English Phrase Parser



Design & API

English Phrase Parser Design & API

# Overview

This document describes the design, implementation and API of a Natural Language English Parser. This parser is layered on the domain grammar layer of recursive descent parser described in the document *Recursive Descent Parser Design.*

The parser leverages the tools and protocols established by the domain independent recursive descent parser. It implements an English language parser mostly by simply providing the lexical and grammatical domain knowledge that constitutes the English language.

All entities, unless otherwise specified are in the Common Lisp :parser package.

# Initialization

The English parser module can be compiled and/or loaded through the use the module :english-parser. The following function initializes the parser and the lexicons

## (initialize-english-parser)

**Example:**

PARSER(3): (initialize-english-parser)

Initializing English parser lexicons...

#P"C:\\Projects\\Data\\Lexicons\\Parts-of-Speech.lisp"

#P"C:\\Projects\\Data\\Lexicons\\Unknown-Words.lisp"

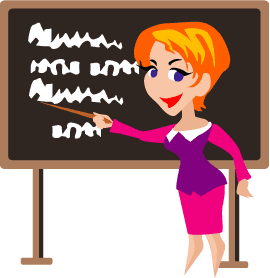
Loading table from file English-Irregular-Nouns ...

Loading table from file English-Irregular-Verbs ...

Loading table from file English-Irregular-Auxiliary ...

Initializing reverse lexicon table...

🡪T



# Lexical Knowledge

## Dictionary Knowledge

The English parser programmatically queries the Merriam-Website website to obtain lexical knowledge of English words. Query results are locally cached for performance purposes. The main purpose of using the Merriam-Website is to confirm word existence and extract part of speech information.

## Morphological Knowledge

For regular English words morphological knowledge is explicitly represented by suffix transformations for both pluralization and conjugation of nouns and verbs respectively. Irregular English words are explicitly enumerated in an irregular nouns and irregular verbs lexicon.

### (analyze-word-morphology <word>)

This function attempts to determine the part of speech and base word if applicable of a specified word. It returns five values: a base word, the word itself, the principal part of speech, additional lexical information and a list of all possible parts of speech. It first calls find-word-part-of-speech to retrieve any previously cached morphological analysis and otherwise proceeds to morphological analyze the word based on known english suffixes. This handles plurals, verb endings as well adjective and adverb formations.

**Examples**

PARSER (39): (analyze-word-morphology “traditionally")

1. "tradition"
2. "traditionally"
3. :ADVERB
4. NIL
5. (:ADVERB)

PARSER (40): (analyze-word-morphology “elephants")

1. "Elephant"
2. "Elephants"
3. :NOUN
4. :PLURAL
5. (:NOUN)

### (find-word-part-of-speech <word>)

This function attempts to determine the part of speech of the specified word by inspecting the local lexical caches as well as the Merriam Webster website. It returns the same five values as analyze-word-morphology.

(FIND-MERRIAM-WEBSTER-WORD <word>)

(FIND-REVERSE-LEXICAL-ENTRY <word>)

## Lexical Caches

There are two lexical caches that are used to amortize the cost of looking up words in online dictionaries or reprocessing previously unrecognized words. These are the parts of speech cache and the unknown words cache. The functions below constitute the API for these two caches.

### Parts of Speech Cache

(ADD-PART-OF-SPEECH-TO-CACHE <word>)

(GET- PART-OF-SPEECH -FROM-CACHE <word>)

(DELETE- PART-OF-SPEECH -FROM-CACHE <word>)

(CLEANUP- PARTS-OF-SPEECH -CACHE )

(CLEAR- PARTS-OF-SPEECH -CACHE )

(LOAD- PARTS-OF-SPEECH -FROM-FILE &optional <filename>)

(SAVE- PARTS-OF-SPEECH -TO-FILE &optional <filename>)

(BACKUP- PARTS-OF-SPEECH -TO-FILE &optional <filename>)

### Unknown Words Cache

(ADD-UNKNOWN-WORD-TO-CACHE <word>)

(GET-UNKNOWN-WORD-FROM-CACHE <word>)

(DELETE-UNKNOWN-WORD-FROM-CACHE <word>)

(CLEANUP-UNKNOWN-WORDS-CACHE )

(CLEAR-UNKNOWN-WORDS-CACHE )

(LOAD-UNKNOWN-WORDS-FROM-FILE &optional <filename>)

(SAVE-UNKNOWN-WORDS-TO-FILE &optional <filename>)

(BACKUP-UNKNOWN-WORDS-TO-FILE &optional <filename>)

Both caches are automatically backed up to disk after a sufficient number of new entries are added to the in-memory caches. Previous on disk backups are time stamped and versioned when this occurs.

