NAME

search − SWISH++ searcher

SYNOPSIS

search [*options*] *query*

DESCRIPTION

search is the SWISH++ searcher. It searches a previously generated index for the words specified in a query. In addition to running from the command-line, it can run as a daemon process functioning as a ‘‘search server.’’

QUERY INPUT

Query Syntax

The formal grammar of a query is:

query:	*query optional_relop meta* *meta*
meta:	*meta_name* `=` *primary* *primary*
meta_name:	*word*
primary:	`(`*query*`)` `not` *meta* *word* *word*`*`

optional_relop:	`and`
	`near`
	`not near`
	`or`
	(empty)

In practice, however, the query is the set of words sought after, possibly restricted to meta data, and possibly combined with the operators ‘‘`and`,’’ ‘‘`or`,’’ ‘‘`near`,’’ ‘‘`not`,’’ and ‘‘`not near`.’’ The asterisk (`*`) can be used as a wildcard character at the end of words. Note that an asterisk is a shell meta-character and as such must be escaped (backslashed) when passed to a shell.

Although syntactically legal, it is a semantic error to have ‘‘`near`’’ just before ‘‘`not`’’ since such queries are nonsensical, e.g.:

```
mouse near not computer
```

Queries are evaluated in left-to-right order, i.e., ‘‘`and`’’ has the same precedence as ‘‘`or`.’’ For more about query syntax, see the EXAMPLES.

Character Mapping and Word Determination

The same character mapping and word determination heuristics used by **index**(1) are used on queries prior to searching.

RESULTS OUTPUT

Result Components

The results are output either in ‘‘classic’’ or XML format. In either case, the components of the results are:

rank	An integer from from 1 to 100.
path-name	The relative path to where the file was originally indexed.
file-size	The file’s size in bytes.
file-title	If the file is of a format that can have titles (HTML, XHTML, LaTeX, mail, or Unix manual pages) and the title was extracted, then *file-title* is its title; otherwise, it is its filename.

Classic Results Format

The "classic" results format is plain text as:

rank path-name file-size file-title

It can be parsed easily in Perl with:

```
($rank,$path,$size,$title) = split( / /, $_, 4 );
```

(The separator can be changed via the **−R** or **−−separator** options or the **ResultSeparator** variable.)

Prior to results lines, comment lines may also appear containing additional information about the query results. Comment lines are in the format of:

`#` *comment-key*`:` *comment-value*

The keys and values are:

ignored: *stop-words*	The list of stop-words (separated by spaces) ignored in the query.
not found: *word*	The word was not found in the index.
results: *result-count*	The total number of results.

XML Results Format

The XML results format is given by the DTD:

```
<!ELEMENT SearchResults (IgnoredList?, ResultCount, ResultList?)>
<!ELEMENT IgnoredList (Ignored+)>
<!ELEMENT Ignored (#PCDATA)>
<!ELEMENT ResultCount (#PCDATA)>
<!ELEMENT ResultList (File+)>
<!ELEMENT File (Rank, Path, Size, Title)>
<!ELEMENT Rank (#PCDATA)>
<!ELEMENT Path (#PCDATA)>
<!ELEMENT Size (#PCDATA)>
```

and by the XML schema located at:

```
http://homepage.mac.com/pauljlucas/software/swish/SearchResults/SearchResults.xs
```

For example:

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE SearchResults SYSTEM
 "http://homepage.mac.com/pauljlucas/software/swish/SearchResults.dtd">
<SearchResults
 xmlns="http://homepage.mac.com/pauljlucas/software/swish/SearchResults"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://homepage.mac.com/pauljlucas/software/swish/SearchRes
                     SearchResults.xsd">
  <IgnoredList>
    <Ignored>stop-word</Ignored>
  ...
  </IgnoredList>
  <ResultCount>42</ResultCount>
  <ResultList>
```

