~ingest >>)
```

Consider the above pattern which reacts to any statement about animals ingested something. A pattern like:

```
s: (<< ~pet ~ingest >>)
```

could react inappropriately to input like “I eat chicken”. A pattern like:

```
s: ( ~pet * ~ingest)
```

is safer in this context, but would still go astray with an unlikely input like “My cat watched TV while I ate dinner.”

Statistical parsers like Stanford's will parse anything but have no real sense of grammar and can get basic sentences wrong. “He hates eating broccoli” is correctly considered a verbal for the main object - “he hates eating” is the core sentence, with broccoli as an object of eating. But “He hates decaying broccoli” uses the verb as an adjective-participle, not a gerund, so the core sentence is “he hates broccoli”. They can't handle that distinction. And because they are based on statistics (typically from Wall Street Journal text) they consider “like” to be a preposition. This means to them “I do like you” consists of the verb “do” and the prepositional phrase “like you”. Even their accuracy rate isn't great. Stanford's, a gold standard among pos-taggers, gets around 97.2% right tagging things in text it has trained on. That translates to about 56% of sentences actually parsed correctly (WSJ sentences exceed 20 words, 5 such sentences will generate around 3 wrong pos-tags, which destroys the parse of at least 2 sentences).

Code Architecture

So, developing a dictionary for a language is good. After that, some NL systems completely pos-tag an input before then attempting to parse it. There are two types of pos taggers, statistical and rule-based. Experts tend to think of pos-tagging as a “solved” NL problem. They use statistical information from some corpus of pos-tagged sentences to train their pos tagger. The gold standard of this is the Stanford

parser. It is considered to be 97.3% accurate pos tagging in the trained domain of the Wall Street Journal. Sounds good, doesn't it?

Not to me. What this means is that if your document has sentences of 20 words (WSJ tends to be longer) then after 5 sentences (100 words) you will have 3 words wrong. If they occur in separate sentences, then 3 of your 5 sentences will be parsed incorrectly. That is, parsing accuracy is worse than pos-tagging accuracy. Worse, if you change domains (like American novels), the accuracy of the pos-tagging declines. I find it unacceptable that for this input:

"The strong outweigh the weak"

Stanfords pos tagger declares *strong* and *weak* to be adjectives and *outweigh* to be a noun. Or for this:

"I can fruit"

it declares that *can* is a modal helper verb and *fruit* is the main verb.
All you have to do to fool it is use words in their less likely meanings based on their statistics. The word *like,* for example, in the WSJ is primarily a preposition. But in chat it is primarily a verb.

The alternative approach to pos-tagging is rule-based, using the grammar of the language. The best reported pos-tagger to date claims a 99% accuracy rate on WSJ. Great! The complaint against rule-based systems is they take specialized knowledge to create and require lots of hand-crafted rules, so they are labor intensive. I say, so what? I need better accuracy.

ChatScript is agnostic. You can fill in whatever code you want for these tasks. For English I went with a rule-based grammar approach.

The first level of processing is a rule-based pos-tagging using local context to remove as much ambiguity as possible, without making any mistakes. That is, it leaves a bunch of words with multiple POS values if needed.

It then turns control over to a parser which is aware of the grammatical structures of sentences and attempts to resolve ambiguities so as to find a legal probable parse. The system will stop with the first valid parse it finds. Otherwise it may have to back up and try alternative parses. That's another advantage of my grammar based approach. It can tell when it hasn't parsed things correctly. The Stanford system has no clue.

Whenever the parser comes into an ambiguous word, it looks at the most probable pos-tag of the word first, to see if enough local structure could support that interpretation. If so, it goes with it. Otherwise if finds an interpretation that could be supported. Later, if it finds a place in the sentence where things are not working out, it may have to alter some prior decision it made, to rectify the structure. So it is primary a "garden path algorithm". Do the obvious as you come to a word, then fix thing backwards if a later word proves to be a problem.

There are three levels of information available to the system. First, each word (and canonical form of it), has its dictionary properties and flags. This is used by both the rule analyzer and the parser. Second, when the system is parsing, it builds current structures, so it knows what it is looking for next. Third, properties of words can be passed in from outside concept sets into the parser. That's how it knows a verb is factitive or not, for example.

Pos Tagging

The rules for the pos tagger are in LIVEDATA/ENGLISH. You can have any number of files with any

names, and the system reads them all in at startup to build its rules table.

When an input sentence is being processed, the system will take each word in turn and run relevant rules with that word as the starting point for staring at. It may or may not be the word we intend to alter the pos-tagging of.

In addition to their specific pos possibilities, words in the dictionary also have useful flag information, some of which is on their property bits and some is on their systemflags. Things like does a verb take a direct object, is this a mass noun, is this a known human name, is this the comparative form of an adjective, and so on. These bits are accessible in rule tests. Further flags for the parser (not for the pos-tagging rules) are accessible from the concept system. The concept ~special_english_attributes contains concepts which list special features of some verbs which do not have to be built into the dictionary. ~factative_noun_verbs, for example, is a list of verbs that can have a noun object complement to the direct object. "We elected him president". Similarly is a list for verbs that can have an adjective object complement like "We proclaimed him worthy". So we can either use bits on words in the dictionary to provide data to the postagger and parser or can declare concepts and write code for the parser to access those bits.

A typical rule looks like this:

```
# verb cannot be followed by possessive - "my *mother's coat was blue"
0 INCLUDE * VERB_TENSES
        IS POSSESSIVE
DISCARD
```

A rule consists of a comment, some number of lines dedicated to testing nearby words. One of those lines will have INCLUDE * … which means that the current word is ambigiuous and has as some of its pos-tagging values the named bits (in this case all possible verb tenses). You have up to 8 tests, after which is DISCARD or KEEP. If the rule matches all the tests, then the result is to either discard the bits of the include or to keep only those bits and discard all others.

The rule starts with a comment (#) that explains the rule's intent. This comment will in appear in traces when watching what rules fired. The quoted sentence in the comment is an example wherein the word with the * prefix is what the current focus word would be while executing this rule. So in this example the system will have looked at rules applied to "my" and is now on rules applied to "mother" (which is potentially a noun and a verb). This rule says that in this context (followed by a possessive) the word cannot be a verb. Sometimes there are also sentences NOT meant to match, given as "but not …."

The next line is the first test line and has a "0". This indicates the rules the operation will start with no offset from the current word it is staring at. One can start the examination at a negative offset, meaning an earlier word in the sentence, or a positive offset, meaning a later word. Normally the system will use apply each line of the rule to successive words, marching forward from the starting point, but you can instead make it march backward by using REVERSE, e.g.,

```
# a verb will never be preceded by a possessive pronoun - "my *mother is great"
REVERSE 0 INCLUDE * VERB_TENSES
        IS POSSESSIVE_BITS
DISCARD
```

The test that INCLUDE makes is merely to see that the word has one or more of the bits listed. No rule will waste time on any other tests if the INCLUDE fails.

```
# a verb will never be preceded by a possessive pronoun - "my *mother is great"
```

TRACE REVERSE 0 INCLUDE * VERB_TENSES
IS POSSESSIVE_BITS
DISCARD

In addition to REVERSE, you can also say TRACE to watch a rule as it attempts to process its rules.

When a rule wants to examine bits, you can name them as an aggregate name like
IS NOUN_BITS
or individually, like
IS NOUN_SINGULAR NOUN_PLURAL NOUN_PROPER_SINGULAR
You can also name them subtractively like
IS NOUN_BITS – NOUN_PROPER_PLURAL
which means all noun bits except noun_proper_plural.

Here is a list all of the op-codes available and what they mean. To understand properly what this pos-tagging phase does, you need to understand the setup. These apply to English. Most could probably be used for other languages, but you would need to write parser code yourself for a different language regardless, so you can redefine op-codes as you wish.

HAS – The current word being looked at has one or more of the bits named. These bits must be POS bits from the properties field, not supplement property bits nor systemflags bits. E.g.

present/past tense cannot follow potential conjunction subordinate (needs noun subject)
-1 HAS CONJUNCTION_SUBORDINATE
INCLUDE * VERB_PRESENT VERB_PRESENT_3PS VERB_PAST
DISCARD

CANONLYBE – similar to HAS except the word may not have any bits other than those given.

ORIGINALVALUE – similar to HAS except looks at the original bits before any reduction happened. So even if pos tagging rules have currently reduced this word to a verb, you can still tell it had the possibility of being a noun originally.

IS – Similar to has, except the word only has one bit meaning (has been determined) and that bit is one of the ones listed.

if adjective/determiner is followed by potential adjective/noun and then potential noun/verb and then prep or comma or conjunction, then assume word before comma etc cannot be a verb.
-2 IS DETERMINER ADJECTIVE_BITS - ADJECTIVE_PARTICIPLE
HAS NOUN_PROPER_SINGULAR NOUN_SINGULAR # an Afrikaner *farm worth anything
INCLUDE * VERB_PRESENT
IS PREPOSITION COMMA PAREN CONJUNCTION_BITS
DISCARD

INCLUDE – Similar to has except that in addition to having the bits named, the word also has other bits not named (hence is a candidate for having some bits kept or deleted). Each rule must have at least one INCLUDE and only one of them can have an * after it, which means this is the focus of KEEP or DELETE.

ISORIGINAL – the argument to this is a literal word, and the word under scrutiny is tested to see if it is this, case insensitive.

ISCANONICAL – similar to ISORIGINAL except it compares the canonical form of the word in the sentence against this word. So if the sentence has *helping*, ISCANONICAL help would match.

START – word is the first word of the sentence

END - word is the last word of the sentence

ISMEMBER – argument is a concept name and the word is a direct member of the named concept

HASPROPERTY – bits named come from the systemflags field and it has one or more of what was named.

HASCANONICALPROPERTY – similar to HASPROPERTY but tested on the canonical form of the word in focus.

HASALLPROPERTIES – similar to HASPROPERTY but it must have all the bits named.

ENDSWITH – given a word ending, sees if the original focus word has it. Used primary to see if a plural noun ends with an s or is an unusual plural noun.

POSSIBLEMASSNOUN – sees if the word might be a mass noun. The property bit associated with that is not available using other bit test opcodes so an opcode has to be dedicated to this.

ISQWORD – sees if the word is flagged as a classic question starter word. The property bit associated with it is not available via other opcodes.

ISABSTRACT – see if the word is flagged as a possible abstract noun.

NONEAFTER – test that these bits cannot be found in any word later in the sentence

NONEBEFORE – test that these bits cannot be found in any word earlier in the sentence.

PRIORCANONICAL – can this word be found anywhere earlier in the sentence looking at the canonical forms of the words.

ISQUESTION – asks if the sentence apparently starts like a question. It takes arguments of AUX and/or QWORD, indicating whether the sentence starts with a question word or an auxilliary verb. QWORD also accepts a question word in 2nd position if the 1st word of the sentence is a preposition (eg., For whom does the bell toll).

HAS2VERBS – asks if there are possibly two verbs in the sentence. If not, it cannot be a subordinate conjunction.

RESETLOCATION - reset the current focus location back to the start, but executing on the rules hereafter. You can supply it with the argument REVERSE, in which case it begins running in the opposite direction. This feature allows you to do multiple tests from the same location or check both

directions from the focus.

Most op-codes perform either a test on a word or a test overall, but there are a few special op-codes that combine with other opcodes.

! - not, inverts the test so it succeeds if the test fails. E.g.,