# Grammatical Knowledge

Grammatical knowledge is currently represented as a context-free grammar which is parsed by a recursive descent parser. It is anticipated that feature based and adaptive grammars will be used eventually.

## \*english-grammar\*

**Example:**

PARSER(32): \*english-grammar\*

#<Plain-Grammar: English Grammar>

**English Grammar PRODUCTIONS**

* (:S (:NP :VP :NP :PP\*) (:NP :VP :NP) (:NP :VP :PP :NP) (:NP :VP :PP)
* (:NP :CONJUNCTION :NP) (:NP :PP\*) (:NP) (:PP\*))
* (:VP (:ADVERB :AUXILIARY :AUXILIARY :VERB) (:AUXILIARY :ADVERB :AUXILIARY :VERB)
* (:AUXILIARY :AUXILIARY :ADVERB :VERB) (:AUXILIARY :AUXILIARY :VERB)
* (:ADVERB :AUXILIARY :VERB) (:AUXILIARY :ADVERB :VERB) (:AUXILIARY :VERB)
* (:ADVERB :VERB) (:ADVERB :AUXILIARY) (:AUXILIARY) (:VERB))
* (:NP (:PN) (:NP1 :CONJUNCTION :NP1) (:NP1 :COMMA :NP) (:ARTICLE :NP1) (:NP1))
* (:NP1 (:ARTICLE :NP\*) (:NP\*)) (:NP\* (:ADJECTIVE :NP\*) (:NOUN :NP\*) (:NOUN))
* (:PN (:ARTICLE :NOUN :PROPER-NOUN) (:PROPER-NOUN :ARTICLE :NOUN) (:PRONOUN)
* (:PROPER-NOUN :INITIAL :PROPER-NOUN) (:PROPER-NOUN :PROPER-NOUN) (:PROPER-NOUN))
* (:PP\* (:PP :PP\*) (:PP)) (:PP (:PREPOSITION :NP))

**English Grammar TERMINAL Symbol Predicates**

* (:NOUN #<STANDARD-GENERIC-FUNCTION ENGLISH-NOUN-P>)
* (:VERB #<STANDARD-GENERIC-FUNCTION ENGLISH-VERB-P>)
* (:ADJECTIVE #<STANDARD-GENERIC-FUNCTION ENGLISH-ADJECTIVE-P>)
* (:PREPOSITION #<STANDARD-GENERIC-FUNCTION ENGLISH-PREPOSITION-P>)
* (:ARTICLE #<STANDARD-GENERIC-FUNCTION ENGLISH-ARTICLE-P>)
* (:PRONOUN #<STANDARD-GENERIC-FUNCTION ENGLISH-PRONOUN-P>)
* (:INITIAL #<STANDARD-GENERIC-FUNCTION ENGLISH-INITIAL-P>)
* (:PROPER-NOUN #<STANDARD-GENERIC-FUNCTION ENGLISH-PROPER-NOUN-P>)
* (:ADVERB #<STANDARD-GENERIC-FUNCTION ENGLISH-ADVERB-P>)
* (:AUXILIARY #<STANDARD-GENERIC-FUNCTION ENGLISH-AUXILIARY-P>)
* (:CONJUNCTION #<STANDARD-GENERIC-FUNCTION ENGLISH-CONJUNCTION-P>)
* (:COMMA #<STANDARD-GENERIC-FUNCTION ENGLISH-COMMA-P>)

**NON-TERMINAL Symbols:** (:S :VP :NP :NP1 :NP\* :PN :PP\* :PP)

CONTEXT NIL

LEXICONS #<Lexicons(0): NIL>

# Semantic Knowledge

## Wordnet

This section describes the fundamental concepts underlying the Wordnet project as well as the API functions from the parser module into the Wordnet module.

