SWISH-E

Simple Web Indexing System for Humans - Enhanced

SWISH-E 2

SWISH-Enhanced is a fast, powerful, flexible, free, and easy to use system for indexing collections of Web pages or other text files. Key features include the ability to limit searches to certain HTML tags (META, TITLE, comments, etc.). SWISH-E is an enhanced version of **SWISH**, which was originally written by Kevin Hughes and modified and released with his permission.

SWISH-E 2 is a new version with much of the code base rewritten for additional features, increased speed of indexing and searching, and improved security. See the SWISH-E L file for a list of features, downloading instructions, documentation overview, and support information.

A product with a similar-sounding name, but completely different is SWISH++.

Swish Version 2.4.3
Documentation created on Dec, 1 2005

1   The Swish-e README File

1.1 Upgrading?

If you are upgrading Swish-e, please review the CHANGES file before installation. The index format may change and existing indexes may need to be re-created before use.

1.2 What is Swish-e?

Swish-e is **S**imple **W**eb **I**ndexing **S**ystem for **H**umans - **E**nhanced. Swish-e can quickly and easily index directories of files or remote web sites and search the generated indexes.

Swish-e is extremely fast in both indexing and searching, highly configurable, and can be seamlessly integrated with existing web sites to maintain a consistent design. Swish-e can index web pages, but can just as easily index text files, mailing list archives, or data stored in a relational database.

Swish is designed to index small- to medium-sized collection of documents, Although a few users are indexing over a million documents, typical usage is more often in the tens of thousands. Currently, Swish-e only indexes eight bit character encodings.

Swish-e version 2.2 was a major rewrite of the code and the addition of many new features. Memory requirements for indexing have been reduced and indexing speed is significantly improved from previous versions. New features allow more control over indexing, better document parsing, improved indexing and searching logic, better filter code, and the ability to index from any data source.

Swish-e version 2.4 includes a major rewrite of the C API and a new Perl module for accessing the Swish-e C library. In addition, Swish-e 2.4 uses the GNU Auto Tools. The significant changes are where files are installed, and the use of Libtool to create the Swish-e library as a shared library on many platforms. Basically, installation is easier than previous versions, and more files are installed in "standard" locations (e.g. documentation is installed in `$prefix/share/doc/swish-e`).

Note: Due to the new build and installation system in Swish-e 2.4, some documentation may incorrectly list the location of files. Please report any documentation errors to the Swish-e Discussion list.

Swish-e is not a "turn-key" indexing and searching solution. The Swish-e distribution contains most of the parts to create such a system, but you need to put the parts together as best meets your needs. This gives you the power to index and search your documents the way you wish and to seamlessly integrate a search engine into your web site or application.

To use Swish-e, you will need to configure Swish-e to index your documents, create an index by running Swish-e, and setup an interface such as a CGI script (a script is included) to search the index and display results. Swish uses helper programs to index documents of types that Swish-e cannot natively index. These programs may need to be installed separately from Swish-e.

Swish-e is an Open Source (see: http://opensource.org) program supported by developers and a large group of users. Please take time to join the Swish-e discussion list at http://Swish-e.org .

1.2.1 Key features

- Quickly index a large number of documents in different formats including text, HTML, and XML.
- Use "filters" to index other types of files such as PDF, gzip, or PostScript.
- Includes a web spider for indexing remote documents over HTTP. Follows Robots Exclusion Rules (including META tags).
- Can use an external program to supply documents to Swish-e, such as an advanced spider for your web server or a program to read and format records from a relational database.
- Document "properties" (some subset of the source document, usually defined as a META or XML elements) may be stored in the index and returned with search results.
- Document summaries can be returned with each search.
- Word stemming, soundex, metaphone, and double-metaphone indexing for "fuzzy" searching
- Phrase searching and wildcard searching
- Limit searches to HTML links.
- Use powerful Regular Expressions to select documents for indexing or exclusion.
- Easily limit searches to parts or all of your web site.
- Results can be sorted by relevance or by any number of properties in ascending or descending order.
- Limit searches to parts of documents such as certain HTML tags (META, TITLE, comments, etc.) or to XML elements.
- Can report structural errors in your XML and HTML documents.
- Index file is portable between platforms.
- A Swish-e library is provided to allow embedding Swish-e into your applications for very fast searching. A Perl module is available that provides a standard API for accessing Swish-e.
- Includes example search script with context summaries and search term and phrase highlighting. Can be used with popular Perl templating systems.
- Swish-e is fast.
- It’s Open Source and FREE! You can customize Swish-e and you can contribute your fancy new features to the project.

- Supported by on-line user and developer groups.

1.3 Where do I get Swish-e?

The current version of Swish-e can be found at:

http://Swish-e.org

Please make sure you use a current version of Swish-e.

Information about Windows binary distributions can also be found at this site.

1.4 How Do I Install Swish-e?

Read the INSTALL page.

Building from source is recommended. On most platforms, Swish-e should build without problems. A list of platforms where Swish-e has been built can be found in the INSTALL page. Information on building for VMS and Win32 can be found in sub-directories of the `src` directory. Check the Swish-e site for information about binary distributions (such as for Windows).

In addition to the INSTALL page, make sure you read the SWISH-FAQ page if you have any questions, or to get an idea of questions that you might someday ask.

Problems or questions about installing Swish-e should be directed to the Swish-e discussion list (see the Swish-e web site at http://Swish-e.org).

Please read `Where do I get help with Swish-e?` below before posting any questions to the Swish-e list.

1.5 The Swish-e Documentation

Documentation is provided as HTML pages installed in $prefix/share/doc/swish-e where `$prefix` is /usr/local if building from source, or /usr if installed as part of a package from your OS vendor. Under Windows `$prefix` is selected at installation time.