```
    <File>
      <Rank>rank</Rank>
      <Path>path-name</Path>
      <Size>file-size</Size>
      <Title>file-title</Title>
    </File>
  ...
   </ResultList>
</SearchResults>
```

RUNNING AS A DAEMON PROCESS

Description

search can alternatively run as a daemon process (via either the **−b** or **−−daemon-type** options or the **SearchDaemon** variable) functioning as a "search server" by listening to a Unix domain socket (specified by either the **−u** or **−−socket-file** options or the **SocketFile** variable), a TCP socket (specified by either the **−a** or **−−socket-address** options or the **SocketAddress** variable), or both. Unix domain sockets are preferred for both performance and security. For search-intensive applications, such as a search engine on a heavily used web site, this can yield a large performance improvement since the start-up cost (**fork**(2), **exec**(2), and initialization) is paid only once.

If the process was started with root privileges, it will give them away immediately after initialization and before servicing any requests.

Clients and Requests

Search clients connect to a daemon via a socket and send a query in the same manner as on the command line (including the first word being "`search`"). The only exception is that shell meta-characters *must not* be escaped (backslashed) since no shell is involved. Search results are returned via the same socket. See the EXAMPLES.

Multithreading

A daemon can serve multiple query requests simultaneously since it is multi-threaded. When started, it "pre-threads" meaning that it creates a pool of threads in advance that service an indefinite number of requests as a further performance improvement since a thread is not created and destroyed per request.

There is an initial, minimum number of threads in the thread pool. The number of threads grows dynamically when there are more requests than threads, but not more than a specified maximum to prevent the server from thrashing. (See the **−t**, **−−min-threads**, **−T**, and **−−max-threads** options or the **ThreadsMin** or **ThreadsMax** variables.) If the number of threads reaches the maximum, subsequent requests are queued until existing threads become available to service them after completing in-progress requests. (See either the **−q** or **−−queue-size** options or the **SocketQueueSize** variable.)

If there are more than the minimum number of threads and some remain idle longer than a specified timeout period (because the number of requests per unit time has dropped), then threads will die off until the pool returns to its original minimum size. (See either the **−O** or **−−thread-timeout** options or the **ThreadTimeout** variable.)

Restrictions

A single daemon can search only a single index. To search multiple indices concurrently, multiple daemons can be run, each searching its own index and using its own socket. An index *must not* be modified or deleted while a daemon is using it.

OPTIONS

Options begin with either a '−' for short options or a "−−" for long options. Either a '−' or "−−" by itself explicitly ends the options; however, the difference is that '−' is returned as the first non-option whereas "−−" is skipped entirely. Either short or long options may be used. Long option names may be abbreviated so long as the abbreviation is unambiguous.

For a short option that takes an argument, the argument is either taken to be the remaining characters of the same option, if any, or, if not, is taken from the next option unless said option begins with a '−'.

Short options that take no arguments can be grouped (but the last option in the group can take an argument), e.g., `-lrv4` is equivalent to `-l -r -v4`.

For a long option that takes an argument, the argument is either taken to be the characters after a '=', if any, or, if not, is taken from the next option unless said option begins with a '−'.

−?
−−help Print the usage (''help'') message and exit.

−a*a*
−−socket-address=*a* When running as a daemon, the address, *a*, to listen to for TCP requests. (Default is all IP addresses and port 1967.) The address argument is of the form:

[*host* **:** **]** *port*

that is: an optional host and colon followed by a port number. The *host* may be one of a host name, an IP address, or the `*` character meaning ''any IP address.'' Omitting the *host* and colon also means ''any IP address.''

−b*t*
−−daemon-type=*t* Run as a daemon process. (Default is not to.) The type, *t*, is one of:

`none` Same as not specifying the option at all. (This does not purport to be useful, but rather consistent with the types that can be specified to the **SearchDaemon** variable.)

`tcp` Listen on a TCP socket (see the **−a** option).

`unix` Listen on a Unix domain socket (see the **−u** option).

`both` Listen on both.

