NAME

findtheinfo, findtheinfo_ds, is_defined, in_wn, index_lookup, parse_index, getindex, read_synset, parse_synset, free_syns, free_synset, free_index, traceptrs_ds, do_trace

SYNOPSIS

#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);

DESCRIPTION

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**(3).

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**(3) 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 reading **wndb**(5WN) 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	ˆ	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. See **wnutil**(3WN) for more information.

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.

SEE ALSO

wn(1WN), **wnb**(1WN), **wnintro**(3WN), **binsrch**(3WN), **malloc**(3), **morph**(3WN), **wnutil**(3WN), **wnintro**(5WN).

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.