A subset of the documentation is installed as system man pages as well.

Documentation is also available on-line at http://swish-e.org.

If your system includes the required support files and programs, the distribution make files can also generate the documentation in these formats:

```
    Postscript
    PDF (Adobe Acrobat)
```

See INSTALL for information on creating the PDF and Postscript versions of the documentation.

Patches or updates to the documentation should be done against the POD files, located in the pod directory of the distribution, or (preferably) against the CVS repository.

1.6 Where do I get help with Swish-e?

If you need help with installing or using Swish-e, please subscribe to the Swish-e mailing list. Visit the Swish-e web site (listed above) for information on subscribing to the mailing list.

Before posting any questions, please read QUESTIONS AND TROUBLESHOOTING.

1.7 Speling mistakes

Please contact the Swish-e list with corrections to this documentation. Any help in cleaning up the docs will be appreciated!

Any patches should be made against the `.pod` files, not the `.html` files.

1.8 Swish-e Development

Swish-e is currently being developed as an Open-Source project on SourceForge http://sourceforge.net.

Contact the Swish-e list for questions about Swish-e development.

1.9 Swish-e's History

SWISH was created by Kevin Hughes, circa 1994, to fill the need of the growing number of Web administrators on the Internet - many of the indexing systems were not well documented, were hard to use and install, and were too complex for their own good. The system was widely used for several years, long enough to collect some bug fixes and requests for enhancements.

In Fall 1996, The Library of UC Berkeley received permission from Kevin Hughes to implement bug fixes and enhancements to the original binary. The result is Swish-enhanced or Swish-e, brought to you by the Swish-e Development Team.

1.10 Document Info

Each document in the Swish-e distribution contains this section. It refers only to the specific page it's located in, and not to the Swish-e program or the documentation as a whole.

$Id: README.pod,v 1.19 2004/10/02 23:56:30 whmoseley Exp $

.

2 INSTALL - Swish-e Installation Instructions

2.1 OVERVIEW

This document describes how to download, build, and install Swish-e from source. Also found below is a basic overview of using Swish-e to index documents, with pointers to other, more advanced examples.

This document also provides instructions on how to get help installing and using Swish-e (and the important information you should provide when asking for help). Please read these instructions **before requesting help** on the Swish-e discussion list. See QUESTIONS AND TROUBLESHOOTING.

Although building from source is recommended, some OS distributions (e.g., Debian) provide pre-compiled binaries. Check with your distribution for available packages. Build from source, if your distribution does not offer the current version of Swish-e.

Also, please read the Swish-e FAQ (SWISH-FAQ), as it answers many frequently-asked questions.

Swish-e knows how to index HTML, XML, and plain text documents. Helper applications and other tools are used to convert documents such as PDF or MS Word into a format that Swish-e can index. These additional applications and tools (listed below) must be installed separately. The process of converting documents is called "filtering".

NOTE: Swish-e version 4.2.0 installs a lot more files when running "make install". Be aware that the Swish-e documentation may thus include errors about where files are located. Please notify the Swish-e discussion list of any documentation errors.

2.1.1 Upgrading from previous versions of Swish-e

If you are upgrading from a previous version of Swish-e, read the CHANGES page first. The Swish-e index format may have changed and existing indexes may not work with the newer version of Swish-e.

If you have existing indexes, you may need to re-index your data before running the "make install" step described below. Swish-e may be run from the build directory after compiling, but before installation.

2.1.2 Windows Users

A Windows binary version is available as a separate download from the Swish-e site (http://swish-e.org). Many of the installation instructions below will not apply to Windows users; the Windows version is pre-compiled and includes *libxml2*, *zlib*, *xpdf*, and *catdoc*.

A number of Perl modules may also be needed. These can be installed with ActiveState's PPM utility.

```
libwww-perl   - the LWP modules (for spidering)
HTML-Tagset   - used by web spider
HTML-Parser   - used by web spider
MIME-Types    - used for filtering documents when not spidering
HTML-Template - formatting output from swish.cgi (optional)
HTML-FillInForm (if HTML-Template is used)
```

2.2 SYSTEM REQUIREMENTS

Swish-e makes use of a number of libraries and tools that are not distributed with Swish-e. Some libraries need to be installed before building Swish-e from source; other tools can be installed at any time. See below for details.

2.2.1 Software Requirements

Swish-e is written in C. It has been tested on a number of platforms, including Sun/Solaris, Dec Alpha, BSD, Linux, Mac OS X, and Open VMS.

The GNU C compiler (gcc) and GNU make are strongly recommended. Repeat: you will find life easier if you use the GNU tools.

2.2.2 Optional but Recommended Packages

Most of the packages listed below are available as easily installable packages. Check with your operating system vendor or install them from source. Most are very common packages that may already be installed on your computer.

As noted below, some packages need to be installed before building Swish-e from source, while others may be added after Swish-e is installed.

- **Libxml2**

  libxml2 is very strongly recommended. It is used for parsing both HTML and XML files. Swish-e can be built and installed without *libxml2*, but the HTML parser that is built into Swish-e is not as accurate as *libxml2*.

  ```
  http://xmlsoft.org/
  ```

  libxml2 must be installed before Swish-e is built, or it will not be used.

  If *libxml2* is installed in a non-standard location (e.g., *libxml2* is built with `--prefix $HOME/local`), make sure that you add the `bin` directory to your `$PATH` before building Swish-e. Swish-e's configure script uses a program created by *libxml2* (`xml2-config`) to find the location of *libxml2*. Use `which xml2-config` to verify that the program can be found where expected.

