WordNet Documentation

Note: This document[1] consolidates and organizes the information provided in the WordNet 3.0 package for Linux, file WordNet-3.0.tar.gz available at Princeton University - WordNet downloads.

General descriptions

The WordNet system consists of lexicographer files, code to convert these files into a database, and search routines and interfaces that display information from the database. The lexicographer files organize nouns, verbs, adjectives and adverbs into groups of synonyms, and describe relations between synonym groups. **grind** converts the lexicographer files into a database that encodes the relations between the synonym groups. The different interfaces to the WordNet database utilize a common library of search routines to display these relations. Note that the lexicographer files and **grind** program are not generally distributed.

Information in WordNet is organized around logical groupings called synsets. Each synset consists of a list of synonymous words or collocations (eg. **"fountain pen"** , **"take in"**), and pointers that describe the relations between this synset and other synsets. A word or collocation may appear in more than one synset, and in more than one part of speech. The words in a synset are grouped such that they are interchangeable in some context.

Two kinds of relations are represented by pointers: lexical and semantic. Lexical relations hold between semantically related word forms; semantic relations hold between word meanings. These relations include (but are not limited to) hypernymy/hyponymy (superordinate/subordinate), antonymy, entailment, and meronymy/holonymy.

Nouns and verbs are organized into hierarchies based on the hypernymy/hyponymy relation between synsets. Additional pointers are be used to indicate other relations.

Adjectives are arranged in clusters containing head synsets and satellite synsets. Each cluster is organized around antonymous pairs (and occasionally antonymous triplets). The antonymous pairs (or triplets) are indicated in the head synsets of a cluster. Most head synsets have one or more satellite synsets, each of which represents a concept that is similar in meaning to the concept represented by the head synset. One way to think of the adjective cluster organization is to visualize a wheel, with a head synset as the hub and satellite synsets as the spokes. Two or more wheels are logically connected via antonymy, which can be thought of as an axle between the wheels.

---

1 This document created from Princeton WordNet 3.0 documentation html files by G. Dalton Bentley April 24, 2020 and included in his github command line simplification repackaging of the WordNet software, distributed at WNtight command line version. We omitted discussion of Prolog-related files and of the Tcl/Tk browser version of WordNet. Got to Princeton Princeton University - WordNet downloads for the complete, original WordNet packages and documentation if you are interested in those omissions.

Pertainyms are relational adjectives and do not follow the structure just described. Pertainyms do not have antonyms; the synset for a pertainym most often contains only one word or collocation and a lexical pointer to the noun that the adjective is "pertaining to". Participial adjectives have lexical pointers to the verbs that they are derived from.

Adverbs are often derived from adjectives, and sometimes have antonyms; therefore the synset for an adverb usually contains a lexical pointer to the adjective from which it is derived.

Command line interface

wn [*searchstr*] [**-h**] [**-g**] [**-a**] [**-l**] [**-o**] [**-s**] [**-n#**] [*search_option ...*]

wn() provides a command line interface to the WordNet database, allowing synsets and relations to be displayed as formatted text. For each word, different searches are provided, based on syntactic category and pointer types. Although only base forms of words are usually stored in WordNet, users may search for inflected forms. A morphological process is applied to the search string to generate a form that is present in WordNet.

The command line interface is often useful when writing scripts to extract information from the WordNet database. Post-processing of the output with various scripting tools can reformat the results as desired.

Options[2]

-h
: Print help text before search results.

-g
: Display textual glosses associated with synsets.

-a
: Display lexicographer file information.

-o
: Display synset offset of each synset.

-s
: Display each word's sense numbers in synsets.

-l
: Display the WordNet copyright notice, version number, and license.

2 Note that these “Options” don’t seem to work in the command line interface reliably (no output). -over (with a search term) and -l (without other input) do function. Perhaps these anomalies have to do with our dropping the Tcl/Tk code in WNtight. G Dalton Bentley April 24, 2020. Consult our manual for WNtight for usage.

-n#
Perform search on sense number # only.
-over
Display overview of all senses of *searchstr* in all syntactic categories.

Search options[3]

Note that the last letter of *search_option* generally denotes the part of speech that the search applies to: **n** for nouns, **v** for verbs, **a** for adjectives, and **r** for adverbs. Multiple searches may be done for *searchstr* with a single command by specifying all the appropriate search options.

-syns (*n* | *v* | *a* | *r*)
Display synonyms and immediate hypernyms of synsets containing *searchstr* . Synsets are ordered by estimated frequency of use. For adjectives, if *searchstr* is in a head synset, the cluster's satellite synsets are displayed in place of hypernyms. If *searchstr* is in a satellite synset, its head synset is also displayed.
-simsv
Display verb synonyms and immediate hypernyms of synsets containing *searchstr* . Synsets are grouped by similarity of meaning.
-ants (*n* | *v* | *a* | *r*)
Display synsets containing antonyms of *searchstr* . For adjectives, if *searchstr* is in a head synset, *searchstr* has a direct antonym. The head synset for the direct antonym is displayed along with the direct antonym's satellite synsets. If *searchstr* is in a satellite synset, *searchstr* has an indirect antonym via the head synset, which is displayed.
-faml (*n* | *v* | *a* | *r*)
Display familiarity and polysemy information for *searchstr* .
-hype (*n* | *v*)
Recursively display hypernym (superordinate) tree for *searchstr* (*searchstr IS A KIND OF* _____ relation).
-hypo (*n* | *v*)
Display immediate hyponyms (subordinates) for *searchstr* (_____ *IS A KIND OF searchstr* relation).
-tree (*n* | *v*)
Display hyponym (subordinate) tree for *searchstr* . This is a recursive search that finds the hyponyms of each hyponym.
-coor (*n* | *v*)
Display the coordinates (sisters) of *searchstr* . This search prints the immediate hypernym for each synset that contains *searchstr* and the hypernym's immediate hyponyms.
-deri (*n* | *v*)

3 These search options work *if* WordNet tells you they are available. Always do an intial search on the word of interest without any option in order to obtain a list of these Search Options that are available for this word, e.g., wn word_of_interest.

Display derivational morphology links between noun and verb forms.

-domn (*n* | *v* | *a* | *r*)
Display domain that *searchstr* has been classified in.

-domt (*n* | *v* | *a* | *r*)
Display all terms classified as members of the *searchstr* 's domain.

-subsn
Display substance meronyms of *searchstr* (*HAS SUBSTANCE* relation).

-partn
Display part meronyms of *searchstr* (*HAS PART* relation).

-membn
Display member meronyms of *searchstr* (*HAS MEMBER* relation).

-meron
Display all meronyms of *searchstr* (*HAS PART, HAS MEMBER, HAS SUBSTANCE* relations).

-hmern
Display meronyms for *searchstr* tree. This is a recursive search that prints all the meronyms of *searchstr* and all of its hypernyms.

-sprtn
Display *part of* holonyms of *searchstr* (*PART OF* relation).

-smemn
Display *member of* holonyms of *searchstr* (*MEMBER OF* relation).

-ssubn
Display *substance of* holonyms of *searchstr* (*SUBSTANCE OF* relation).

-holon
Display all holonyms of *searchstr* (*PART OF, MEMBER OF, SUBSTANCE OF* relations).

-hholn
Display holonyms for *searchstr* tree. This is a recursive search that prints all the holonyms of *searchstr* and all of each holonym's holonyms.

-entav
Display entailment relations of *searchstr* .

-framv
Display applicable verb sentence frames for *searchstr* .

-causv
Display *cause to* relations of *searchstr* .

-pert (*a* | *r*)
Display pertainyms of *searchstr* .

-attr (*n* | *a*)
Display adjective values for noun attribute, or noun attributes of adjective values.

-grep (*n* | *v* | *a* | *r*)
List compound words containing *searchstr* as a substring.

WordNet Documentation

Search results

The results of a search are written to the standard output. For each search, the output consists a one line description of the search, followed by the search results.

All searches other than **-over** list all senses matching the search results in the following general format. Items enclosed in italicized square brackets (*[...]*) may not be present.

One line listing the number of senses matching the search request.

Each sense matching the search requested displayed as follows:

> **Sense *n***
> ***[{synset_offset}] [<lex_filename>] word1[#sense_number][, word2...]***
>
> Where *n* is the sense number of the search word, *synset_offset* is the byte offset of the synset in the **data.*pos*** file corresponding to the syntactic category, *lex_filename* is the name of the lexicographer file that the synset comes from, *word1* is the first word in the synset (note that this is not necessarily the search word) and *sense_number* is the WordNet sense number assigned to the preceding word. *synset_offset*, *lex_filename* , and *sense_number* are generated when the **-o, -a,** and **-s** options, respectively, are specified.

The synsets matching the search requested are printed below each sense's synset output described above. Each line of output is preceded by a marker (usually =>), then a synset, formatted as described above. If a search traverses more one level of the tree, then successive lines are indented by spaces corresponding to its level in the hierarchy. When the **-g** option is specified, synset glosses are displayed in parentheses at the end of each synset. Each synset is printed on one line.

Senses are generally ordered from most to least frequently used, with the most common sense numbered **1** . Frequency of use is determined by the number of times a sense is tagged in the various semantic concordance texts. Senses that are not semantically tagged follow the ordered senses. Note that this ordering is only an estimate based on usage in a small corpus.

Verb senses can be grouped by similarity of meaning, rather than ordered by frequency of use. The **-simsv** search prints all senses that are close in meaning together, with a line of dashes indicating the end of a group.

The **-over** search displays an overview of all the senses of the search word in all syntactic categories. The results of this search are similar to the **-syns** search, however no additional (ex. hypernym) synsets are displayed, and synset glosses are always printed. The senses are grouped by syntactic category, and each synset is annotated as described above with *synset_offset* , *lex_filename* , and *sense_number* as dictated by the **-o, -a,** and **-s** options. The overview search also indicates how many of the senses in

each syntactic category are represented in the tagged texts. This is a way for the user to determine whether a sense's sense number is based on semantic tagging data, or was arbitrarily assigned. For each sense that has appeared in such texts, the number of semantic tags to that sense are indicated in parentheses after the sense number.

If a search cannot be performed on some senses of *searchstr* , the search results are headed by a string of the form: X of Y senses of *searchstr*

The output of the **-deri** search shows word forms that are morphologically related to **searchstr** . Each word form pointed to from *searchstr* is displayed, preceded by **RELATED TO->** and the syntactic category of the link, followed, on the next line, by its synset. Printed after the word form is **#** *n* where *n* indicates the WordNet sense number of the term pointed to.