```
0  !HAS VERB_TENSES
   INCLUDE * NOUN_BITS
KEEP
```

The next special op code is:

STAY – do not advance (or reduce) the current word location. This allows you to perform an additional test on the current word. E.g.,

```
        0       ISCANONICAL who
                STAY FIRST
```

is a way to see if the word is “who” and is first in the sentence

The last special opcode is:

SKIP – keep advancing (or declining) the current word location as long as the test succeeds (or if the test fails and you are also using !)

```
        # present tense cannot exist with potential be in front of it (will be participle or adjective)
        REVERSE 0   INCLUDE * VERB_PRESENT VERB_PRESENT_3PS
        SKIP IS  ADJECTIVE_BITS ADVERB_BITS NOUN_BITS PRONOUN_BITS
        ISCANONICAL be
DISCARD
```

This rule considers the current word as a present tense verb, but if you ignore various words in front of it that are known to be resolved as adjective or adverb or noun or pronoun, then if before any of those is the word “be” in any of its tenses, don't accept this as a present tense verb.

Finally, a special modifer on the result of KEEP or DISCARD is a guess level. When the system runs out of safe things that clearly can be done, you have the option of executing heuristic rules that might be wrong (guesses). In the English version there are no guess rules because it is followed by a parser that does all the guessing. But if the parser is turned off, you can do guesses using pos-tag rules. When the system runs out of normal rules, it ups the guess level to the next one and tries again. There are 4 guess levels.