- **Zlib Compression**

  The *Zlib* compression library is commonly installed on most systems and is recommended for use with Swish-e. *Zlib* is used for compressing text stored in the Swish-e index.

  ```
  http://www.gzip.org/zlib/
  ```

Zlib must be installed before building Swish-e.

- **Perl Modules**

  Although Swish-e is a compiled C program, many support features use Perl. For example, both the web spiders and modules to help with filtering documents are written in Perl.

  The following Perl modules may be required. Check your current Perl installation, as many may already be installed.

  ```
  LWP
  URI
  HTML::Parser
  HTML::Tagset
  MIME::Types (optional)
  ```

  Note that installing `Bundle::LWP` with the CPAN module

  ```
  perl -MCPAN -e 'install Bundle::LWP'
  ```

  will install many of the above modules.

  If you wish to use `HTML-Template` with swish.cgi to generate output, install:

  ```
  HTML::Template
  HTML::FillInForm
  ```

  If you wish to use `Template-Toolkit` with `swish.cgi` to generate output, install:

  ```
  Template
  ```

  Questions about installing these modules may be sent to the Swish-e discussion list.

  The `search.cgi` example script requires both `Template-Toolkit` and `HTML::FillInForm`.

- **Indexing PDF Documents**

  Indexing PDF files requires the `xpdf` package. This is a common package, available with most operating systems and often provided as an add-on package.

  ```
  http://www.foolabs.com/xpdf/
  ```

  Xpdf may be added after Swish-e is installed.

- **Indexing MS Word Documents**

  Indexing MS Word documents requires the *Catdoc* program.

  ```
  http://www.45.free.net/~vitus/ice/catdoc
  ```

  Catdoc may be added after Swish-e is installed.

- **Indexing MP3 ID3 Tags**

  Indexing MP3 ID3 Tags requires the `MP3::Tag` Perl module. See http://search.cpan.org. `MP3::Tag` may be installed after Swish-e is installed.

- **Indexing MS Excel Files**

  Indexing MS Excel files is supported by the following Perl modules, also available at http://search.cpan.org.

  ```
  Spreadsheet::ParseExcel
  HTML::Entities
  ```

  These Perl modules may be installed after Swish-e is installed.

2.3 INSTALLATION

Here are brief installation instructions that should work in most cases. Following this section are more detailed instructions and examples.

2.3.1 Building Swish-e

Download Swish-e using your favorite web browser or a utility such as `wget`, `lynx`, or `lwp-download`. Unpack and build the distribution, using the following steps:

Note: "swish-e-2.4.0" is used as an example. Download the most current available version and adjust the commands below! Also, if you are running Debian, see the notes below on building a `.deb` package from the Swish-e source package.

Pay careful attention to the "prompt" character used on the following command lines. A "$" prompt indicates steps run as an unprivileged user. A "#" indicates steps run as the superuser (root).

```
$ wget http://swish-e.org/Download/swish-e-2.4.0.tar.gz
$ gzip -dc swihs-e-2.4.0.tar.gz | tar xof -
$ cd swish-e-2.4.0  (this directory will depend on the version of Swish-e)
```

```
$ ./configure
$ make
$ make check
...
==================
All 3 tests passed
==================
```

```
$ su root  (or use sudo)
(enter password)
```

```
# make install
# exit
$ swish-e -V
SWISH-E 2.4.0
```

IMPORTANT: Once Swish-e is installed, do not run it as the superuser (root) -- root is only required during the installation step, when installing into system directories. Please do not break this rule.

NOTE: If you are upgrading from an older version of Swish-e, be sure and review the CHANGES file. Old index files may not be compatible with newer versions of Swish-e. After building Swish-e (but before running "make install"), Swish-e can be run from the build directory:

```
$ src/swish-e -V
```

To minimize downtime, create new index files before running "make install", by using Swish-e from the build directory. Then, copy the index files to the live location and run "make install":

```
$ src/swish-e -c /path/to/config -f index.new
```

Keep in mind that the location you index from may affect the paths stored in the index file.

2.3.2 Installing without root access

Here's another installation example. This might be used if you do not have root access or you wish to install Swish-e someplace other than `/usr/local`.

This example also shows building Swish-e in a "build" directory that is separate from where the source files are located. This is the recommended way to build Swish-e, but it requires GNU Make. Without GNU Make, you will likely need to build from within the source directory, as shown in the previous example.

```
$ tar zxof swish-e-2.4.0.tar.gz  (GNU tar with "z" option)
$ mkdir build
$ cd build
```

Note that the current directory is not where Swish-e was unpacked.

Swish-e uses a *configure* script. *configure* has many options, but it uses reasonable and standard defaults. Running

```
$ ../swish-e-2.4.0/configure --help
```

will display the options.

Two options are of common interest: `--prefix` sets the top-level installation directory; `--disable-shared` will link Swish-e statically, which may be needed on some platforms (Solaris 2.6, perhaps).

Platforms may require varying link instructions when libraries are installed in non-standard locations. Swish-e uses the GNU *autoconf* tools for building the package. *autoconf* is good at building and testing, but still requires you to provide information appropriate for your platform. This may mean reading the manual page for your compiler and linker to see how to specify non-standard file locations.

For most Unix-type platforms, you can use `LDFLAGS` and `CPPFLAGS` environment variables to specify paths to "include" (header) files and to libraries that are not in standard locations.

In this example, we do not have root access. We have installed *libxml2* and *libz* in `$HOME/local`. Swish-e will also be installed in `$HOME/local` (by using the `--prefix` setting).

In this case, you would need to add `$HOME/local/bin` to the start of your shell's `$PATH` setting. This is required because *libxml2* installs a program that is used when running the configure script. Before running configure, type:

```
$ which xml2-config
```

It should list `$HOME/local/bin/xml2-config`.