The **-domn** and **-domt** searches show the domain that a synset has been classified in and, conversely, all of the terms that have been assigned to a specific domain. A domain is either a **TOPIC, REGION** or **USAGE,** as reflected in the specific pointer character stored in the database, and displayed in the output. A **-domn** search on a term shows the domain, if any, that each synset containing *searchstr* has been classified in. The output display shows the domain type (**TOPIC, REGION** or **USAGE**), followed by the syntactic category of the domain synset and the terms in the synset. Each term is followed by **#** *n* where *n* indicates the WordNet sense number of the term. The converse search, **-domt** , shows all of the synsets that have been placed into the domain *searchstr* , with analogous markers.

When **-framv** is specified, sample illustrative sentences and generic sentence frames are displayed. If a sample sentence is found, the base form of *search* is substituted into the sentence, and it is printed below the synset, preceded with the **EX:** marker. When no sample sentences are found, the generic sentence frames are displayed. Sentence frames that are acceptable for all words in a synset are preceded by the marker ***>** . If a frame is acceptable for the search word only, it is preceded by the marker => .

Search results for adjectives are slightly different from those for other parts of speech. When an adjective is printed, its direct antonym, if it has one, is also printed in parentheses. When *searchstr* is in a head synset, all of the head synset's satellites are also displayed. The position of an adjective in relation to the noun may be restricted to the *prenominal* , *postnominal* or *predicative* position. Where present, these restrictions are noted in parentheses.

When an adjective is a participle of a verb, the output indicates the verb and displays its synset.

When an adverb is derived from an adjective, the specific adjectival sense on which it is based is indicated.

WordNet Documentation

The morphological transformations performed by the search code may result in more than one word to search for. WordNet automatically performs the requested search on all of the strings and returns the results grouped by word. For example, the verb **saw** is both the present tense of **saw** and the past tense of **see** . When passed *searchstr* **saw** , WordNet performs the desired search first on **saw** and next on **see** , returning the list of **saw** senses and search results, followed by those for **see** .

Software library functions

Introduction to WordNet library functions

Functions are organized into the following categories:

Category	Manual Page	Object File
Database Search	wnsearch (3WN)	search.o
Morphology	morph (3WN)	morph.o
Misc. Utility	wnutil (3WN)	wnutil.o
Binary Search	binsrch (3WN)	binsrch.o

The WordNet library is used by all of the searching interfaces provided with the various WordNet packages. Additional programs in the system, such as **grind**, also use functions in this library.

The WordNet library is provided in both source and binary forms (on some platforms) to allow users to build applications and tools to their own specifications that utilize the WordNet database. We do not provide programming support or assistance.

The code conforms to ANSI C standards. Functions are defined with function prototypes. If you do not have a compiler that accepts prototypes, you must edit the source code and remove the prototypes before compiling.

LIST OF WORDNET LIBRARY FUNCTIONS

Not all library functions are listed below. Missing are mainly functions that are called by documented ones, or ones that were written for specific applications or tools used during WordNet development. Data structures are defined in **wn.h** .

WordNet Documentation

Database Searching Functions (search.o)

findtheinfo
: Primary search function for WordNet database. Returns formatted search results in text buffer. Used by WordNet interfaces to perform requested search.

findtheinfo_ds
: Primary search function for WordNet database. Returns search results in linked list data structure.

is_defined
: Set bit for each search type that is valid for the search word passed and return bit mask.

in_wn
: Set bit for each syntactic category that search word is in.

index_lookup
: Find word in index file and return parsed entry in data structure. Input word must be exact match of string in database. Called by **getindex()** .

getindex
: Find word in index file, trying different techniques - replace hyphens with underscores, replace underscores with hyphens, strip hyphens and underscores, strip periods.

read_synset
: Read synset from data file at byte offset passed and return parsed entry in data structure. Calls **parse_synset()** .

parse_synset
: Read synset at current byte offset in file and return parsed entry in data structure.

free_syns
: Free a synset linked list allocated by **findtheinfo_ds()** .

free_synset
: Free a synset structure.

free_index
: Free an index structure.

traceptrs_ds
: Recursive search algorithm to trace a pointer tree and return results in linked list.

do_trace
: Do requested search on synset passed returning formatted output in buffer.

Morphology Functions (morph.o)

morphinit
: Open exception list files.

re_morphinit
: Close exception list files and reopen.

morphstr
: Try to find base form (lemma) of word or collocation in syntactic category passed. Calls **morphword()** for each word in string passed.

morphword

Try to find base form (lemma) of individual word in syntactic category passed.

Utility Functions (wnutil.o)

wninit

Top level function to open database files and morphology exception lists: opens the files necessary for using WordNet with the WordNet library functions. The database files are opened, and **morphinit()** is called to open the exception list files. Returns **0** if successful, **-1** otherwise. The database and exception list files must be open before the WordNet search and morphology functions are used. If the database is successfully opened, the global variable **OpenDB** is set to **1** . Note that it is possible for the database files to be opened (**OpenDB == 1**), but not the exception list files.

re_wninit

Top level function to close and reopen database files and morphology exception lists: is used to close the database files and reopen them, and is used exclusively for WordNet development. **re_morphinit()** is called to close and reopen the exception list files. Return codes are as described above.

cntwords

Count the number of underscore or space separated words in a string: counts the number of underscore or space separated words in *str* . A hyphen is passed in *separator* if is is to be considered a word delimiter. Otherwise *separator* can be any other character, or an underscore if another character is not desired.

strtolower

Convert string to lower case and remove trailing adjective marker if found: converts *str* to lower case and removes a trailing adjective marker, if present. *str* is actually modified by this function, and a pointer to the modified string is returned.

ToLowerCase

Convert string passed to lower case: converts *str* to lower case as above, without removing an adjective marker.

strsubst

Replace all occurrences of *from* with *to* in *str :* replaces all occurrences of from with to in str and returns resulting string.

getptrtype

Return code for pointer type character passed: returns the integer *ptr_type* corresponding to the pointer character passed in *ptr_symbol* .

getpos

Return syntactic category code for string passed: returns the integer constant corresponding to the synset type passed. *ss_type* may be one of the following: **n, v, a, r, s** . If **s** is passed, **ADJ** is returned. Exits with **-1** if *ss_type* is invalid.

getsstype

Return synset type code for string passed: works like **getpos()** , but returns **SATELLITE** if *ss_type* is **s** .

FmtSynset
Reconstruct synset string from synset pointer.

StrToPos
Passed string for syntactic category, returns corresponding integer value: returns the integer constant corresponding to the syntactic category passed in *pos* . *string* must be one of the following: **noun, verb, adj, adv** . **-1** is returned if *pos* is invalid.

GetSynsetForSense
Return synset for sense key passed: returns the synset that contains the word sense *sense_key* and **NULL** in case of error.

GetDataOffset
Find synset offset for sense: returns the synset offset for synset that contains the word sense *sense_key* , and **0** if *sense_key* is not in sense index file

GetPolyCount
Find polysemy count for sense passed: returns the polysemy count (number of senses in WordNet) for *lemma* encoded in *sense_key* and **0** if word is not found.

GetWORD
Return word part of sense key.

GetPOS
Return syntactic category code for sense key passed.

WNSnsToStr
Generate sense key for index entry passed: returns sense key encoding for *sense_num* entry in *idx* .

GetValidIndexPointer
Search for string and/or base form of word in database and return index structure for word if found: returns the Index structure for *word* in *pos* . Calls **morphstr** to find a valid base form if *word* is inflected

GetWNSense
Return sense number in database for sense key: returns the WordNet sense number for the sense key encoding represented by *lemma* and *lex_sense* .

GetSenseIndex
Return parsed sense index entry for sense key passed: returns parsed sense index entry for *sense_key* and **NULL** if *sense_key* is not in sense index.

default_display_message
Default function to use as value of **display_message** . Simply returns **-1** .This is the default value for the global variable **display_message** , that points to a function to call to display an error message. In general, applications (including the WordNet interfaces) define an application specific function and set **display_message** to point to it.

GetTagcnt() returns the number of times the sense passed has been tagged according to the *cntlist* file.

include/wn.h lists all the pointer and search types and their corresponding constant values. There is no description of what each search type is or the results returned. Using the WordNet interface is the best way to see what types of searches are available, and the data returned for each.

Error checking on passed arguments is not rigorous. Passing **NULL** pointers or invalid values will often cause an application to die.

Binary Search Functions (binsrch.o)

bin_search
: General purpose binary search function to search for key as first item on line in sorted file.

copyfile
: Copy contents from one file to another.

replace_line
: Replace a line in a sorted file.

insert_line
: Insert a line into a sorted file.

HEADER FILE

wn.h
: WordNet include file of constants, data structures, external declarations for global variables initialized in **wnglobal.c** . Also lists function prototypes for library API. It must be included to use any WordNet library functions.

NOTES

All library functions that access the database files expect the files to be open. The function **wninit** must be called before other database access functions such as **findtheinfo** or **read_synset**.

Inclusion of the header file **wn.h** is necessary.

The command line interface is a good example of a simple application that uses several WordNet library functions.

Many of the library functions are passed or return syntactic category or synset type information. The following table lists the possible categories as integer codes, synset type constant names, syntactic category constant names, single characters and character strings.

Integer	Synset Type	Syntactic Category	Char	String
1	NOUN	NOUN	n	noun

2	VERB	VERB	v	verb
3	ADJ	ADJ	a	adj
4	ADV	ADV	r	adv
5	SATELLITE	ADJ	s	*n/a*

FILES

lib/libwn.a
WordNet library (Unix)
lib\wn.lib
WordNet library (Windows)
include
header files for use with WordNet library

binary search and modifying sorted files

char *bin_search(char *key, FILE *fp);

void copyfile(FILE *fromfp, FILE *tofp);

char *replace_line(char *new_line, char *key, FILE *fp);

char *insert_line(char *new_line, char *key, FILE *fp);

The WordNet library contains several general purpose functions for performing a binary search and modifying sorted files.

bin_search() is the primary binary search algorithm to search for *key* as the first item on a line in the file pointed to by *fp* . The delimiter between the key and the rest of the fields on the line, if any, must be a space. A pointer to a static variable containing the entire line is returned. **NULL** is returned if a match is not found.