### Definitions

The following terms characterize useful semantic relationships between pairs of words and word phrases.

|  |  |  |
| --- | --- | --- |
| **Relation Name** | **Relation Definition** | **Example** |
| Synonym | Words with similar meaning | Big & Large |
| Antonym | Words with opposite meaning | Large & Small |
| Homonym | Words with similar sound or spelling but different meaning | Whether & Weather |
| Hypernym | Word that generalizes another word | Insect and bug |
| Hyponym | Word that specializes another word | Bug and Insect |
| Meronym | Word that names a component of another word | Finger and hand |
| Holonym | Word that names the whole of what another word is a component of. | Hand and Finger |
| Pertainym | Word (adjective) that classifies another word | Musical and Instrument |

### API

The following functions are in the Wordnet package.

**(INITIALIZE-WORDNET)**

This loads and initializes the Wordnet module.

(**FIND-WORDNET-WORD** <word>)

This function returns a list of Wordnet words. These are objects that inherit from the Wordnet class SYNSET. The argument <word> can be a string or a common lisp symbol.

(**WORDNET-HYPERNYMS** <wordnet-word>)

(**WORDNET-HYPONYMS** <wordnet-word>)

(**WORDNET-SYNONYMS** <wordnet-word>)

# Parser Module Classes

## Token Classes

Token objects are created as a result of invoking the scan-tokens generic function. Different types of token objects are created based on the character composition of the token. If we assert that there three fundamental types of characters (alphabetic, numeric and special) then we following we have seven possible character combinations. The following seven token classes capture this notion:

* Alphabetic-Token
* Numeric-Token
* Special-Token
* Alpha-Numeric-Token
* Alpha-Special-Token
* Numeric-Special-Token
* Alpha-Numeric-Special-Token

These classes inherit from the abstract *token* class as well as the appropriate mixin classes:

* alphabetic-token-mixin
* numerical-token-mixin
* special-token-mixin



## English Word Classes

Word objects are created as a result of invoking the *analyze-tokens* generic function. They represent a mapping from lexical token classes to syntactic english word classes which are essentially responsible for assigning syntactic roles, i.e. parts of speech, to the tokens. These syntactic role assignments are then leveraged by the parser in an attempt to find a successful path(s) through the grammar that accommodates the input tokens.

* Domain-Word
* English-Word
* Unknown-Word

### Standard Word Roles

* English-Noun
* English-Proper-Noun
* English-Article
* English-Verb
* English-Auxiliary
* English-Adjective
* English-Adverb
* English-Preposition
* English-Pronoun
* English-Conjunction
* English-Punctuation

### Advanced Word Roles

* English-Proper-Name
* English-Initial
* English-Number
* English-Year
* English-Date

(**object-token** <word>)

This returns a token object of the appropriate class denoting the sequence of characters that compose <word>.

(**word-base-form** <word>)

This returns a string denoting the base for of the word. For nouns this could be the singular form, for verbs the infinitive form or for adjectives and adverbs this could be the noun they were formed from. See *analyze-word-morphology*.

## Grammar Classes

### Standard Grammar Classes

* **Grammar**: Fundamental grammar class with productions, terminal & non-terminal symbols.
* **Context-Grammar**: Adds a context slot to the grammar class
* **Domain-Grammar**: Adds a lexicons slot to the context grammar class
* **English-Grammar**: Implements a particular domain-specific grammar

(**define-grammar** <name> <productions> <terminal-symbols> &key <class>)

This macro can be used to define a standard grammar that inherits from the class grammar. It returns a grammar object.

(**define-context-grammar** <name> <productions> <terminal-symbols> <context> &key <class>))

This macro can be used to define a context grammar that inherits from the class context-grammar. It returns a context grammar object.

### Advanced Grammar Classes

The following classes are used to implement an adaptive learning grammar that essentially recognizes frequent patterns that consist of an unrecognized entity surrounded by recognizable grammatical patterns and automatically augments the grammar to recognize such patterns in the future. The grammar essentially learns new productions.

* Adaptive-Grammar-Mixin
* Adaptive-English-Grammar

(**enable-adaptive-grammar** &optional <grammar>)

This function turns on the adaptive grammar behavior and enables grammatical pattern logging. The optional <grammar> argument

(**disable-adaptive-grammar** &optional <grammar>)

This function turns off the adaptive grammar behavior and disables grammatical pattern logging.

## Parse-Node Classes

* **Parse-Node**: An abstract class that the following classes all inherit from.
* **Parse-Tree**: The root and representative node of a tree of parse nodes.
* **Internal-Parse-Node**: Internal nodes representing non-terminal symbols
* **Leaf-Parse-Node**: Leaf nodes representing terminal symbols
* **Unused-Parse-Node**: Nodes representing unparsed words.