Now run *configure* (remember, we are in a separate "build" directory):

```
$ ../swish-e-2.4.0/configure \
    --prefix=$HOME/local \
    CPPFLAGS=-I$HOME/local/include \
    LDFLAGS="-R$HOME/local/lib -L$HOME/local/lib"
```

```
$ make >/dev/null  (redirect output to only see warnings and errors)
```

```
$ make check
...
==================
All 3 tests passed
==================
$ make install
$ $HOME/local/bin/swish-e -V
SWISH-E 2.4.0
```

Note the use of double quotes in the `LDFLAGS` line above. This allows `$HOME` to be expanded within the text string.

2.3.3 Run-time paths

The `-R` option says to add a specified path (or paths) to those that are used to find shared libraries at run time. These paths are stored in the Swish-e binary. When Swish-e is run, it will look in these directories for shared libraries.

Some platforms may not support the `-R` option. In this event, set the `LD_RUN_PATH` environment variable **before** running make.

Some systems, such as Redhat, do not look in `/usr/local/lib` for libraries. In these cases, you can either use `-R`, as above, when building Swish-e or add `/usr/local/lib` to `/etc/ld.so.conf` and run *ldconfig* as root.

If all else fails, you may need to actually read the man pages for your platform.

2.3.4 Building a Debian Package

The Swish-e distribution includes the files required to build a Debian package.

```
    $ tar zxof swish-e-2.4.0.tar.gz  (GNU tar with "z" option)
    $ cd swish-e-2.4.0
    $ fakeroot debian/rules binary
    [lots of output]
    dpkg-deb: building package ‘swish-e’ in ‘../swish-e_2.4.0-0_i386.deb’.
    $ su
    # dpkg -i ../swish-e_2.4.0-0_i386.deb
```

2.3.5 What’s installed

Swish installs a number of files. By default, all files are installed below `/usr/local`, but this can be changed by setting `--prefix` when running *configure* (as shown above). Individual paths may also be set. Run `configure --help` for details.

```
  $prefix/bin/swish-e          The Swish-e binary program
  $prefix/share/doc/swish-e/  Full documentation and examples
  $prefix/lib/libswish-e      The Swish-e C library
  $prefix/include/swish-e.h   The library header file
  $prefix/man/man1/           Documentation as manual pages
  $prefix/lib/swish-e/        Helper programs (spider.pl, swishspider, swish.cgi)
  $prefix/lib/swish-e/perl/   Perl helper modules
```

Note that the Perl modules are *not* installed in the system Perl library. Swish-e and the Perl scripts that require the modules know where to find the modules, but the *perldoc* program (used for reading documentation) does not. This can be corrected by adding `$prefix/lib/swish-e` and `$prefix/lib/swish-e/perl` to the `PERL5LIB` environment variable.

2.3.6 Documentation

Documentation can be found in the `$prefix/share/doc/swish-e` directory. Documentation can also be read on-line at the Swish-e web site:

```
    http://swish-e.org/
```

2.3.7 The Swish-e documentation as man(1) pages

Running "make install" installs some of the Swish-e documentation as man pages. The following man pages are installed:

```
    SWISH-FAQ(1)
    SWISH-CONFIG(1)
    SWISH-RUN(1)
    SWISH-LIBRARY(1)
```

The man pages are installed, by default, in the system man directory. This directory is determined when *configure* is run; it can be set by passing a directory name to *configure*.

For example,

```
    ./configure --mandir=/usr/local/doc/man
```

The man directory is specified relative to the `--prefix` setting. If you use `--prefix`, you do not normally need to also specify `--mandir`.

Information on running *configure* can be found by typing:

```
    ./configure --help
```

2.3.8 Join the Swish-e discussion list

The final step, when installing Swish-e, is to join the Swish-e discussion list.

The Swish-e discussion list is the place to ask questions about installing and using Swish-e, see or post bug fixes or security announcements, and offer help to others. Please do not contact the developers directly.

The list is typically *very low traffic*, so it won't overload your inbox. Please take the time to subscribe. See http://Swish-e.org.

If you are using Swish-e on a public site, please let the list know, so that your URL can be added to the list of sites that use Swish-e!

Please review the next section before posting questions to the Swish-e list.

2.4 QUESTIONS AND TROUBLESHOOTING

Support for installation, configuration, and usage is available via the Swish-e discussion list. Visit http://swish-e.org for information. Do not contact developers directly for help -- always post your question to the list.

It's very important to provide the right information when asking for help.

Please search the Swish-e list archive before posting a question. Also, check the SWISH-FAQ to see if your question has already been asked and answered.

Before posting, use the available tools to narrow down the problem.

Swish-e has several switches (e.g., `-T`, `-v`, and `-k`) that may help you resolve issues. These switches are described on the SWISH-RUN page. For example, if you cannot find a document by a keyword that you believe should be indexed, try indexing just that single file and use the `-T INDEXED_WORDS` option to see if the word is actually being indexed. First, try it without any changes to default settings:

```
    swish-e -i testdoc.html -T indexed_words | less
```

if that works, add in your configuration file:

```
    swish-e -i testdoc.html -c swish.conf -T indexed_words | less
```

If it still isn't working as you expect, try to reduce the test document to a very small example. This will be very helpful to your readers, when you are asking for help.

Another useful trick is to use `-H9` when searching, to display full headers in search results. Look at the "Parsed Words" header to see what words Swish-e is searching for.

2.4.1 When posting, please provide the following information:

Use these guidelines when asking for help. The most important tip is to provide the **least** amount of information that can be used to reproduce your problem. Do not paraphrase output -- copy-and-paste -- but trim text that is not necessary.