By default, if executed from the command-line, **search** appears to return immediately; however, it has merely detached from the terminal and put itself into the background. There is no need to follow the command with an '`&`'.

−B
−−no-background When running as a daemon process, do not detach from the terminal and run in the background. (Default does.)

The reason not to run in the background is so a wrapper script can see if the process dies for any reason and automatically restart it.

−c*f*
−−config-file=*f* The name of the configuration file, *f*, to use. (Default is `swish++.conf` in the current directory.) A configuration file is not required: if none is specified and the default does not exist, none is used; however, if one is specified and it does not exist, then this is an error.

−d
−−dump-words Dump the query word indices to standard output and exit. Wildcards are not permitted.

−D
−−dump-index Dump the entire word index to standard output and exit.

−F*f*
−−format=*f* The format, *f*, search results are output in. The format is either `classic` or `XML`. (Default is `classic`.)

−G*s*
−−group=*s* The group, *s*, to switch the process to after starting and only if started as root. (Default is `nobody`.)

−i*f*
−−index-file=*f* The name of the index file, *f*, to use. (Default is `swish++.index` in the current directory.)

−m*n*
−−max-results=*n* The maximum number of results, *n*, to return. (Default is 100.)

−M
−−dump-meta Dump the meta-name index to standard output and exit.

−n*n*
−−near=*n* The maximum number of words apart, *n*, two words can be to be considered ''near'' each other in queries using `near`. (Default is 10.)

−o*s*
−−socket-timeout=*s* The number of seconds, *s*, a search client has to complete a query request before the socket connection is closed. (Default is 10.) This is to prevent a client from connecting, not completing a request, and causing the thread servicing the request to wait forever.

−O*s*
−−thread-timeout=*s* The number of seconds, *s*, until an idle spare thread dies while running as a daemon. (Default is 30.)

−p*n*
−−word-percent=*n* The maximum percentage, *n*, of files a word may occur in before it is discarded as being too frequent. (Default is 100.) If you want to keep all words regardless, specify 101.

−P*f*
−−pid-file=*f* The name of the file to record the process ID of **search** if running as a daemon. (Default is none.)

−q*n*
−−queue-size=*n* The maximum number of socket connections to queue. (Default is 511.)

−r*n*
−−skip-results=*n* The initial number of results, *n*, to skip. (Default is 0.) Used in conjunction with **−m** or **−−max-results**, results can be returned in ''pages.''

−R*s*
−−separator=*s* The classic result separator string. (Default is " ".)

−s
−−stem-words Perform stemming (suffix stripping) on words during the search. Words that end in the wildcard character are not stemmed. (Default is no.)

−S
−−dump-stop Dump the stop-word index to standard output and exit.

−t*n*
−−min-threads=*n* Minimum number of threads to maintain while running as a daemon.

−T*n*
−−max-threads=*n* Maximum number of threads to allow while running as a daemon.

−u*f*
−−socket-file=*f* The name of the Unix domain socket file to use while running as a daemon. (Default is `/tmp/search.socket`.)

−U*s*
−−user=*s* The user, *s*, to switch the process to after starting and only if started as root. (Default is `nobody`.)

−V
−−version Print the version number of **SWISH++** to standard output and exit.

−w*n*[**,***c*]
−−window=*n*[**,***c*] Dump a ''window'' of at most *n* lines around each query word matching *c* characters. Wildcards are not permitted. (Default for *c* is 0.) Every window ends with a blank line.

CONFIGURATION FILE

The following variables can be set in a configuration file. Variables and command-line options can be mixed, the latter taking priority.

Group	Same as **−G** or **−−group**
IndexFile	Same as **−i** or **−−index-file**
PidFile	Same as **−P** or **−−pid-file**
ResultSeparator	Same as **−R** or **−−separator**
ResultsFormat	Same as **−F** or **−−format**
ResultsMax	Same as **−m** or **−−max-results**
SearchBackground	Same as **−B** or **−−no-background**
SearchDaemon	Same as **−b** or **−−daemon-type**
SocketAddress	Same as **−a** or **−−socket-address**
SocketFile	Same as **−u** or **−−socket-file**
SocketQueueSize	Same as **−q** or **−−queue-size**
SocketTimeout	Same as **−o** or **−−socket-timeout**
StemWords	Same as **−s** or **−−stem-words**
ThreadsMax	Same as **−T** or **−−max-threads**
ThreadsMin	Same as **−t** or **−−min-threads**
ThreadTimeout	Same as **−O** or **−−thread-timeout**
User	Same as **−U** or **−−user**
WordFilesMax	Same as **−f** or **−−word-files**
WordPercentMax	Same as **−p** or **−−word-percent**
WordsNear	Same as **−n** or **−−near**

EXAMPLES

Simple Queries

The query:

```
computer mouse
```

is the same as and short for:

```
computer and mouse
```

(because ''`and`'' is implicit) and would return only those documents that contain both words. The query:

```
cat or kitten or feline
```

would return only those documents regarding cats. The query:

```
mouse and computer or keyboard
```

is the same as:

```
(mouse and computer) or keyboard
```

(because queries are evaluated left-to-right) in that they will both return only those documents regarding either mice attached to a computer or any kind of keyboard. However, neither of those is the same as:

```
mouse and (computer or keyboard)
```

that would return only those documents regarding mice (including the rodents) and either a computer or a keyboard.

Queries Using Wildcards

The query:

```
comput*
```

would return only those documents that contain words beginning with ‘‘comput’’ such as ‘‘computation,’’ ‘‘computational,’’ ‘‘computer,’’ ‘‘computerize,’’ ‘‘computing,’’ and others. Wildcarded words can be used anywhere ordinary words can be. The query:

```
comput* (medicine or doctor*)
```

would return only those documents that contain something about computer use in medicine or by doctors.

Queries Using “not”

The query:

```
mouse or mice and not computer*
```

would return only those documents regarding mice (the rodents) and not the kind attached to a computer.

Queries Using “near”

Using ‘‘`near`’’ is the same as using ‘‘`and`’’ except that it not only requires both words to be in the documents, but that they be *near* each other, i.e., it returns potentially fewer documents than the corresponding ‘‘`and`’’ query. The query:

```
computer near mouse
```

would return only those documents where both words are near each other. They query:

```
mouse near (computer or keyboard)
```

is the same as:

```
(mouse near computer) or (mouse near keyboard)
```

i.e., ‘‘near’’ gets *distributed* across parenthesized subqueries.

Queries Using “not near”

Using ‘‘`not near`’’ is the same as using ‘‘`and not`’’ except that it allows the right-hand side words to be in the documents, just *not near* the left-hand side words, i.e., it returns potentially more documents than the corresponding ‘‘`and not`’’ query. Of course the word(s) on the right-hand side need not be in the documents at all, i.e., they would be considered ‘‘infinitely far’’ apart. The query:

```
mouse or mice not near computer*
```

would return only those documents regarding mice (the rodents) more effectively than the query:

```
mouse or mice and not computer*
```

because the latter would exclude documents about mice (the rodents) where computers just so happened to be mentioned in the same documents.

Queries Using Meta Data

The query:

```
author = hawking
```

would return only those documents whose author attribute contains ‘‘hawking.’’ The query:

```
author = hawking radiation
```

would return only those documents regarding radiation whose author attribute contains ‘‘hawking.’’ The query:

```
author = (stephen hawking)
```

would return only those documents whose author is Stephen Hawking. The query:

```
author = (stephen hawking) or (black near hole*)
```

would return only those documents whose author is Stephen Hawking or that contain the word ‘‘black’’ near ‘‘hole’’ or ‘‘holes’’ regardless of the author. Note that the second set of parentheses are necessary otherwise the query would have been the same as:

```
(author = (stephen hawking) or black) near hole*
```

that would have additionally required both ‘‘stephen’’ and ‘‘hawking’’ to be near ‘‘hole’’ or ‘‘holes.’’

Sending Queries to a Search Daemon

To send a query request to a search daemon using Perl, first open the socket and connect to the daemon (see [Wall], pp. 439-440):

```
use Socket;

$SocketFile = '/tmp/search.socket';
socket( SEARCH, PF_UNIX, SOCK_STREAM, 0 ) or
      die "can not open socket: $!\n";
connect( SEARCH, sockaddr_un( $SocketFile ) ) or
      die "can not connect to \"$SocketFile\": $!\n";
```

Autoflush *must* be set for the socket filehandle (see [Wall], p. 781), otherwise the server thread will hang since I/O buffering will wait for the buffer to fill that will never happen since queries are short:

```
select( (select( SEARCH ), $| = 1)[0] );
```

Next, send a query request (beginning with the word ‘‘search’’ and any options just as with a command-line) to the daemon via the socket filehandle making sure to include a trailing newline since the server reads an entire line of input (so therefore it looks and waits for a newline):

```
$query = 'mouse and computer';
print SEARCH "search $query\n";
```

Finally, read the results back and print them:

```
print while <SEARCH>;
close( SEARCH );
```

EXIT STATUS

Exits with one of the values given below:

0	Success.
1	Error in configuration file.
2	Error in command-line options.
40	Unable to read index file.
50	Malformed query.
51	Attempted "near" search without word-position data.
60	Could not write to PID file.
61	Host or IP address is invalid or nonexistent.
62	Could not open a TCP socket.
63	Could not open a Unix domain socket.
64	Could not **unlink**(2) a Unix domain socket file.
65	Could not **bind**(3) to a TCP socket.
66	Could not **bind**(3) to a Unix domain socket.
67	Could not **listen**(3) to a TCP socket.
68	Could not **listen**(3) to a Unix domain socket.
69	Could not **select**(3).
70	Could not **accept**(3) a socket connection.
71	Could not **fork**(2) child process.
72	Could not change directory to `/`.
73	Could not create thread.
74	Could not create thread key.
75	Could not detach thread.
76	Could not initialize thread condition.
77	Could not initialize thread mutex.
78	Could not switch to user.
79	Could not switch to group.

CAVEATS

1. Stemming can be done **only** when searching through and index of files that are in English because the Porter stemming algorithm used only stems English words.
2. When run as a daemon using a TCP socket, there are no security restrictions on who may connect and search. The code to implement domain and IP address restrictions isn't worth it since such things are better handled by firewalls and routers.
3. XML output can currently only be obtained for actual search results and not word, index, meta-name, or stop-word dumps.

FILES

`swish++.conf`	default configuration file name
`swish++.index`	default index file name

SEE ALSO

index(1), **perlfunc**(1), **exec**(2), **fork**(2), **unlink**(2), **accept**(3), **bind**(3), **listen**(3), **select**(3), **swish++.conf**(4), **searchmonitor**(8)

Tim Bray, et al. *Extensible Markup Language (XML) 1.0*, February 10, 1998.

Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads Programming*, O'Reilly & Associates, Sebastopol, CA, 1996.

M.F. Porter. "An Algorithm For Suffix Stripping," *Program*, 14(3), July 1980, pp. 130-137.

W. Richard Stevens. *Unix Network Programming, Vol 1, 2nd ed.*, Prentice-Hall, Upper Saddle River, NJ, 1998.

Larry Wall, et al. *Programming Perl, 3rd ed.*, O'Reilly & Associates, Inc., Sebastopol, CA, 2000.

AUTHOR

Paul J. Lucas <*pauljlucas@mac.com*>