The remaining functions are not used by WordNet, and are only briefly described.

copyfile() copies the contents of one file to another.

replace_line() replaces a line in a file having searchkey *key* with the contents of *new_line* . It returns the original line or **NULL** in case of error.

insert_line() finds the proper place to insert the contents of *new_line* , having searchkey *key* in the sorted file pointed to by *fp* . It returns **NULL** if a line with this searchkey is already in the file.

The maximum length of *key* is 1024.
The maximum line length in a file is 25K.

If there are no additional fields after the search key, the key must be followed by at least one space before the newline character.

binsearch() returns a pointer to a static character buffer. The returned string should be copied by the caller if the results need to be saved, as a subsequent call will replace the contents of the static buffer.

Database files

WordNet 3.0 database statistics

Number of words, synsets, and senses

POS	Unique Strings	Synsets	Total Word-Sense Pairs
Noun	117798	82115	146312
Verb	11529	13767	25047
Adjective	21479	18156	30002
Adverb	4481	3621	5580
Totals	155287	117659	206941

Polysemy information

POS	Monosemous Words and Senses	Polysemous Words	Polysemous Senses
Noun	101863	15935	44449
Verb	6277	5252	18770
Adjective	16503	4976	14399
Adverb	3748	733	1832
Totals	128391	26896	79450

POS	Average Polysemy	Average Polysemy

	Including Monosemous Words	Excluding Monosemous Words
Noun	1.24	2.79
Verb	2.17	3.57
Adjective	1.40	2.71
Adverb	1.25	2.50

NOTES

Statistics for all types of adjectives and adjective satellites are combined.

The total of all unique noun, verb, adjective, and adverb strings is actually 147278. However, many strings are unique within a syntactic category, but are in more than one syntactic category. The figures in the table represent the unique strings in each syntactic category.

WordNet files

cntlist - format of **cntlist** and **cntlist.rev** files

lexnames - list of lexicographer file names and numbers

prologdb - description of Prolog database files

senseidx - format of sense index file

sensemap - mapping from senses in WordNet 2.1 to corresponding 3.0 senses

wndb - format of WordNet database files

wninput - format of WordNet lexicographer files

All files are in ASCII. Fields are generally separated by one space, unless otherwise noted, and each line is terminated with a newline character. In the file format descriptions, terms in *italics* refer to field names. Characters or strings in **boldface** represent an actual character or string as it appears in the file. Items enclosed in italicized square brackets (*[ ]*) may not be present. Since several files contain fields that have the identical meaning, field names are consistently defined. For example, several WordNet files contain one or more *synset_offset* fields. In each case, the definition of *synset_offset* is identical.

WordNet Documentation

cntlist file

cntlist - file listing number of times each tagged sense occurs in a semantic concordance, sorted most to least frequently tagged

cntlist.rev - file listing number of times each tagged sense occurs in a semantic concordance, sorted by sense key

A cntlist file for a semantic concordance lists the number of times each semantically tagged sense occurs in the concordance and its sense number in the WordNet database. Each line in the file corresponds to a sense in the WordNet database to which at least one semantic tag points. Only senses that are tagged in a concordance are in the concordance's cntlist file.

In the WordNet database, words are assigned sense numbers based on frequency of use in semantically tagged corpora. The cntlist file used by **grind** to build the WordNet database and assign the sense numbers is a union of the cntlist files from the various semantic concordances that were formerly released by Princeton University. This combined cntlist file is provided with the WordNet package and is found in the **WNSEARCHDIR** directory.

The *cntlist.rev* file is used at run-time by the WordNet library code and browser interfaces to print in the output display the number of times each sense has been tagged.

File format

Each line in a cntlist file contains information for one sense. The file is ordered from most to least frequently tagged sense. The fields are separated by one space, and each line is terminated with a newline character. Senses having the same *tag_cnt* value are listed in reverse alphabetical order of the *lemma* field of the *sense_key* .

Each line in **cntlist** is of the form:

tag_cnt sense_key sense_number

where *tag_cnt* is the decimal number of times the sense is tagged in the corresponding semantic concordance. *sense_key* is a WordNet sense encoding and *sense_number* is a WordNet sense number as described in

The *cntlist.rev* file contains the same fields described above, in the following order:

sense_key sense_number tag_cnt

WordNet Documentation

Princeton no longer maintains or releases the Semantic Concordance files. The *cntlist* file used to order the senses in WordNet 3.0 was generated from the Semantic Concordance files at the point that they were last updated in 2001. In general, the order of senses presented usually reflects what the user would expect, however sense ordering is now less reliable than in prior releases and should not be construed as an accurate indicator of frequency of use.

WordNet database files in detail

index.noun, data.noun, index.verb, data.verb, index.adj, data.adj, index.adv, data.adv - WordNet database files
noun.exc, verb.exc. adj.exc adv.exc - morphology exception lists

sentidx.vrb, sents.vrb - files used by search code to display sentences illustrating the use of some specific verbs

For each syntactic category, two files are needed to represent the contents of the WordNet database - **index.** *pos* and **data.** *pos* , where *pos* is **noun** , **verb** , **adj** and **adv** . The other auxiliary files are used by the WordNet library's searching functions and are needed to run the various WordNet browsers. Each index file is an alphabetized list of all the words found in WordNet in the corresponding part of speech. On each line, following the word, is a list of byte offsets (*synset_offset* s) in the corresponding data file, one for each synset containing the word. Words in the index file are in lower case only, regardless of how they were entered in the lexicographer files. This folds various orthographic representations of the word into one line enabling database searches to be case insensitive.

A data file for a syntactic category contains information corresponding to the synsets that were specified in the lexicographer files, with relational pointers resolved to *synset_offset* s. Each line corresponds to a synset. Pointers are followed and hierarchies traversed by moving from one synset to another via the *synset_offset* s.

The exception list files, *pos* **.exc** , are used to help the morphological processor find base forms from irregular inflections.

The files **sentidx.vrb** and **sents.vrb** contain sentences illustrating the use of specific senses of some verbs. These files are used by the searching software in response to a request for verb sentence frames. Generic sentence frames are displayed when an illustrative sentence is not present.

The various database files are in ASCII formats that are easily read by both humans and machines. All fields, unless otherwise noted, are separated by one space character, and all lines are terminated by a newline character. Fields enclosed in italicized square brackets may not be present.

WordNet Documentation

Index file format

Each index file begins with several lines containing a copyright notice, version number and license agreement. These lines all begin with two spaces and the line number so they do not interfere with the binary search algorithm that is used to look up entries in the index files. All other lines are in the following format. In the field descriptions, **number** always refers to a decimal integer unless otherwise defined.

lemma pos synset_cnt p_cnt [ptr_symbol...] sense_cnt tagsense_cnt
synset_offset [synset_offset...]

lemma
lower case ASCII text of word or collocation. Collocations are formed by joining individual words with an underscore (_) character.

pos
Syntactic category: **n** for noun files, **v** for verb files, **a** for adjective files, **r** for adverb files.

All remaining fields are with respect to senses of *lemma* in *pos* .

synset_cnt
Number of synsets that *lemma* is in. This is the number of senses of the word in WordNet. See **Sense Numbers** below for a discussion of how sense numbers are assigned and the order of *synset_offset* s in the index files.

p_cnt
Number of different pointers that *lemma* has in all synsets containing it.

ptr_symbol
A space separated list of *p_cnt* different types of pointers that *lemma* has in all synsets containing it. If all senses of *lemma* have no pointers, this field is omitted and *p_cnt* is **0** .

sense_cnt
Same as *sense_cnt* above. This is redundant, but the field was preserved for compatibility reasons.

tagsense_cnt
Number of senses of *lemma* that are ranked according to their frequency of occurrence in semantic concordance texts.

synset_offset
Byte offset in **data.*pos*** file of a synset containing *lemma* . Each *synset_offset* in the list corresponds to a different sense of *lemma* in WordNet. *synset_offset* is an 8 digit, zero-filled decimal integer that can be used with **fseek** to read a synset from the data file. When passed to **read_synset** along with the syntactic category, a data structure containing the parsed synset is returned.

Data file format

Each data file begins with several lines containing a copyright notice, version number and license agreement. These lines all begin with two spaces and the line number. All other lines are in the following format. Integer fields are of fixed length, and are zero-filled.

synset_offset lex_filenum ss_type w_cnt word lex_id [word lex_id...] p_cnt [ptr...] [frames...] | gloss

synset_offset
: Current byte offset in the file represented as an 8 digit decimal integer.

lex_filenum
: Two digit decimal integer corresponding to the lexicographer file name containing the synset.

ss_type
: One character code indicating the synset type:

n NOUN
v VERB
a ADJECTIVE
s ADJECTIVE SATELLITE
r ADVERB

w_cnt
: Two digit hexadecimal integer indicating the number of words in the synset.

word
: ASCII form of a word as entered in the synset by the lexicographer, with spaces replaced by underscore characters (_). The text of the word is case sensitive, in contrast to its form in the corresponding **index.** *pos* file, that contains only lower-case forms. In **data.adj** , a *word* is followed by a syntactic marker if one was specified in the lexicographer file. A syntactic marker is appended, in parentheses, onto *word* without any intervening spaces.

lex_id
: One digit hexadecimal integer that, when appended onto *lemma* , uniquely identifies a sense within a lexicographer file. *lex_id* numbers usually start with **0** , and are incremented as additional senses of the word are added to the same file, although there is no requirement that the numbers be consecutive or begin with **0** . Note that a value of **0** is the default, and therefore is not present in lexicographer files.

p_cnt
: Three digit decimal integer indicating the number of pointers from this synset to other synsets. If *p_cnt* is **000** the synset has no pointers.

ptr
: A pointer from this synset to another. *ptr* is of the form:

pointer_symbol synset_offset pos source/target

where *synset_offset* is the byte offset of the target synset in the data file corresponding to *pos* .

The *source/target* field distinguishes lexical and semantic pointers. It is a four byte field, containing two two-digit hexadecimal integers. The first two digits indicates the word number in the current (source) synset, the last two digits indicate the word number in the target synset. A value of **0000** means that *pointer_symbol* represents a semantic relation between the current (source) synset and the target synset indicated by *synset_offset* .