- The exact version of Swish-e that you are using. Running Swish-e with the `-V` switch will print the version number. Also, supply the output from `uname -a` or similar command that identifies the operating system you are running on. If you are running an old version of swish, be prepared for a response of "upgrade" to your question.
- A summary of the problem. This should include the commands issued (e.g. for indexing or searching) and their output, along with an explanation of why you don't think it's working correctly. Please copy-and-paste the exact commands and their output, instead of retyping, to avoid errors.
- Include a copy of the configuration file you are using, if any. Swish-e has reasonable defaults, so in many cases you can run it without using a configuration file. But, if you need to use a configuration file, **reduce it down** to the absolute minimum number of commands that is required to demonstrate your problem. Again, copy-and-paste.
- A small copy of a source document that demonstrates the problem.

  If you are having problems spidering a web server, use lwp-download or wget to copy the file locally, then make sure you can index the document using the file system method. This will help you determine if the problem is with spidering or indexing.

  If you expect help with spidering, don't post fake URLs, as it makes it impossible to test. If you don't want to expose your web page to the people on the Swish-e list, find some other site to test spidering on. If that works, but you still cannot spider your own site, you may need to request help from others. If so, you must post your real URL or make a test document available via some other source.
- If you are having trouble building Swish-e, please copy-and-paste the output from make (or from `./configure`, if that's where the problem is).

The key is to provide enough information so that others may reproduce the problem.

2.5 ADDITIONAL INSTALLATION OPTIONS

These steps are not required for normal use of Swish-e.

2.5.1 The SWISH::API Perl Module

The Swish-e distribution includes a module that provides a Perl interface to the Swish-e C library. This module provides a way to search a Swish-e index without running the Swish-e program. Searching an index will be many times faster when running under a persistent environment such as Apache/mod_perl with the `SWISH::API` module.

See the *perl/README* file for information on installing and using the `SWISH::API` Perl module.

2.5.2 Creating PDF and Postscript documentation

The HTML version of the Swish-e documentation was created with `Pod::HtmlPsPdf`, a package of Perl modules written and/or modified by Stas Bekman to automate the conversion of documents in POD format (see `perldoc perlpod`) to HTML, PostScript, and PDF. A slightly modified version of this package is included with the Swish-e distribution and used for building the HTML.

If your system has the **necessary tools** to build PostScript and the converter `ps2pdf` is installed, you may be able to build the PostScript and PDF versions of the documentation. After you have run *configure*, go to the `doc` directory of the distribution and type:

```
    make pdf
```

With any luck, you will end up with the these two files in the top-level directory:

```
    swish-e_documentation.pdf
    swish-e_documentation.ps
```

Most people, however, find reading the documentation in HTML to be the most convenient approach.

2.6 GENERAL CONFIGURATION AND USAGE

This section should give you a basic overview of indexing and searching with **Swish-e**. Other examples can be found in the `conf` directory; these will step you through a number of different configurations. Also, please review the SWISH-FAQ.

Swish-e is a command-line program. The program is controlled by passing switches on the command line. A configuration file may be used, but often is not required. Swish-e does not include a graphical user interface. Example CGI scripts are provided in the distribution, but they require additional setup to use.

2.6.1 Introduction to Indexing and Searching

Swish-e can index files that are located on the local file system. For example, running:

```
     swish-e -i /var/www/htdocs
```

will index *all* files in the `/var/www/htdocs` directory. You may specify one or more files or directories with the `-i` option. By default, this will create an index called `index.swish-e` in the current directory.

To search the resulting index for a given word, try:

```
 swish-e -w apache
```

This will find the word "apache" in the body or title of the indexed documents.

As mentioned above, Swish-e will index all files in a directory, unless instructed otherwise. So, if `/var/www/htdocs` contains non-HTML files, you will need a configuration file to limit the files that Swish-e indexes. Create a file called `swish.conf`:

```
# Example configuration file

# Tell Swish-e what to index (same as -i switch above)
IndexDir /var/www/htdocs

# Only index HTML and text files
IndexOnly .htm .html .txt

# Tell Swish-e that .txt files are to use the text parser.
IndexContents TXT* .txt

# Otherwise, use the HTML parser
DefaultContents HTML*
```

After saving the configuration file, reindex:

```
swish-e -c swish.conf
```

The Swish-e configuration settings are described in the SWISH-CONFIG manual page. The order of statements in the configuration file is typically not important, although some statements depend on previously set statements. There are many possible settings. Good advice is to use as few settings as possible when first starting out with Swish-e.

The runtime options (switches) are described in the SWISH-RUN manual page. You may also see a summary of options by running:

```
swish-e -h
```

Swish-e has two other methods for reading input files. One method uses a Perl helper script and the LWP Perl library to spider remote web sites:

```
swish-e -S http -i http://localhost/index.html -v2
```

This will spider the web server running on the local host. The `-S` option defines the input source method to be "http", `-i` specifies the URL to spider, and `-v` sets the verbose level to two. There are a number of configuration options that are specific to the `-S` http input source. See SWISH-CONFIG. Note that only files of `Content-Type text/*` will be indexed.

The `-S http` method is deprecated, however, in favor of a variation on the following input method.

There is a general-purpose input method wherein Swish-e reads input from a program that produces documents in a special format. The program might read and format data stored in a database, or parse and format messages in a mailing list archive, or run a program that spiders web sites (like the previous method).

The Swish-e distribution includes a spider program that uses this method of input. This spider program is much more configurable and feature-rich than the previous (`-S http`) method.