```
        0       INCLUDE * VERB_TENSES
        …..
        GUESS KEEP
or      GUESS1 KEEP
or      GUESS2 KEEP
or      GUESS3 KEEP
```

When you want to test the pos-tagger you can type

:pos this is a sentence

and see the trace of rules that fired and reduced the bits, the summary results of what is left, and if parsing is enabled the results of parsing.

POS TAGS

Here are the legal pos-tags that the system manages:

COMMA - ","
PAREN – "(", ")" used for open and close paren
PARTICLE – preposition used with a phrasal verb - "I will think it *over"
FOREIGN_WORD – "merci" - any foreign word we don't understand
TO_INFINITIVE - "to" when used for infinitive and not as a preposition (or particle)
PREPOSITION - "from"
THERE_EXISTENTIAL – there when used existentially as in "*there is something here"
DETERMINER - "a"
PREDETERMINER - "*half the size"
POSSESSIVE - the ′ at the end of a word or the *′s* at the end of a word when indicating possession

VERB_INFINITIVE - "go", "be"
VERB_PRESENT - "go", "am", "are"
VERB_PRESENT_3PS - "goes", "is"
VERB_PAST - "went", "was"
VERB_PAST_PARTICIPLE - "gone", "been"
VERB_PRESENT_PARTICIPLE - "going", "being"
collectively referred to as VERB_TENSES

NOUN_SINGULAR - "ox"
NOUN_PLURAL - "oxen", "birds"
NOUN_PROPER_SINGULAR - "John"
NOUN_PROPER_PLURAL - "Grammies"
NOUN_GERUND – "swmming"
NOUN_CARDINAL - "2", "3.14"
NOUN_ORDINAL - "I am *first", "fifty-fifth"
NOUN_INFINITIVE - "to swim" and includes bare infinitives: "We watched him *clear the table"
collectively referred to as NOUN_BITS
the collection NORMAL_NOUN_BITS is just the singular and plural of simple and proper nouns.

ADJECTIVE_NORMAL - "blue", "happier", "fastest"
ADJECTIVE_CARDINAL - "2"
ADJECTIVE_ORDINAL - "I see the *first tree"
ADEJCTIVE_PARTICIPLE - "the *walking dead", "the *hooked fish", "mathematics is *interesting"
collectively referred to as ADJECTIVE_BITS

ADVERB_NORMAL - "rapidly"
collectively referred to as ADVERB_BITS

PRONOUN_SUBJECT - "I", "one"
PRONOUN_OBJECT - "me", "whomever", "her"

PRONOUN_POSSESSIVE - “mine”, “her”
collectively referred to as PRONOUN_BITS

CONJUNCTION_COORDINATE - “and”
CONJUNCTION_SUBORDINATE - “while”
collectively referred to as CONJUNCTION_BITS

AUX_VERB_PRESENT - “can”
AUX_VERB_FUTURE - “will”
AUX_VERB_PAST - “dared”
collectively referred to as AUX_VERB_TENSES but special forms do not use those tense names (you can find their tense from property bits)
AUX_VERB_BE - all forms of be
AUX_VERB_HAVE – all forms of have
AUX_VERB_DO – all forms of do
the entirety referred to as AUX_VERB_BITS

That is the entirety of pos-tag bits used for English. Some bits have been reserved for German, but there is no German pos-tagger at present, so those names don't matter.

There are supplemental bits that can be tested using HASPROPERTY/HASCANONICALPROPERTY opcode:

PREP_TIME – a preposition used in time phrases
PREP_LOCATION - a preposition used in location references
LINKING_VERB - verb is a copular verb (takes adjective as object) - “seem”
SEPARABLE_PHRASAL_VERB – verb can take a separated particle - “Something to *think about”

BASIC_FORM – adjective or adverb is simple, not a more or most form
MORE_FORM – adjective or adverb is in the more or er state
MOST_FORM – adjective or adverb is in the most or est state
VERB_INDIRECTOBJECT - verb can take an indirect object - “he *hit her the ball”
VERB_DIRECTOBJECT – verb can take a direct object - “he *hit the ball”
POTENTIAL_CLAUSE_STARTER – aside from subordinate conjunctions, this might start a clause
ANIMATE_BEING – this noun is has possible volition – can be a reasonable indirect object
TIMEWORD – this word is associated with time - “I will fly home *tomorrow”
ACTUAL_TIME- this word is time itself “2:45”
WEB_URL -this word is a URL “www.amazon.com”
COMMON_PARTICIPLE_VERB – this verb will be an adjective after be or seem

OMITTABLE_TIME_PREPOSITION – word can be used without a preposition but still be a time phrase

GRADE5_6 – word is learn in grades 5-6
GRADE3_4 word is learned in grades 3-4
GRADE1_2 - word is learn in grades 1-2
KINDERGARTEN - word is learned in kindergarten
collectively known as AGE_LEARNED, all other words will be learned as adults

NOUN – statistically this word is likely a noun
ADVERB – statistically this word is likely an adverb
ADJECTIVE – statistically this word is likely an adjectives
VERB – statistically this word is likely a verb
PREPOSITION – statistically this word is likely a preposition
multiple of the above bits may be on simultaneously.

CONJUNCT_SUBORD_NOUN – subordinate conjunction starter can act as subject of clausse
OTHER_PLURAL - word is plural pronoun
OTHER_SINGULAR – word is singular pronoun
ALLOW_INFINITIVE – word allows infinitive - “I watched her swim”
PRONOUN_REFLEXIVE – word is a reflexive pronoun
EXTENT_ADVERB – adverb measures extent of some property – in front of adverb or adj, accept as adverb automatically.
VERBS_ACCEPTING_OF_AFTER - of can follow this verb
EXISTENTIAL_BE – these verbs can take an existential there as the subject
PREDETERMINER_TARGET – predeterminer allowed immediately before this
ADJECTIVE_POSTPOSITIVE – adjective that can occur immediately after a noun