A lexical relation between two words in different synsets is represented by non-zero values in the source and target word numbers. The first and last two bytes of this field indicate the word numbers in the source and target synsets, respectively, between which the relation holds. Word numbers are assigned to the *word* fields in a synset, from left to right, beginning with **1** .

frames

In **data.verb** only, a list of numbers corresponding to the generic verb sentence frames for *word* s in the synset. *frames* is of the form:

f_cnt + f_num w_num [+ f_num w_num...]

where *f_cnt* a two digit decimal integer indicating the number of generic frames listed, *f_num* is a two digit decimal integer frame number, and *w_num* is a two digit hexadecimal integer indicating the word in the synset that the frame applies to. As with pointers, if this number is **00** , *f_num* applies to all *word* s in the synset. If non-zero, it is applicable only to the word indicated. Word numbers are assigned as described for pointers. Each *f_num w_num* pair is preceded by a + .

gloss

Each synset contains a gloss. A *gloss* is represented as a vertical bar (|), followed by a text string that continues until the end of the line. The gloss may contain a definition, one or more example sentences, or both.

Sense numbers

Senses in WordNet are generally ordered from most to least frequently used, with the most common sense numbered **1** . Frequency of use is determined by the number of times a sense is tagged in the various semantic concordance texts. Senses that are not semantically tagged follow the ordered senses. The *tagsense_cnt* field for each entry in the **index.*pos*** files indicates how many of the senses in the list have been tagged.

The **cntlist** file provided with the database lists the number of times each sense is tagged in the semantic concordances. The data from **cntlist** is used by **grind** to order the senses of each word. When

the **index** *.pos* files are generated, the *synset_offset* s are output in sense number order, with sense 1 first in the list. Senses with the same number of semantic tags are assigned unique but consecutive sense numbers. The WordNet **OVERVIEW** search displays all senses of the specified word, in all syntactic categories, and indicates which of the senses are represented in the semantically tagged texts.

Exception list file format

Exception lists are alphabetized lists of inflected forms of words and their base forms. The first field of each line is an inflected form, followed by a space separated list of one or more base forms of the word. There is one exception list file for each syntactic category.

Note that the noun and verb exception lists were automatically generated from a machine-readable dictionary, and contain many words that are not in WordNet. Also, for many of the inflected forms, base forms could be easily derived using the standard rules of detachment programmed into Morphy. These anomalies are allowed to remain in the exception list files, as they do no harm.

Verb example sentences

For some verb senses, example sentences illustrating the use of the verb sense can be displayed. Each line of the file **sentidx.vrb** contains a *sense_key* followed by a space and a comma separated list of example sentence template numbers, in decimal. The file **sents.vrb** lists all of the example sentence templates. Each line begins with the template number followed by a space. The rest of the line is the text of a template example sentence, with **%s** used as a placeholder in the text for the verb. Both files are sorted alphabetically so that the *sense_key* and template sentence number can be used as indices, via **binsearch**, into the appropriate file.

When a request for **FRAMES** is made, the WordNet search code looks for the sense in **sentidx.vrb** . If found, the sentence template(s) listed is retrieved from **sents.vrb** , and the **%s** is replaced with the verb. If the sense is not found, the applicable generic sentence frame(s) listed in *frames* is displayed.

Information in the **data.*pos*** and **index.*pos*** files represents all of the word senses and synsets in the WordNet database. The *word* , *lex_id* , and *lex_filenum* fields together uniquely identify each word sense in WordNet. These can be encoded in a *sense_key*. Each synset in the database can be uniquely identified by combining the *synset_offset* for the synset with a code for the syntactic category (since it is possible for synsets in different **data.*pos*** files to have the same *synset_offset*).

The WordNet system provide both command line and window-based browser interfaces to the database. Both interfaces utilize a common library of search and morphology code. The source code for the library and interfaces is included in the WordNet package.

index.*pos*

database index files

data.*pos*

database data files

***.vrb**

files of sentences illustrating the use of verbs

***pos* .exc**

morphology exception lists

lexicographer source files

WordNet's source files are written by lexicographers. They are the product of a detailed relational analysis of lexical semantics: a variety of lexical and semantic relations are used to represent the organization of lexical knowledge. Two kinds of building blocks are distinguished in the source files: word forms and word meanings. Word forms are represented in their familiar orthography; word meanings are represented by synonym sets (*synset* s) - lists of synonymous word forms that are interchangeable in some context. Two kinds of relations are recognized: lexical and semantic. Lexical relations hold between word forms; semantic relations hold between word meanings.
Lexicographer files correspond to the syntactic categories implemented in WordNet - noun, verb, adjective and adverb. All of the synsets in a lexicographer file are in the same syntactic category. Each synset consists of a list of synonymous words or collocations (eg. **"fountain pen"** , **"take in"**), and pointers that describe the relations between this synset and other synsets. These relations include (but are not limited to) hypernymy/hyponymy, antonymy, entailment, and meronymy/holonymy. A word or collocation may appear in more than one synset, and in more than one part of speech. Each use of a word in a synset represents a sense of that word in the part of speech corresponding to the synset.

Adjectives may be organized into clusters containing head synsets and satellite synsets. Adverbs generally point to the adjectives from which they are derived.

Lexicographer file names
The names of the lexicographer files are of the form:

pos.*suffix*

where *pos* is either **noun** , **verb** , **adj** or **adv** . *suffix* may be used to organize groups of synsets into different files, for example **noun.animal** and **noun.plant** .

Pointers

Pointers are used to represent the relations between the words in one synset and another. Semantic pointers represent relations between word meanings, and therefore pertain to all of the words in the source and target synsets. Lexical pointers represent relations between word forms, and pertain only to

specific words in the source and target synsets. The following pointer types are usually used to indicate lexical relations: Antonym, Pertainym, Participle, Also See, Derivationally Related. The remaining pointer types are generally used to represent semantic relations.

A relation from a source to a target synset is formed by specifying a word from the target synset in the source synset, followed by the *pointer_symbol* indicating the pointer type. The location of a pointer within a synset defines it as either lexical or semantic. The **Lexicographer File Format** section describes the syntax for entering a semantic pointer, and **Word Syntax** describes the syntax for entering a lexical pointer.

Although there are many pointer types, only certain types of relations are permitted between synsets of each syntactic category.

The *pointer_symbol* s for nouns are:

! Antonym
@ Hypernym
@i Instance Hypernym
Hyponym
i Instance Hyponym
#m Member holonym
#s Substance holonym
#p Part holonym
%m Member meronym
%s Substance meronym
%p Part meronym
= Attribute
+ Derivationally related form
;c Domain of synset - TOPIC
-c Member of this domain - TOPIC
;r Domain of synset - REGION
-r Member of this domain - REGION
;u Domain of synset - USAGE
-u Member of this domain - USAGE

The *pointer_symbol* s for verbs are:

! Antonym
@ Hypernym
Hyponym
***** Entailment

> Cause
^ Also see
$ Verb Group
+ Derivationally related form
;c Domain of synset - TOPIC
;r Domain of synset - REGION
;u Domain of synset - USAGE

The *pointer_symbol* s for adjectives are:

! Antonym
& Similar to
< Participle of verb
\ Pertainym (pertains to noun)
= Attribute
^ Also see
;c Domain of synset - TOPIC
;r Domain of synset - REGION
;u Domain of synset - USAGE

The *pointer_symbol* s for adverbs are:

! Antonym
\ Derived from adjective
;c Domain of synset - TOPIC
;r Domain of synset - REGION
;u Domain of synset - USAGE

Many pointer types are reflexive, meaning that if a synset contains a pointer to another synset, the other synset should contain a corresponding reflexive pointer. **Grind** automatically inserts missing reflexive pointers for the following pointer types:

Pointer	Reflect
Antonym	Antonym
Hyponym	Hypernym
Hypernym	Hyponym
Instance Hyponym	Instance Hypernym
Instance Hypernym	Instance Hyponym
Holonym	Meronym
Meronym	Holonym

Similar to	Similar to
Attribute	Attribute
Verb Group	Verb Group
Derivationally Related	Derivationally Related
Domain of synset	Member of Doman

Verb Frames

Each verb synset contains a list of generic sentence frames illustrating the types of simple sentences in which the verbs in the synset can be used. For some verb senses, example sentences illustrating actual uses of the verb are provided. Whenever there is no example sentence, the generic sentence frames specified by the lexicographer are used. The generic sentence frames are entered in a synset as a comma-separated list of integer frame numbers. The following list is the text of the generic frames, preceded by their frame numbers:

1 Something ----s
2 Somebody ----s
3 It is ----ing
4 Something is ----ing PP
5 Something ----s something Adjective/Noun
6 Something ----s Adjective/Noun
7 Somebody ----s Adjective
8 Somebody ----s something
9 Somebody ----s somebody
10 Something ----s somebody
11 Something ----s something
12 Something ----s to somebody
13 Somebody ----s on something
14 Somebody ----s somebody something
15 Somebody ----s something to somebody
16 Somebody ----s something from somebody
17 Somebody ----s somebody with something
18 Somebody ----s somebody of something
19 Somebody ----s something on somebody
20 Somebody ----s somebody PP
21 Somebody ----s something PP
22 Somebody ----s PP
23 Somebody's (body part) ----s
24 Somebody ----s somebody to INFINITIVE

25 Somebody ----s somebody INFINITIVE
26 Somebody ----s that CLAUSE
27 Somebody ----s to somebody
28 Somebody ----s to INFINITIVE
29 Somebody ----s whether INFINITIVE
30 Somebody ----s somebody into V-ing something
31 Somebody ----s something with something
32 Somebody ----s INFINITIVE
33 Somebody ----s VERB-ing
34 It ----s that CLAUSE
35 Something ----s INFINITIVE

Lexicographer File Format

Synsets are entered one per line, and each line is terminated with a newline character. A line containing a synset may be as long as necessary, but no newlines can be entered within a synset. Within a synset, spaces or tabs may be used to separate entities. Items enclosed in italicized square brackets may not be present.

The general synset syntax is:

{ *words pointers* **(** *gloss* **)** **}**

Synsets of this form are valid for all syntactic categories except verb, and are referred to as basic synsets. At least one *word* and a *gloss* are required to form a valid synset. Pointers entered following all the *words* in a synset represent semantic relations between all the words in the source and target synsets.

For verbs, the basic synset syntax is defined as follows:

{ *words pointers frames* **(** *gloss* **)** **}**

Adjective may be organized into clusters containing one or more head synsets and optional satellite synsets. Adjective clusters are of the form:

[
head synset
[satellite synsets]
[-]
[additional head/satellite synsets]
]

Each adjective cluster is enclosed in square brackets, and may have one or more parts. Each part consists of a head synset and optional satellite synsets that are conceptually similar to the head synset's meaning. Parts of a cluster are separated by one or more hyphens (-) on a line by themselves, with the terminating square bracket following the last synset. Head and satellite synsets follow the syntax of basic synsets, however a "Similar to" pointer must be specified in a head synset for each of its satellite synsets. Most adjective clusters contain two antonymous parts.