To duplicate the previous example, create a configuration file called `swish2.conf`:

```
# Example for spidering
# Use the "spider.pl" program included with Swish-e
IndexDir spider.pl
```

```
# Define what site to index
SwishProgParameters default http://localhost/index.html
```

Then, create the index using the command:

```
swish-e -S prog -c swish2.conf
```

This says to use the `-S prog` input source method. Note that, in this case, the `IndexDir` setting does not specify a file or directory to index, but a program name to be run. This program, `spider.pl`, does the work of fetching the documents from the web server and passing them to Swish-e for indexing.

The `SwishProgParameters` option is a special feature that allows passing command-line parameters to the program specified with `IndexDir`. In this case, we are passing the word `default` (which tells `spider.pl` to use default settings) and the URL to spider.

Running a script under Windows requires specifying the interpreter (e.g., `perl.exe`) and then using `SwishPropParameters` to specify the script and the script's parameters. See *Notes when using `-S prog` on MS Windows* on the SWISH-RUN page.

The advantage of the `-S prog` method of spidering (over the previous `-S http` method) is that the Perl code is only compiled once instead of once for every document fetched from the web server. In addition, it is a much more advanced spider with many, many features. Still, as used here, `spider.pl` will automatically index PDF or MS Word documents if (when) Xpdf and Catdoc are installed.

A special form of the `-S prog` input source method is:

```
./myprog --option | swish-e -S prog -i stdin -c config
```

This allows running Swish-e from a program (instead of running the external program from Swish-e). So, this also can be done as:

```
./myprog --option > outfile
swish-e -S prog -i stdin -c config < outfile
```

or

```
./myprog --option > outfile
cat outfile | swish-e -S prog -i stdin -c config
```

One final note about the `-S prog` input source method. The program specified with `-i` or `IndexDir` needs to be an absolute path. The exception is when the program is installed in the `libexecdir` directory. Then, a plain program name may be specified (as in the example showing `spider.pl`, above).

All three input source methods are described in more detail on the SWISH-RUN page.

2.6.2 Metanames and Properties

There are two key Swish-e concepts that you need to be familiar with: Metanames and Properties.

- **Metanames**

  Swish-e creates a reverse (i.e., inverted) index. Just like an index in a book, you look up a word and it lists the pages (or documents) where that word can be found.

  Swish-e can create multiple index tables within the same index file. For example, you might want to create an index that only contains words in HTML titles, so that searches can be limited to title text. Or, you might have descriptive words that you would like to search, stored in a meta tag called "keywords".

  Some database systems might call these different "fields" or "columns", but Swish-e calls them *MetaNames* (as a result of its first indexing HTML "meta" tags).

  To find documents containing "foo" in their titles, you might run:

  ```
  swish-e -w swishtitle=foo
  ```

  or, a more advanced example:

```
swish-e -w swishtitle=(foo or bar) or swishdefault=(baz)
```

The Metaname "swishdefault" is the name that is used by Swish-e if no other name is specified. The following two searches are thus equivalent:

```
swish-e -w foo
swish-e -w swishdefault=foo
```

When indexing HTML documents, Swish-e indexes words in the body and title under the Metaname "swishdefault".

- **Properties**

  Swish-e's search result is a list of files -- actually, Swish-e uses file numbers internally. Data can be associated with each file number when indexing. For example, by default Swish-e associates the file's name, title, last modified date, and size with the file number. These items can be printed in search results.

  In Swish-e, this associated data is called a file's *Properties*. Properties can be any data you wish to associated with a document -- in fact, the entire text of the document can be stored in the index. What data is stored as a Property is controlled by the *PropertyNames* (and other) configuration directives.

  What properties are printed with search results depends on the `-x` or `-p` switches. By default, Swish-e returns the rank, path/URL, title, and file size in bytes for each result.

2.6.3 Getting Started With Swish-e

Swish-e reads a configuration file (see SWISH-CONFIG) for directives that control whether and how Swish-e indexes files. Swish-e is also controlled by command-line arguments (see SWISH-RUN). Many of the command-line arguments have equivalent configuration directives (e.g., `-i` and `IndexDir`).

Swish-e does not require a configuration file, but most people change its default behavior by placing settings in a configuration file.

To try the examples below, go to the `tests` subdirectory of the distribution. The tests will use the `*.html` files in this directory when creating the test index. You may wish to review these `*.html` files to get an idea of the various native file formats that Swish-e supports.

You may also use your own test documents. It's recommended to use small test documents when first using Swish-e.

2.6.4 Step 1: Create a Configuration File

The configuration file controls what and how Swish-e indexes. The configuration file consists of directives, comments, and blank lines. The configuration file can be any name you like.

This example will work with the documents in the *tests* directory. You may wish to review the *tests/test.config* configuration file used for the `make test` tests.

For example, a simple configuration file (*swish-e.conf*):

```
# Example Swish-e Configuration file
```

```
# Define *what* to index
# IndexDir can point to a directories and/or a files
# Here it’s pointing to the current directory
# Swish-e will also recurse into sub-directories.
IndexDir .
```

```
# But only index the .html files
IndexOnly .html
```

```
# Show basic info while indexing
IndexReport 1
```

And that’s a simple configuration file. It says to index all the `.html` files in the current directory and sub-directories, if any, and provide some basic output while indexing.

As mentioned above, the complete list of all configuration file directives is detailed in SWISH-CONFIG.