(**print-parse-tree** <node> &optional <stream>)

(**parse-tree-to-sentence** <parse-tree>)

This function takes a *parse-tree* object and returns a newly constructed *sentence* object.

## Phrase Classes

* **English-Phrase**
* **Noun-Phrase**
* **Verb Phrase**
* **Prepositional Phrase**

(phrase-tokens <phrase>)

(pronoun-phrase-p <phrase>)

## Sentence Classes

* **English-Sentence**
* **Parsed-Sentence**

(**object-phrases** <sentence>)

(**object-parse-trees** <sentence>)

(**find-subject-phrase** <sentence>)

This function tries to identify the subject of <sentence> and returns a new subject-phrase object or nil if no sentence can be identified.

(**find-verb-phrase** <sentence>)

This function tries to identify the verb of <sentence> and returns a new verb-phrase object or nil if no verb phrase can be identified.

(**find-direct-object-phrase** <sentence>)

This function tries to identify the direct-object of <sentence> and returns a new direct-object-phrase object or nil if no direct object can be identified.

(**find-prepositional-phrases** <sentence>)

This function tries to identify the prepositional phrases in <sentence> and returns new prepositional-phrase objects or nil if no prepositional phrases can be identified.

(sentence-number <sentence>)

# Parser Functions

## Basic Parser Functions

(**SCAN-TOKENS** <string> <grammar> &key <delimiter>)

**Example:**

PARSER(6): (scan-tokens "The cat chased the mouse at 3:00PM in 1987." nil)

(#<Alphabetic Token: The> #<Alphabetic Token: cat> #<Alphabetic Token: chased> #<Alphabetic Token: The>

#<Alphabetic Token: mouse> #<Alphabetic Token: at> #<Alpha-Numeric-Special Token: 3:00PM>

#<Alphabetic Token: in> #<Numeric-Special Token: 1987.>)

(**ANALYZE-TOKENS** <tokens-or-string> <grammar> &key <delimiter>)

**Example:**

PARSER(9): (analyze-tokens "The cat chased the mouse at 3:00PM in 1987." \*english-grammar\*)

(#<ARTICLE: The> #<NOUN: Cat> #<VERB: Chased> #<ARTICLE: The> #<NOUN: Mouse> #<PREPOSITION: at> #<UNKNOWN: 3:00PM> #<PREPOSITION: in> #<YEAR: 1987> #<PUNCTUATION: PERIOD>)

(**SIMPLE-PARSER** <tokens-or-string> <grammar> &key <delimiter>)

This returns four values: a list of parse trees, a list of lists of phrase words, a list of unconsumed tokens/words and a list of productions that are aligned with the second value returned. The last three values will in fact be deprecated as they can in fact be extracted and generated from the first value returned.

**Example:**

PARSER(22): (simple-parser "The cat chased a white mouse in the room." \*english-grammar\*)

1. (#<PARSE-TREE: The cat chased a white mouse in The room> #<UNPARSED: .>)
2. ((#<ARTICLE: The> #<NOUN: Cat> #<VERB: Chased> #<ARTICLE: a> #<ADJECTIVE: White> #<NOUN: Mouse> #<PREPOSITION: in> #<ARTICLE: The> #<NOUN: Room>))
3. (#<PUNCTUATION: PERIOD>)
4. ((:ARTICLE :NOUN :VERB :ARTICLE :ADJECTIVE :NOUN :PREPOSITION :ARTICLE :NOUN))

(**PARSE** <tokens-or-string>)

This function is provided for convenience and simply calls *simple-parser* with the \*english-grammar\* object and the space character as the delimiter.

(**PARSE-SENTENCE**  <tokens-or-string><grammar>)

This function returns an object of class parsed-sentence.

**Example:**

PARSER(35): (setf sentence (parse-sentence "The cat chased a white mouse in the room." \*english-grammar\*))

#<SENTENCE(10): The cat chased ...>

PARSER(36): (describe sentence)

#<SENTENCE(10): The cat chased ...> is an instance of #<STANDARD-CLASS PARSED-SENTENCE>:

The following slots have :INSTANCE allocation:

UTILITIES::VALUE NIL

UTILITIES::NAME "The cat chased ..."