Synsets for relational adjectives (pertainyms) and participial adjectives do not adhere to the cluster structure. They use the basic synset syntax.

Comments can be entered in a lexicographer file by enclosing the text of the comment in parentheses. Note that comments **cannot** appear within a synset, as parentheses within a synset have an entirely different meaning (see **Gloss Syntax**). However, entire synsets (or adjective clusters) can be "commented out" by enclosing them in parentheses. This is often used by the lexicographers to verify the syntax of files under development or to leave a note to oneself while working on entries.

Word Syntax

A synset must have at least one word, and the words of a synset must appear after the opening brace and before any other synset constructs. A word may be entered in either the simple word or word/pointer syntax.

A simple word is of the form:

word[**(** *marker* **)** *][lex_id]* **,**

word may be entered in any combination of upper and lower case unless it is in an adjective cluster. A collocation is entered by joining the individual words with an underscore character (_). Numbers (integer or real) may be entered, either by themselves or as part of a word string, by following the number with a double quote (").

See **Special Adjective Syntax** for a description of adjective clusters and markers.

word may be followed by an integer *lex_id* from **1** to **15** . The *lex_id* is used to distinguish different senses of the same word within a lexicographer file. The lexicographer assigns *lex_id* values, usually in ascending order, although there is no requirement that the numbers be consecutive. The default is **0** , and does not have to be specified. A *lex_id* must be used on pointers if the desired sense has a non-zero *lex_id* in its synset specification.

Word/pointer syntax is of the form:

[*word[* **(** *marker* **)** *][lex_id]* **,** *pointers* **]**

This syntax is used when one or more pointers correspond only to the specific word in the word/pointer set, rather than all the words in the synset, and represents a lexical relation. Note that a word/pointer set appears within a synset, therefore the square brackets used to enclose it are treated differently from those used to define an adjective cluster. Only one word can be specified in each word/pointer set, and any number of pointers may be included. A synset can have any number of word/pointer sets. Each is treated by **grind** essentially as a *word* , so they all must appear before any synset *pointers* representing semantic relations.

For verbs, the word/pointer syntax is extended in the following manner to allow the user to specify generic sentence frames that, like pointers, correspond only to a specific word, rather than all the words in the synset. In this case, *pointers* are optional.

[*word* **,** *[pointers] frames* **]**

Pointer Syntax

Pointers are optional in synsets. If a pointer is specified outside of a word/pointer set, the relation is applied to all of the words in the synset, including any words specified using the word/pointer syntax. This indicates a semantic relation between the meanings of the words in the synsets. If specified within a word/pointer set, the relation corresponds only to the word in the set and represents a lexical relation.

A pointer is of the form:

[lex_filename **:** *]word[lex_id]* **,** *pointer_symbol*

or:

[lex_filename **:** *]word[lex_id]* ^ *word[lex_id]* **,** *pointer_symbol*

For pointers, *word* indicates a word in another synset. When the second form of a pointer is used, the first *word* indicates a word in a head synset, and the second is a word in a satellite of that cluster. *word* may be followed by a *lex_id* that is used to match the pointer to the correct target synset. The synset containing *word* may reside in another lexicographer file. In this case, *word* is preceded by *lex_filename* as shown.

See **Pointers** for a list of *pointer_symbol* s and their meanings.

Verb Frame List Syntax

Frame numbers corresponding to generic sentence frames must be entered in each verb synset. If a frame list is specified outside of a word/pointer set, the verb frames in the list apply to all of the words in the synset, including any words specified using the word/pointer syntax. If specified within a word/pointer set, the verb frames in the list correspond only to the word in the set.

A frame number list is entered as follows:

frames: *f_num* [**,** *f_num...]*

Where *f_num* specifies a generic frame number. See **Verb Frames** for a list of generic sentences and their corresponding frame numbers.

Gloss Syntax

A gloss is included in all synsets. The lexicographer may enter a text string of any length desired. A gloss is simply a string enclosed in parentheses with no embedded carriage returns. It provides a definition of what the synset represents and/or example sentences.

Special Adjective Syntax

The syntax for representing antonymous adjective synsets requires several additional conditions.

The first word of a head synset **must** be entered in upper case, and can be thought of as the head word of the head synset. The *word* part of a pointer from one head synset to another head synset within the same cluster (usually an antonym) must also be entered in upper case. Usually antonymous adjectives are entered using the word/pointer syntax described in **Word Syntax** to indicate a lexical relation. There is no restriction on the number of parts that a cluster may have, and some clusters have three parts, representing antonymous triplets, such as **solid** , **liquid** , and **gas** .

A cross-cluster pointer may be specified, allowing a head or satellite synset to point to a head synset in a different cluster. A cross-cluster pointer is indicated by entering the *word* part of the pointer in upper case.

An adjective may be annotated with a syntactic marker indicating a limitation on the syntactic position the adjective may have in relation to noun that it modifies. If so marked, the marker appears between the word and its following comma. If a *lex_id* is specified, the marker immediately follows it. The syntactic markers are:

(p) predicate position
(a) prenominal (attributive) position
(ip) immediately postnominal position

EXAMPLES

(Note that these are hypothetical examples not found in the WordNet lexicographer files.)

Sample noun synsets:

```
{ canine, [ dog1, cat,! ] pooch, canid,@ }
{ collie, dog1,@ (large multi-colored dog with pointy nose) }
{ hound, hunting_dog, pack,#m dog1,@ }
{ dog, }
```

Sample verb synsets:

```
{ [ confuse, clarify,! frames: 1 ] blur, obscure, frames: 8, 10 }
{ [ clarify, confuse,! ] make_clear, interpret,@ frames: 8 }
{ interpret, construe, understand,@ frames: 8 }
```

Sample adjective clusters:

```
[
{ [ HOT, COLD,! ] lukewarm(a), TEPID,^ (hot to the touch) }
{ warm, }
-
{ [ COLD, HOT,! ] frigid, (cold to the touch) }
{ freezing, }
]
```

Sample adverb synsets:

```
{ [ basically, adj.all:essential^basic,\ ] [ essentially, adj.all:basic^fundamental,\ ] ( by one's
very nature )}
{ pointedly, adj.all:pungent^pointed,\ }
{ [ badly, adj.all:bad,\ well,! ] ill, ("He was badly prepared") }
```

grind

grind - process WordNet lexicographer files

grind [**-v**] [**-s**] [**-L** *logfile*] [**-a**] [**-d**] [**-i**] [**-o**] [**-n**] *filename* [*filename ...*]

WordNet Documentation

grind() processes WordNet lexicographer files, producing database files suitable for use with the WordNet search and interface code and other applications. The syntactic and structural integrity of the input files is verified. Warnings and errors are reported via **stderr** and a run-time log is produced on **stdout** . A database is generated only if there are no errors.

Input files correspond to the syntactic categories implemented in WordNet - **noun**, **verb**, **adjective** and **adverb**. Each input lexicographer file consists of a list of synonym sets (*synsets*) for one part of speech. Although the basic synset syntax is the same for all of the parts of speech, some parts of the syntax only apply to a particular part of speech.

Each *filename* specified is of the form:

where *pathname* is optional and *pos* is either **noun**, **verb**, **adj** or **adv**. *suffix* may be used to separate groups of synsets into different files, for example **noun.animal** and **noun.plant** . One or more input files, in any combination of syntactic categories, may be specified.

grind() produces the following output files:

Filename	Description
index.*pos*	Index file for each syntactic category
data.*pos*	Data file for each syntactic category
index.sense	Sense index

Each time **grind()** is run, any existing database files are overwritten with the database files generated from the specified input files. If no input files from a syntactic category are specified, the corresponding database files are not overwritten.

Senses are generally ordered from most to least frequently used, with the most common sense numbered **1** . Frequency of use is determined by the number of times a sense is tagged in the various semantic concordance texts. Senses that are not semantically tagged follow the ordered senses in an arbitrary order. Note that this ordering is only an estimate based on usage in a small corpus.
The *tagsense_cnt* field for each entry in the **index.*pos*** files indicates how many of the senses in the list have been tagged.

The **cntlist** file provided with the database lists the number of times each sense is tagged in the semantic concordances. **grind()** uses the data from **cntlist** to order the senses of each word. When the **index** .*pos* files are generated, the *synset_offset* s are output in sense number order, with sense 1 first in the list. Senses with the same number of semantic tags are assigned unique but consecutive sense

numbers. The WordNet **OVERVIEW** search displays all senses of the specified word, in all syntactic categories, and indicates which of the senses are represented in the semantically tagged texts.

options

-v
: Verify integrity of input without generating database.

-s
: Suppress generation of warning messages. Usually **grind** is run with this option until all syntactic and structural errors are corrected since the warning messages may make it difficult to spot error messages.

-L*logfile*
: Write all messages to *logfile* instead of **stderr** .

-a
: Generate statistical report on input files processed.

-d
: Generate distribution of senses by string length report on input files processed.

-i
: Generate sense index file.

-o
: Order senses using **cntlist** .

-n
: Generate nominalization (derivational morphology) links in database.

filename
: Input file of the form described in **Input**

files

pos* .
: lexicographer files to use to build database

cntlist
: file of combined semantic concordance **cntlist** files. Used to assign sense numbers in WordNet database

Exit status is normally 0. Exit status is -1 if non-specific error occurs. If syntactic or structural errors exist, exit status is number of errors detected.

usage: grind [-v] [-s] [-Llogfile] [-a] [-d] [-i] [-o] [-n] filename [filename...]
: Invalid options were specified on the command line.

No input files processed.
: None of the filenames specified were of the appropriate form.

***n* syntactic errors found.**
: Syntax errors were found while parsing the input files.

***n* structural errors found.**

Pointer errors were found that could not be automatically corrected.

lexicographer file names and numbers

List of WordNet lexicographer file names and numbers

During WordNet development synsets are organized into forty-five lexicographer files based on syntactic category and logical groupings. **grind** processes these files and produces a database suitable for use with the WordNet library, interface code, and other applications.

A file number corresponds to each lexicographer file. File numbers are encoded in several parts of the WordNet system as an efficient way to indicate a lexicographer file name. The file **lexnames** lists the mapping between file names and numbers, and can be used by programs or end users to correlate the two.