2.6.5 Step 2: Index your Files

Run Swish-e, using the `-c` switch to specify the name of the configuration file.

```
swish-e -c swish-e.conf
```

```
Indexing Data Source: "File-System"
Indexing "."
Removing very common words...
no words removed.
Writing main index...
Sorting words ...
Sorting 55 words alphabetically
Writing header ...
Writing index entries ...
  Writing word text: Complete
  Writing word hash: Complete
  Writing word data: Complete
55 unique words indexed.
4 properties sorted.
5 files indexed.  1252 total bytes.  140 total words.
Elapsed time: 00:00:00 CPU time: 00:00:00
Indexing done!
```

This created the index file `index.swish-e`. This is the default index file name, unless the **IndexFile** directive is specified in the configuration file:

```
IndexFile ./website.index
```

You may use the `-f` switch to specify a index file at indexing time. The `-f` option overrides any `IndexFile` setting that may be in the configuration file.

2.6.6 Step 3: Search

You specify your search terms with the `-w` switch. For example, to find the files that contain the word `sample`, you would issue the command:

```
swish-e -w sample
```

This example assumes that you are in the `tests` directory. Swish-e returns the following, in response to this command:

```
swish-e -w sample
```

```
# SWISH format: 2.4.0
# Search words: sample
# Number of hits: 2
# Search time: 0.000 seconds
# Run time: 0.005 seconds
1000 ./test_xml.html "If you are seeing this, the METATAG XML search was successful!" 159
1000 ./test.html "If you are seeing this, the test was successful!" 437
.
```

So, the word `sample` was found in two documents. The first number shown is the relevance (or rank) of the search term, followed by the file containing the search term, the title of the document, and finally, the length of the document (in bytes).

The period ("."), sitting alone at the end, marks the end of the search results.

Much more information may be retrieved while searching, by using the `-x` and `-H` switches (see SWISH-RUN) and by using Document Properties (see SWISH-CONFIG).

2.6.7 Phrase Searching

To search for a phrase in a document, use double-quotes to delimit your search terms. (The default phrase delimiter is set in `src/swish.h`.)

You must protect the quotes from the shell.

For example, under Unix:

```
    swish-e -w '"this is a phrase" or (this and that)'
    swish-e -w 'meta1=("this is a phrase") or (this and that)'
```

Or under the Windows `command.com` shell.

```
    swish-e -w \"this is a phrase\" or (this and that)
```

The phrase delimiter can be set with the `-P` switch.

2.6.8 Boolean Searching

You can use the Boolean operators **and**, **or**, or **not** in searching. Without these Boolean operatots, Swish-e will assume you're **and**ing the words together.

Here are some examples:

```
    swish-e -w 'apples oranges'
    swish-e -w 'apples and oranges'  ( Same thing )
```

```
    swish-e -w 'apples or oranges'
```

```
    swish-e -w 'apples or oranges not juice' -f myIndex
```

retrieves first the files that contain both the words "apples" and "oranges"; then among those, selects the ones that do not contain the word "juice".

A few other examples to ponder:

```
    swish-e -w 'apples and oranges or pears'
    swish-e -w '(apples and oranges) or pears'  ( Same thing )
    swish-e -w 'apples and (oranges or pears)'  ( Not the same thing )
```

Swish processes the query left to right.

See SWISH-SEARCH for more information.

2.6.9 Context Searching

The `-t` option in the search command line allows you to search for words that exist only in specific HTML tags. This option takes a string of characters as its argument. Each character represents a different tag in which the word is searched; that is, you can use any combinations of the following characters:

```
H search in all <HEAD> tags
B search in the <BODY> tags
t search in <TITLE> tags
h is <H1> to <H6> (header) tags
e is emphasized tags (this may be <B>, <I>, <EM>, or <STRONG>)
c is HTML comment tags (<!-- ... -->)
```

For example:

```
# Find only documents with the word "linux" in the <TITLE> tags.
swish-e -w linux -t t
```

```
# Find the word "apple" in titles or comments
swish-e -w apple -t tc
```

2.6.10 META Tags

As mentioned above, Metanames are a way to define "fields" in your documents. You can use the Metanames in your queries to limit the search to just the words contained in that META name of your document. For example, you might have a META-tagged field called `subjects` in your documents. This would let you search your documents for the word "foo", but only return documents where "foo" is within the `subjects` META tag.

Document *Properties* are somewhat related: Properties allow the content of a META tag in a source document to be stored within the index, and that text to be returned along with search results.

META tags can have two formats in your documents.

```
<META NAME="keyName" CONTENT="some Content">
```

And in XML format

```
<keyName>
    Some Content
</keyName>
```

If using *libxml*, you can optionally use a non-HTML tag as a metaname:

```
<html>
    <body>
        Hello swish users!
        <keyName>
            this is meta data
        </keyName>.
    </body>
```

This, of course, is invalid HTML.

To continue with our sample `Swish-e.conf` file, add the following lines:

```
# Define META tags
MetaNames meta1 meta2 meta3
```

Reindex to include the changes:

```
swish-e -c swish-e.conf
```

Now search, but this time limit your search to META tag `meta1`:

```
swish-e -w 'meta1=metatest1'
```

Again, please see SWISH-RUN and SWISH-CONFIG for complete documentation of the various indexing and searching options.

2.6.11 Spidering and Searching with a Web form.

This example demonstrates how to spider a web site and set up the included CGI script to provide a web-based search page. This example uses Perl programs that are included in the Swish-e distribution: *spider.pl* will be used for reading files from the web server; *swish.cgi* will provide the web search form and display results.

As an example, we will index the Apache Web Server documentation, installed on the local computer at http://localhost/apache_docs/index.html.

1. **Make a Working Directory**

   Create a directory to store the Swish-e configuration and the Swish-e index.