WORDS

(#<ARTICLE: The> #<NOUN: Cat> #<VERB: Chased> #<ARTICLE: a> #<ADJECTIVE: White> #<NOUN: Mouse> #<PREPOSITION: in> #<ARTICLE: The> #<NOUN: Room> #<PUNCTUATION: PERIOD>)

NUMBER 0

PARSE-TREES

(#<PARSE-TREE: The cat chased a white mouse in The room> #<UNPARSED: .>)

PARSED-WORDS

((#<ARTICLE: The> #<NOUN: Cat> #<VERB: Chased> #<ARTICLE: a> #<ADJECTIVE: White> #<NOUN: Mouse> #<PREPOSITION: in> #<ARTICLE: The> #<NOUN: Room>) NIL)

PARSED-PRODUCTIONS

((:ARTICLE :NOUN :VERB :ARTICLE :ADJECTIVE :NOUN :PREPOSITION :ARTICLE :NOUN) NIL)

UNPARSED-WORDS (NIL (#<PUNCTUATION: PERIOD>))

PERCENTAGE-PARSED 90.0

(**SPLIT-WORDS-INTO-SENTENCES** <words>)

This functions partitions <words> into sub-lists based on sentence punctuation. The argument <words> should be a list of word or token objects as returned by *analyze-tokens*.

Note: This function needs to be improved. Currently only uses punctuation as an end of sentence indicator.

## Advanced Parser Functions

(**PARSE-STATEMENT**  <string> &key <class> <grammar> <delimiter>)

This function exists for convenience. It simply invokes parse-sentence with the appropriate arguments. The argument <grammar> defaults to \*declarative-english-grammar\* and <delimiter> defaults to #\Space. This function returns a parsed sentence object of class *declarative-sentence* or of the specified class which should inherit from this class.

**Example**:

KB(158): (parser::parse-statement "An elephant is a mammal from Africa")

#<STATEMENT(7): An elephant is a mammal...>

(**PARSE-QUESTION** <string> &key <class> <grammar> <delimiter>)

This function exists for convenience. It simply invokes parse-sentence with the appropriate arguments. The argument <grammar> defaults to \*interrogative-english-grammar\* and <delimiter> defaults to #\Space. This function returns a parsed sentence object of class *interrogative-sentence* or of the specified class which should inherit from this class.

**Example**:

KB(160): (parser::parse-question "What is an elephant?")

#<QUESTION(5): What is an elephant ?...>

(**FIND-SUBJECT-PHRASE** <sentence>)

This function returns the noun phrase object that is the subject of <sentence> or nil if none exists.

(**FIND-VERB-PHRASE** <sentence>)

This function returns the verb phrase object that is the verb phrase of <sentence> or nil if none exists.

(**FIND-DIRECT-OBJECT-PHRASE** <sentence>)

This function returns the noun phrase object that is the direct object of <sentence> or nil if none exists.

(**FIND-PREPOSITIONAL-PHRASES** <sentence>)

This function returns a list of the prepositional phrase objects that are the prepositional phrases of <sentence> or nil if none exist.

(**PARSE-TREE-PHRASES** <parse-tree>)

This function extracts and creates phrase objects from <parse-tree>. It returns a list of phrase objects of appropriate classes.

(**PARSE-TREE-TO-SENTENCE** <parse-tree>)

This function is to be deprecated. It takes a *parse-tree* object and returns a newly constructed *sentence* object. The *object-value* of the sentence object is a list of entries with each entry corresponding to a phrase in the sentence. Each entry is a list composed of a parse-node object denoting the root of the subtree for a particular phrase followed a list of the word objects that comprise that phrase.

## Adaptive Parser Functions

This section describes the adaptive parser functionality API. This allows the parser to learn new grammatical productions based on frequently observed but as of yet unrecognized grammatical productions.

### Manipulating Grammatical Patterns

#### Pattern Caches

* \*phrase-patterns-cache\*
* \*production-patterns-cache\*
* \*sentence-patterns-cache\*

#### Pattern Extraction

* (extract-phrase-patterns <sentence>)
* (extract-production-pattern <sentence>)
* (**delete-phrase-pattern-from-cache** extract-word-pattern <sentence>)

#### Phrase Patterns Cache

Of the three types of grammatical patterns described, phrase patters are the most actively used ones mostly because of their granularity and frequency of occurrence. Furthermore they correspond more naturally to patterns inside grammatical productions. The following functions manipulate the phrase patterns cache.