Each line in **lexnames** contains 3 tab separated fields, and is terminated with a newline character. The first field is the two digit decimal integer file number. (The first file in the list is numbered **00** .) The second field is the name of the lexicographer file that is represented by that number, and the third field is an integer that indicates the syntactic category of the synsets contained in the file. This is simply a shortcut for programs and scripts, since the syntactic category is also part of the lexicographer file's name.

The syntactic category field is encoded as follows:

1 NOUN
2 VERB
3 ADJECTIVE
4 ADVERB

The names of the lexicographer files and their corresponding file numbers are listed below along with a brief description each file's contents.

File Number	Name	Contents
00	adj.all	all adjective clusters
01	adj.pert	relational adjectives (pertainyms)
02	adv.all	all adverbs
03	noun.Tops	unique beginner for nouns
04	noun.act	nouns denoting acts or actions
05	noun.animal	nouns denoting animals
06	noun.artifact	nouns denoting man-made objects

07	noun.attribute	nouns denoting attributes of people and objects
08	noun.body	nouns denoting body parts
09	noun.cognition	nouns denoting cognitive processes and contents
10	noun.communication	nouns denoting communicative processes and contents
11	noun.event	nouns denoting natural events
12	noun.feeling	nouns denoting feelings and emotions
13	noun.food	nouns denoting foods and drinks
14	noun.group	nouns denoting groupings of people or objects
15	noun.location	nouns denoting spatial position
16	noun.motive	nouns denoting goals
17	noun.object	nouns denoting natural objects (not man-made)
18	noun.person	nouns denoting people
19	noun.phenomenon	nouns denoting natural phenomena
20	noun.plant	nouns denoting plants
21	noun.possession	nouns denoting possession and transfer of possession
22	noun.process	nouns denoting natural processes
23	noun.quantity	nouns denoting quantities and units of measure
24	noun.relation	nouns denoting relations between people or things or ideas
25	noun.shape	nouns denoting two and three dimensional shapes
26	noun.state	nouns denoting stable states of affairs
27	noun.substance	nouns denoting substances
28	noun.time	nouns denoting time and temporal relations
29	verb.body	verbs of grooming, dressing and bodily care
30	verb.change	verbs of size, temperature change, intensifying, etc.
31	verb.cognition	verbs of thinking, judging, analyzing, doubting
32	verb.communication	verbs of telling, asking, ordering, singing
33	verb.competition	verbs of fighting, athletic activities
34	verb.consumption	verbs of eating and drinking
35	verb.contact	verbs of touching, hitting, tying, digging
36	verb.creation	verbs of sewing, baking, painting, performing
37	verb.emotion	verbs of feeling
38	verb.motion	verbs of walking, flying, swimming
39	verb.perception	verbs of seeing, hearing, feeling
40	verb.possession	verbs of buying, selling, owning

41	verb.social	verbs of political and social activities and events
42	verb.stative	verbs of being, having, spatial relations
43	verb.weather	verbs of raining, snowing, thawing, thundering
44	adj.ppl	participial adjectives

The lexicographer files are not included in the WordNet database package.

WordNet functions

Functions for searching the WordNet database

#include "wn.h"

char *findtheinfo(char *searchstr, int pos, int ptr_type, int sense_num);

SynsetPtr findtheinfo_ds(char *searchstr, int pos, int ptr_type, int sense_num);

unsigned int is_defined(char *searchstr, int pos);

unsigned int in_wn(char *searchstr, int pos);

IndexPtr index_lookup(char *searchstr, int pos);

IndexPtr parse_index(long offset, int dabase, char *line);

IndexPtr getindex(char *searchstr, int pos);

SynsetPtr read_synset(int pos, long synset_offset, char *searchstr);

SynsetPtr parse_synset(FILE *fp, int pos, char *searchstr);

void free_syns(SynsetPtr synptr);

void free_synset(SynsetPtr synptr);

void free_index(IndexPtr idx);

SynsetPtr traceptrs_ds(SynsetPtr synptr, int ptr_type, int pos, int depth);

char *do_trace(SynsetPtr synptr, int ptr_type, int pos, int depth);

These functions are used for searching the WordNet database. They generally fall into several categories: functions for reading and parsing index file entries; functions for reading and parsing

synsets in data files; functions for tracing pointers and hierarchies; functions for freeing space occupied by data structures allocated with **malloc**.

In the following function descriptions, *pos* is one of the following:

1 NOUN
2 VERB
3 ADJECTIVE
4 ADVERB

findtheinfo() is the primary search algorithm for use with database interface applications. Search results are automatically formatted, and a pointer to the text buffer is returned. All searches listed in **WNHOME/include/wn.h** can be done by **findtheinfo()**. **findtheinfo_ds()** can be used to perform most of the searches, with results returned in a linked list data structure. This is for use with applications that need to analyze the search results rather than just display them.

Both functions are passed the same arguments: *searchstr* is the word or collocation to search for; *pos* indicates the syntactic category to search in; *ptr_type* is one of the valid search types for *searchstr* in *pos* . (Available searches can be obtained by calling **is_defined()** described below.) *sense_num* should be **ALLSENSES** if the search is to be done on all senses of *searchstr* in *pos* , or a positive integer indicating which sense to search.

findtheinfo_ds() returns a linked list data structures representing synsets. Senses are linked through the *nextss* field of a **Synset** data structure. For each sense, synsets that match the search specified with *ptr_type* are linked through the *ptrlist* field. See **Synset Navigation** below, for detailed information on the linked lists returned.

is_defined() sets a bit for each search type that is valid for *searchstr* in *pos* , and returns the resulting unsigned integer. Each bit number corresponds to a pointer type constant defined in **WNHOME/include/wn.h** . For example, if bit 2 is set, the **HYPERPTR** search is valid for *searchstr* . There are 29 possible searches.

in_wn() is used to find the syntactic categories in the WordNet database that contain one or more senses of *searchstr* . If *pos* is **ALL_POS,** all syntactic categories are checked. Otherwise, only the part of speech passed is checked. An unsigned integer is returned with a bit set corresponding to each syntactic category containing *searchstr* . The bit number matches the number for the part of speech. **0** is returned if *searchstr* is not present in *pos* .

index_lookup() finds *searchstr* in the index file for *pos* and returns a pointer to the parsed entry in an **Index** data structure. *searchstr* must exactly match the form of the word (lower case only, hyphens and underscores in the same places) in the index file. **NULL** is returned if a match is not found.

parse_index() parses an entry from an index file and returns a pointer to the parsed entry in an **Index** data structure. Passed the byte *offset* and syntactic category, it reads the index entry at the desired location in the corresponding file. If passed *line* , *line* contains an index file entry and the database index file is not consulted. However, *offset* and *dbase* should still be passed so the information can be stored in the **Index** structure.

getindex() is a "smart" search for *searchstr* in the index file corresponding to *pos* . It applies to *searchstr* an algorithm that replaces underscores with hyphens, hyphens with underscores, removes hyphens and underscores, and removes periods in an attempt to find a form of the string that is an exact match for an entry in the index file corresponding to *pos* . **index_lookup()** is called on each transformed string until a match is found or all the different strings have been tried. It returns a pointer to the parsed **Index** data structure for *searchstr* , or **NULL** if a match is not found.

read_synset() is used to read a synset from a byte offset in a data file. It performs an **fseek** to *synset_offset* in the data file corresponding to *pos* , and calls **parse_synset()** to read and parse the synset. A pointer to the **Synset** data structure containing the parsed synset is returned.

parse_synset() reads the synset at the current offset in the file indicated by *fp* . *pos* is the syntactic category, and *searchstr* , if not **NULL,** indicates the word in the synset that the caller is interested in. An attempt is made to match *searchstr* to one of the words in the synset. If an exact match is found, the *whichword* field in the **Synset** structure is set to that word's number in the synset (beginning to count from **1**).

free_syns() is used to free a linked list of **Synset** structures allocated by **findtheinfo_ds()** . *synptr* is a pointer to the list to free.

free_synset() frees the **Synset** structure pointed to by *synptr* .

free_index() frees the **Index** structure pointed to by *idx* .

traceptrs_ds() is a recursive search algorithm that traces pointers matching *ptr_type* starting with the synset pointed to by *synptr* . Setting *depth* to **1** when **traceptrs_ds()** is called indicates a recursive search; **0** indicates a non-recursive call. *synptr* points to the data structure representing the synset to search for a pointer of type *ptr_type* . When a pointer type match is found, the synset pointed to is read is linked onto the *nextss* chain. Levels of the tree generated by a recursive search are linked via the *ptrlist* field structure until **NULL** is found, indicating the top (or bottom) of the tree. This function is usually called from **findtheinfo_ds()** for each sense of the word. See **Synset Navigation** below, for detailed information on the linked lists returned.

do_trace() performs the search indicated by *ptr_type* on synset synptr in syntactic category *pos* . *depth* is defined as above. **do_trace()** returns the search results formatted in a text buffer.

Synset Navigation

Since the **Synset** structure is used to represent the synsets for both word senses and pointers, the *ptrlist* and *nextss* fields have different meanings depending on whether the structure is a word sense or pointer. This can make navigation through the lists returned by **findtheinfo_ds()** confusing.

Navigation through the returned list involves the following:

Following the *nextss* chain from the synset returned moves through the various senses of *searchstr* . **NULL** indicates that end of the chain of senses.

Following the *ptrlist* chain from a **Synset** structure representing a sense traces the hierarchy of the search results for that sense. Subsequent links in the *ptrlist* chain indicate the next level (up or down, depending on the search) in the hierarchy. **NULL** indicates the end of the chain of search result synsets.

If a synset pointed to by *ptrlist* has a value in the *nextss* field, it represents another pointer of the same type at that level in the hierarchy. For example, some noun synsets have two hypernyms. Following this *nextss* pointer, and then the *ptrlist* chain from the **Synset** structure pointed to, traces another, parallel, hierarchy, until the end is indicated by **NULL** on that *ptrlist* chain. So, a **synset** representing a pointer (versus a sense of *searchstr*) having a non-NULL value in *nextss* has another chain of search results linked through the *ptrlist* chain of the synset pointed to by *nextss* .

If *searchstr* contains more than one base form in WordNet (as in the noun **axes** , which has base forms **axe** and **axis**), synsets representing the search results for each base form are linked through the *nextform* pointer of the **Synset** structure.

WordNet Searches

There is no extensive description of what each search type is or the results returned. Using the WordNet interface, examining the source code, and searching in this manual are the best ways to see what types of searches are available and the data returned for each.

Listed below are the valid searches that can be passed as *ptr_type* to **findtheinfo()** . Passing a negative value (when applicable) causes a recursive, hierarchical search by setting *depth* to **1** when **traceptrs()** is called.

ptr_type	Value	Pointer Symbol	Search
ANTPTR	1	!	Antonyms
HYPERPTR	2	@	Hypernyms
HYPOPTR	3		Hyponyms
ENTAILPTR	4	*	Entailment

SIMPTR	5	&	Similar
ISMEMBERPTR	6	#m	Member meronym
ISSTUFFPTR	7	#s	Substance meronym
ISPARTPTR	8	#p	Part meronym
HASMEMBERPTR	9	%m	Member holonym
HASSTUFFPTR	10	%s	Substance holonym
HASPARTPTR	11	%p	Part holonym
MERONYM	12	%	All meronyms
HOLONYM	13	#	All holonyms
CAUSETO	14	>	Cause
PPLPTR	15	<	Participle of verb
SEEALSOPTR	16	^	Also see
PERTPTR	17	\	Pertains to noun or derived from adjective
ATTRIBUTE	18	\=	Attribute
VERBGROUP	19	$	Verb group
DERIVATION	20	+	Derivationally related form
CLASSIFICATION	21	;	Domain of synset
CLASS	22	-	Member of this domain
SYNS	23	*n/a*	Find synonyms
FREQ	24	*n/a*	Polysemy
FRAMES	25	*n/a*	Verb example sentences and generic frames
COORDS	26	*n/a*	Noun coordinates
RELATIVES	27	*n/a*	Group related senses
HMERONYM	28	*n/a*	Hierarchical meronym search
HHOLONYM	29	*n/a*	Hierarchical holonym search
WNGREP	30	*n/a*	Find keywords by substring
OVERVIEW	31	*n/a*	Show all synsets for word
CLASSIF_CATEGORY	32	;c	Show domain topic
CLASSIF_USAGE	33	;u	Show domain usage
CLASSIF_REGIONAL	34	;r	Show domain region
CLASS_CATEGORY	35	-c	Show domain terms for topic
CLASS_USAGE	36	-u	Show domain terms for usage
CLASS_REGIONAL	37	-r	Show domain terms for region
INSTANCE	38	@i	Instance of
INSTANCES	39	i	Show instances

findtheinfo_ds() cannot perform the following searches:

SEEALSOPTR
PERTPTR
VERBGROUP
FREQ
FRAMES
RELATIVES
WNGREP
OVERVIEW

NOTES

Applications that use WordNet and/or the morphological functions must call **wninit()** at the start of the program.

In all function calls, *searchstr* may be either a word or a collocation formed by joining individual words with underscore characters (_).

The **SearchResults** structure defines fields in the *wnresults* global variable that are set by the various search functions. This is a way to get additional information, such as the number of senses the word has, from the search functions. The *searchds* field is set by **findtheinfo_ds()** .

The *pos* passed to **traceptrs_ds()** is not used.

WARNINGS

parse_synset() must find an exact match between the *searchstr* passed and a word in the synset to set *whichword* . No attempt is made to translate hyphens and underscores, as is done in **getindex()** .

The WordNet database and exception list files must be opened with **wninit** prior to using any of the searching functions.

A large search may cause **findtheinfo()** to run out of buffer space. The maximum buffer size is determined by computer platform. If the buffer size is exceeded the following message is printed in the output buffer: **"Search too large. Narrow search and try again..."** .

Passing an invalid *pos* will probably result in a core dump.

WordNet Documentation

WordNet morphological processor, Morphy

morphinit, re_morphinit, morphstr, morphword

#include "wn.h"

int morphinit(void);

int re_morphinit(void);

char *morphstr(char *origstr, int pos);

char *morphword(char *word, int pos);

The WordNet morphological processor, Morphy, is accessed through these functions:

morphinit() is used to open the exception list files. It returns **0** if successful, **-1** otherwise. The exception list files must be opened before **morphstr()** or **morphword()** are called.

re_morphinit() is used to close the exception list files and reopen them, and is used exclusively for WordNet development. Return codes are as described above.

morphstr() is the basic user interface to Morphy. It tries to find the base form (lemma) of the word or collocation *origstr* in the specified *pos* . The first call (with *origstr* specified) returns a pointer to the first base form found. Subsequent calls requesting base forms of the same string must be made with the first argument of **NULL.** When no more base forms for *origstr* can be found, **NULL** is returned. Note that **morphstr()** returns a pointer to a static character buffer. A subsequent call to **morphstr()** with a new string (instead of **NULL**) will overwrite the string pointed to by a previous call. Users should copy the returned string into a local buffer, or use the C library function **strdup** to duplicate the returned string into a *malloc'd* buffer.

morphword() tries to find the base form of *word* in the specified *pos* . This function is called by **morphstr()** for each individual word in a collocation. Note that **morphword()** returns a pointer to a static character buffer. A subsequent call to **morphword()** will overwrite the string pointed to by a previous call. Users should copy the ret**morphinit()** is called by **wninit()** and is not intended to be called directly by an application. Applications wishing to use WordNet and/or the morphological functions must call **wninit()** at the start of the program.

origstr may be either a word or a collocation formed by joining individual words with underscore characters (_).

Usually only **morphstr()** is called from applications, as it works on both words and collocations.

pos must be one of the following:

1 NOUN
2 VERB
3 ADJECTIVE
4 ADVERB
5 ADJECTIVE_SATELLITE

If **ADJECTIVE_SATELLITE** is passed, it is treated by **morphstr()** as **ADJECTIVE.**

urned string into a local buffer, or use the C library function **strdup** to duplicate the returned string into a *malloc'd* buffer.

Passing an invalid part of speech will result in a core dump.

The WordNet database files must be open to use **morphstr()** or **morphword().**

Morphy will allow non-words to be converted to words, if they follow one of the rules described above. For example, it will happily convert **plantes** to **plants** .

wngroups - search code to group similar verb senses

Some similar senses of verbs have been grouped by the lexicographers. This grouping is done statically in the lexicographer source files using the semantic *pointer_symbol* **$** . Transitivity is used to combine groups of overlapping senses into the largest sense groups possible.

Coverage of verb groups is incomplete.

sentidx.vrb
verb sense keys and sentence frame numbers
sents.vrb
example sentence frames

WordNet processing context

morphy - discussion of WordNet's morphological processing

Although only base forms of words are usually stored in WordNet, searches may be done on inflected forms. A set of morphology functions, Morphy, is applied to the search string to generate a form that is present in WordNet.

WordNet Documentation

Morphology in WordNet uses two types of processes to try to convert the string passed into one that can be found in the WordNet database. There are lists of inflectional endings, based on syntactic category, that can be detached from individual words in an attempt to find a form of the word that is in WordNet. There are also exception list files, one for each syntactic category, in which a search for an inflected form is done. Morphy tries to use these two processes in an intelligent manner to translate the string passed to the base form found in WordNet. Morphy first checks for exceptions, then uses the rules of detachment. The Morphy functions are not independent from WordNet. After each transformation, WordNet is searched for the resulting string in the syntactic category specified.

The Morphy functions are passed a string and a syntactic category. A string is either a single word or a collocation. Since some words, such as **axes** can have more than one base form (**axe** and **axis**), Morphy works in the following manner. The first time that Morphy is called with a specific string, it returns a base form. For each subsequent call to Morphy made with a **NULL** string argument, Morphy returns another base form. Whenever Morphy cannot perform a transformation, whether on the first call for a word or subsequent calls, **NULL** is returned. A transformation to a valid English string will return **NULL** if the base form of the string is not in WordNet.

The morphological functions are found in the WordNet library.

The following table shows the rules of detachment used by Morphy. If a word ends with one of the suffixes, it is stripped from the word and the corresponding ending is added. Then WordNet is searched for the resulting string. No rules are applicable to adverbs.

POS	Suffix	Ending
NOUN	"s"	""
NOUN	"ses"	"s"
NOUN	"xes"	"x"
NOUN	"zes"	"z"
NOUN	"ches"	"ch"
NOUN	"shes"	"sh"
NOUN	"men"	"man"
NOUN	"ies"	"y"
VERB	"s"	""
VERB	"ies"	"y"
VERB	"es"	"e"
VERB	"es"	""
VERB	"ed"	"e"
VERB	"ed"	""
VERB	"ing"	"e"

VERB	"ing"	""
ADJ	"er"	""
ADJ	"est"	""
ADJ	"er"	"e"
ADJ	"est"	"e"

There is one exception list file for each syntactic category. The exception lists contain the morphological transformations for strings that are not regular and therefore cannot be processed in an algorithmic manner. Each line of an exception list contains an inflected form of a word or collocation, followed by one or more base forms. The list is kept in alphabetical order and a binary search is used to find words in these lists.

In general, single words are relatively easy to process. Morphy first looks for the word in the exception list. If it is found the first base form is returned. Subsequent calls with a **NULL** argument return additional base forms, if present. A **NULL** is returned when there are no more base forms of the word. If the word is not found in the exception list corresponding to the syntactic category, an algorithmic process using the rules of detachment looks for a matching suffix. If a matching suffix is found, a corresponding ending is applied (sometimes this ending is a **NULL** string, so in effect the suffix is removed from the word), and WordNet is consulted to see if the resulting word is found in the desired part of speech.

As opposed to single words, collocations can be quite difficult to transform into a base form that is present in WordNet. In general, only base forms of words, even those comprising collocations, are stored in WordNet, such as **attorney general** . Transforming the collocation **attorneys general** is then simply a matter of finding the base forms of the individual words comprising the collocation. This usually works for nouns, therefore non-conforming nouns, such as **customs duty** are presently entered in the noun exception list.

Verb collocations that contain prepositions, such as **ask for it** , are more difficult. As with single words, the exception list is searched first. If the collocation is not found, special code in Morphy determines whether a verb collocation includes a preposition. If it does, a function is called to try to find the base form in the following manner. It is assumed that the first word in the collocation is a verb and that the last word is a noun. The algorithm then builds a search string with the base forms of the verb and noun, leaving the remainder of the collocation (usually just the preposition, but more words may be involved) in the middle. For example, passed **asking for it** , the database search would be performed with **ask for it** , which is found in WordNet, and therefore returned from Morphy. If a verb collocation does not contain a preposition, then the base form of each word in the collocation is found and WordNet is searched for the resulting string.