(**phrase-patterns-cache**)

(**add-phrase-pattern-to-cache** <phrase-pattern>)

**(<phrase-pattern>**)

(**clear-phrase-patterns-cache**)

(**phrase-patterns-list**)

(**phrase-patterns-count**)

(**save-phrase-patterns-to-file**)

(**load-phrase-patterns-from-file**)

#### Example Phrase Patterns

The following table shows some of the most frequent patterns observed whilst parsing the 228 related topic documents of the topic “elephant”. Notice how parenthetical patterns are split into patterns involving either left or right parentheses. Also notice how the leading phrase patterns mostly involve grammatical patterns that include some form of punctuation.

While these observed patterns could be used as guidelines to manually augment and refine the predefined English grammar, the goal here is to actually have the system automatically modify the English grammar based on the most frequently observed patterns. In the next section we describe an algorithm for doing this safely and effectively.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phrase 1** | **Unparsed Token** | **Phrase 2** | **Count** |
| (NP) | , | (NP) | 1324 |
| (NP) | ( | (NP) | 778 |
| (NP) | as | (NP) | 422 |
| (NP) | , | (PP\*) | 414 |
| (NP) | ) | NIL | 344 |
| (PP\*) | , | (NP) | 316 |
| (PP\*) | and | (NP) | 246 |
| (NP) | have | (NP) | 218 |
| (NP PP\*) | and | (NP) | 187 |
| (NP PP\*) | , | (NP) | 187 |
| (NP) | ; | (NP) | 180 |
| (PP\*) | , | (PP\*) | 174 |
| (PP\*) | ( | (NP) | 171 |
| (NP) | , | (NP CONJUNCTION NP) | 149 |
| (NP PP\*) | ( | (NP) | 147 |
| (NP) | , | (NP PP\*) | 139 |

### Dynamically Adapting a Grammar

For a phrase pattern of the form: (<phrase-1> <unparsed-token> <phrase-2>)

*if <phrase-1> consists of a single NTS*

1. *then*
2. *identify a production P, who’s LHS is <phrase-1>*
3. *add the entire phrase pattern as a new disjunction to the RHS of P.*
4. *else*
5. *identify a production whose RHS contains <phrase-1>*
6. *add the entire phrase patterns as a new disjunction that occurs before <phrase-1> in the RHS.*
7. *Verify that old test sentences still parse*
8. *Verify that sentences containing the phrase pattern now parse*
9. *Save the augmented grammar*

### Manipulating Grammar Productions

(production-lhs <production>)

(production-rhs <production>)

(find-production <nts> <grammar>)

(find-productions-with-nts <nts> <grammar>)

(production-contains-nts-p <production> <nts>)

(recursive-production-p <nts> <grammar>)

(disjunction-starts-disjunction-p <disjunc-1> <disjunc-2>)

(disjunction-finishes-disjunction-p <disjunc-1> <disjunc-2>)

### Modifying Grammar Productions

(add-terminal-symbol-to-grammar <terminal-symbol> <grammar> &key <position>)

(remove-terminal-symbol-from-grammar <terminal-symbol> <grammar>)

(add-production-disjunction-to-grammar <nts> <rhs-disjunct> <grammar>)



# Language Understanding

* Associate nouns with noun phrases
* Associate adjectives with noun phrases
* Associate noun phrases with verbs
* Associate verbs with nouns
* Etc…

# Interrogative Sentences

Interrogative Pronouns & Adverbs

PARSER(4): (analyze-word-morphology "what")

"What" "What" :PRONOUN NIL ((:PRONOUN) (:ADVERB) (:ADJECTIVE) (:CONJUNCTION) (:NOUN))

PARSER(5): (analyze-word-morphology "who")

"Who" "Who" :PRONOUN NIL ((:PRONOUN) (:NOUN))

PARSER(6): (analyze-word-morphology "where")

"where" "where" :ADVERB NIL ((:ADVERB) (:CONJUNCTION) (:NOUN))

PARSER(7): (analyze-word-morphology "when")

"When" "When" :ADVERB NIL ((:ADVERB) (:CONJUNCTION) (:PRONOUN) (:NOUN))

PARSER(8): (analyze-word-morphology "how")

"How" "How" :ADVERB NIL ((:ADVERB) (:CONJUNCTION) (:NOUN))

# Online References

<http://www.merriam-webster.com/>

<http://www.lexic.us/>

<http://dictionary.reference.com/>

<http://ask.reference.com/>

<http://www.mfnames.com/name-search-page.htm>



`(,@rpl)