Hyphenation also presents special difficulties when searching WordNet. It is often a subjective decision as to whether a word is hyphenated, joined as one word, or is a collocation of several words, and which of the various forms are entered into WordNet. When Morphy breaks a string into "words", it looks for both spaces and hyphens as delimiters. It also looks for periods in strings and removes them if an exact match is not found. A search for an abbreviation like **oct.** return the synset for **{ October, Oct }** . Not every pattern of hyphenated and collocated string is searched for properly, so it may be advantageous to specify several search strings if the results of a search attempt seem incomplete.

Morphy contains code that searches for nouns ending with **ful** and performs a transformation on the substring preceeding it. It then appends 'ful' back onto the resulting string and returns it. For example, if passed the nouns **boxesful** , it will return **boxful** .

ince many noun collocations contains prepositions, such as **line of products** , an algorithm similar to that used for verbs should be written for nouns. In the present scheme, if Morphy is passed **lines of products** , the search string becomes **line of product** , which is not in WordNet
Morphy will allow non-words to be converted to words, if they follow one of the rules described above. For example, it will happily convert **plantes** to **plants** .

***pos* .exc**
morphology exception lists

WordNet's sense index

index.sense, sense.idx - WordNet's sense index

The WordNet sense index provides an alternate method for accessing synsets and word senses in the WordNet database. It is useful to applications that retrieve synsets or other information related to a specific sense in WordNet, rather than all the senses of a word or collocation. It can also be used with tools like **grep** and Perl to find all senses of a word in one or more parts of speech. A specific WordNet sense, encoded as a *sense_key* , can be used as an index into this file to obtain its WordNet sense number, the database byte offset of the synset containing the sense, and the number of times it has been tagged in the semantic concordance texts.
Concatenating the *lemma* and *lex_sense* fields of a semantically tagged word (represented in a **<wf** ... **>** attribute/value pair) in a semantic concordance file, using **%** as the concatenation character, creates the *sense_key* for that sense, which can in turn be used to search the sense index file.

A *sense_key* is the best way to represent a sense in semantic tagging or other systems that refer to WordNet senses. *sense_key* s are independent of WordNet sense numbers and *synset_offset* s, which vary between versions of the database. Using the sense index and a *sense_key* , the corresponding synset (via the *synset_offset*) and WordNet sense number can easily be obtained. A mapping from noun *sense_key* s in WordNet 1.6 to corresponding 2.0 *sense_key* s is provided with version 2.0.

The sense index file lists all of the senses in the WordNet database with each line representing one sense. The file is in alphabetical order, fields are separated by one space, and each line is terminated with a newline character.

Each line is of the form:

sense_key synset_offset sense_number tag_cnt

sense_key is an encoding of the word sense. Programs can construct a sense key in this format and use it as a binary search key into the sense index file. The format of a *sense_key* is described below.

synset_offset is the byte offset that the synset containing the sense is found at in the database "data" file corresponding to the part of speech encoded in the *sense_key* . *synset_offset* is an 8 digit, zero-filled decimal integer, and can be used with **fseek** to read a synset from the data file. When passed to the WordNet library function **read_synset()** along with the syntactic category, a data structure containing the parsed synset is returned.

sense_number is a decimal integer indicating the sense number of the word, within the part of speech encoded in *sense_key* , in the WordNet database.

tag_cnt represents the decimal number of times the sense is tagged in various semantic concordance texts. A *tag_cnt* of **0** indicates that the sense has not been semantically tagged.

A *sense_key* is represented as:

lemma **%** *lex_sense*

where *lex_sense* is encoded as:

*ss_type****:lex_filenum:lex_id:head_word:head_id***

lemma is the ASCII text of the word or collocation as found in the WordNet database index file corresponding to *pos* . *lemma* is in lower case, and collocations are formed by joining individual words with an underscore (**_**) character.

ss_type is a one digit decimal integer representing the synset type for the sense. See **Synset Type** below for a listing of the numbers corresponding to each synset type.

lex_filenum is a two digit decimal integer representing the name of the lexicographer file containing the synset for the sense.

lex_id is a two digit decimal integer that, when appended onto *lemma* , uniquely identifies a sense within a lexicographer file. *lex_id* numbers usually start with **00** , and are incremented as additional

senses of the word are added to the same file, although there is no requirement that the numbers be consecutive or begin with **00** . Note that a value of **00** is the default, and therefore is not present in lexicographer files. Only non-default *lex_id* values must be explicitly assigned in lexicographer files.

head_word is only present if the sense is in an adjective satellite synset. It is the lemma of the first word of the satellite's head synset.

head_id is a two digit decimal integer that, when appended onto *head_word* , uniquely identifies the sense of *head_word* within a lexicographer file, as described for *lex_id* . There is a value in this field only if *head_word* is present.

The synset type is encoded as follows:

1 NOUN
2 VERB
3 ADJECTIVE
4 ADVERB
5 ADJECTIVE SATELLITE

index.sense
sense index

uniqbeg - unique beginners for noun hierarchies

All of the WordNet noun synsets are organized into hierarchies, headed by the unique beginner synset for **entity** in the file **noun.Tops** .

{ entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving)) }

noun.Tops
unique beginners for nouns

Glossary

Many terms used in the *WordNet Reference Manual* are unique to the WordNet system. Other general terms have specific meanings when used in the WordNet documentation. Definitions for many of these terms are given to help with the interpretation and understanding of the reference manual, and in the use of the WordNet system.
In following definitions **word** is used in place of **word or collocation** .

adjective cluster

A group of adjective synsets that are organized around antonymous pairs or triplets. An adjective cluster contains two or more **head synsets** which represent antonymous concepts. Each head synset has one or more **satellite synsets** .

attribute

A noun for which adjectives express values. The noun **weight** is an attribute, for which the adjectives **light** and **heavy** express values.

base form

The base form of a word or collocation is the form to which inflections are added.

basic synset

Syntactically, same as **synset** .

collocation

A collocation in WordNet is a string of two or more words, connected by spaces or hyphens. Examples are: **man-eating shark** , **blue-collar** , **depend on** , **line of products** . In the database files spaces are represented as underscore (_) characters.

coordinate

Coordinate terms are nouns or verbs that have the same **hypernym** .

cross-cluster pointer

A **semantic pointer** from one adjective cluster to another.

derivationally related forms

Terms in different syntactic categories that have the same root form and are semantically related.

direct antonyms

A pair of words between which there is an associative bond resulting from their frequent co-occurrence. In **adjective clusters** , direct antonyms appears only in **head synsets** .

domain

A topical classification to which a synset has been linked with a CATEGORY, REGION or USAGE pointer.

domain term

A synset belonging to a topical class. A domain term is further identified as being a CATEGORY_TERM, REGION_TERM or USAGE_TERM.

entailment

A verb **X** entails **Y** if **X** cannot be done unless **Y** is, or has been, done.

exception list

Morphological transformations for words that are not regular and therefore cannot be processed in an algorithmic manner.

group

Verb senses that similar in meaning and have been manually grouped together.

gloss

Each synset contains **gloss** consisting of a definition and optionally example sentences.

head synset

Synset in an adjective **cluster** containing at least one word that has a **direct antonym** .

holonym

The name of the whole of which the meronym names a part. **Y** is a holonym of **X** if **X** is a part of **Y** .

hypernym

The generic term used to designate a whole class of specific instances. **Y** is a hypernym of **X** if **X** is a (kind of) **Y** .

hyponym

The specific term used to designate a member of a class. **X** is a hyponym of **Y** if **X** is a (kind of) **Y** .

indirect antonym

An adjective in a **satellite synset** that does not have a **direct antonym** has an indirect antonyms via the direct antonym of the **head synset** .

instance

A proper noun that refers to a particular, unique referent (as distinguished from nouns that refer to classes). This is a specific form of hyponym.

lemma

Lower case ASCII text of word as found in the WordNet database index files. Usually the **base form** for a word or collocation.

lexical pointer

A lexical pointer indicates a relation between words in synsets (word forms).

lexicographer file

Files containing the raw data for WordNet synsets, edited by lexicographers, that are input to the **grind** program to generate a WordNet database.

lexicographer id (lex id)

A decimal integer that, when appended onto **lemma** , uniquely identifies a sense within a lexicographer file.

monosemous

Having only one sense in a syntactic category.

meronym

The name of a constituent part of, the substance of, or a member of something. **X** is a meronym of **Y** if **X** is a part of **Y** .

part of speech

WordNet defines "part of speech" as either noun, verb, adjective, or adverb. Same as **syntactic category** .

participial adjective

An adjective that is derived from a verb.

pertainym

A relational adjective. Adjectives that are pertainyms are usually defined by such phrases as "of or pertaining to" and do not have antonyms. A pertainym can point to a noun or another pertainym.

polysemous

Having more than one sense in a syntactic category.

polysemy count

Number of senses of a word in a syntactic category, in WordNet.

postnominal

A postnominal adjective occurs only immediately following the noun that it modifies.

predicative

An adjective that can be used only in predicate positions. If **X** is a predicate adjective, it can only be used in such phrases as "it is **X** " and never prenominally.

prenominal

An adjective that can occur only before the noun that it modifies: it cannot be used predicatively.

satellite synset

Synset in an adjective **cluster** representing a concept that is similar in meaning to the concept represented by its **head synset** .

semantic concordance

A textual corpus (e.g. the Brown Corpus) and a lexicon (e.g. WordNet) so combined that every substantive word in the text is linked to its appropriate sense in the lexicon via a **semantic tag** .

semantic tag

A pointer from a word in a text file to a specific sense of that word in the WordNet database. A semantic tag in a semantic concordance is represented by a **sense key** .

semantic pointer

A semantic pointer indicates a relation between synsets (concepts).

sense

A meaning of a word in WordNet. Each sense of a word is in a different **synset** .

sense key

Information necessary to find a sense in the WordNet database. A sense key combines a **lemma** field and codes for the synset type, lexicographer id, lexicographer file number, and information about a satellite's **head synset** , if required.

subordinate

Same as **hyponym** .

superordinate

Same as **hypernym** .

synset

A synonym set; a set of words that are interchangeable in some context without changing the truth value of the preposition in which they are embedded.

troponym

A verb expressing a specific manner elaboration of another verb. **X** is a troponym of **Y** if **to X** is **to Y** in some manner.

unique beginner

A noun synset with no **superordinate** .

WordNet Documentation

WordNet license