   ```
   ~$ mkdir web_index
   ~$ cd web_index/
   ~/web_index$
   ```

2. **Create a Swish-e Configuration file**

   ```
   ~/web_index$ cat swish.conf
   # Swish-e config to index the Apache documentation
   #
   # Use spider.pl for indexing (location of spider.pl set at installation time)
   IndexDir spider.pl
   ```

```
# Use spider.pl’s default configuration and specify the URL to spider
SwishProgParameters default http://localhost/apache_docs/index.html
```

```
# Allow extra searching by title, path
Metanames swishtitle swishdocpath
```

```
# Set StoreDescription for each parser
#  to display context with search results
StoreDescription TXT* 10000
StoreDescription HTML* <body> 10000
```

3. **Generate the Index**

   Now, run Swish-e to create the index:

```
~/web_index$ swish-e -S prog -c swish.conf
```

```
Indexing Data Source: "External-Program"
Indexing "spider.pl"
/usr/local/lib/swish-e/spider.pl: Reading parameters from ’default’
```

```
Summary for: http://localhost/apache_docs/index.html
    Duplicates:     4,188  (349.0/sec)
Off-site links:       276  (23.0/sec)
       Skipped:         1  (0.1/sec)
   Total Bytes: 2,090,125  (174177.1/sec)
    Total Docs:       147  (12.2/sec)
   Unique URLs:       149  (12.4/sec)
Removing very common words...
no words removed.
Writing main index...
Sorting words ...
Sorting 7736 words alphabetically
Writing header ...
Writing index entries ...
  Writing word text: Complete
  Writing word hash: Complete
  Writing word data: Complete
7736 unique words indexed.
5 properties sorted.
147 files indexed.  2090125 total bytes.  200783 total words.
Elapsed time: 00:00:13 CPU time: 00:00:02
Indexing done!
```

   The above output is actually a mix of output from both Swish-e and `spider.pl`. `spider.pl` reports the "Summary for: http://localhost/apache_docs/index.html".

Also note that Swish-e knows to find `spider.pl` at `/usr/local/lib/swish-e/spider.pl`. The script installation directory (called `libexecdir`) is set at configure time. You can see your setting by running `swish-e -h`:

```
~/web_index$ swish-e -h | grep libexecdir
 Scripts and Modules at: (libexecdir) = /usr/local/lib/swish-e
```

This directory will be needed in the next step, when setting up the CGI script.

Finally, verify that the index can be searched from the command line:

```
~/web_index$ swish-e -w installing -m3
# SWISH format: 2.4.0
# Search words: installing
# Removed stopwords:
# Number of hits: 17
# Search time: 0.018 seconds
# Run time: 0.050 seconds
1000 http://localhost/apache_docs/install.html "Compiling and Installing Apache" 17960
718 http://localhost/apache_docs/install-tpf.html "Installing Apache on TPF" 25734
680 http://localhost/apache_docs/windows.html "Using Apache with Microsoft Windows" 27165
.
```

Now, try limiting the search to the title:

```
~/web_index$ swish-e -w swishtitle=installing -m3
# SWISH format: 2.3.5
# Search words: swishtitle=installing
# Removed stopwords:
# Number of hits: 2
# Search time: 0.018 seconds
# Run time: 0.048 seconds
1000 http://localhost/apache_docs/install-tpf.html "Installing Apache on TPF" 25734
1000 http://localhost/apache_docs/install.html "Compiling and Installing Apache" 17960
.
```

Note that the above can also be done using the `-t` option:

```
~/web_index$ swish-e -w installing -m3 -tH
```

4. **Set up the CGI script**

Swish-e does not include a web server. So, you must use your locally installed web server. Apache is highly recommended, of course.

Locate your web server's CGI directory. This may be a `cgi-bin` directory in your home directory or a central `cgi-bin` directory set up by the web server administrator. Once this is located, copy the `swish.cgi` script into the `cgi-bin` directory.

Where CGI scripts can be located depends completely on the web server that is being used and how it has been configured. See your web server's documentation or your site's administrator for additional information.

This example will use a site `cgi-bin` directory, located at `/usr/lib/cgi-bin`. Copy the `swish.cgi` script into the `cgi-bin` directory. Again, we will need the location of the `libexecdir` directory:

```
~/web_index$ swish-e -h | grep libexecdir
 Scripts and Modules at: (libexecdir) = /usr/local/lib/swish-e
```

```
~/web_index$ cd /usr/lib/cgi-bin
/usr/lib/cgi-bin$ su
Password:
/usr/lib/cgi-bin# cp /usr/local/lib/swish-e/swish.cgi.
```

If your operating system supports symbolic links **and** your web server allows programs to be symbolic links, then you may wish to create a link to the `swish.cgi` program, instead.

```
/usr/lib/cgi-bin# ln -s /usr/local/lib/swish-e/swish.cgi
```

We need to tell the `swish.cgi` script where to look for the index created in the previous step. It's also recommended to enter the path to the swish-e binary. Otherwise, the `swish.cgi` script will look for the binary in the `PATH`, and that may change when running under the CGI environment.

Here's the configuration file:

```
/usr/lib/cgi-bin# cat .swishcgi.conf
return {
    title        => 'Search Apache Documentation',
    swish_binary => '/usr/local/bin/swish-e',
    swish_index  => '/home/moseley/web_index/index.swish-e',
}
```

Now, test the script from the command line (as a normal user!):

```
/usr/lib/cgi-bin# exit
exit
```

```
/usr/lib/cgi-bin$  ./swish.cgi | head
Content-Type: text/html; charset=ISO-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
       <title>
          Search Apache Documentation
       </title>
    </head>
    <body>
```

Notice that the CGI script returns the HTTP header (Content-Type) and the body of the web page, just like a well behaved CGI scrip should do.

Now, test using the web server (this step depends on the location of your `cgi-bin` directory). This example uses the "GET" command that is part of the LWP Perl library, but any web browser can run this test.

```
/usr/lib/cgi-bin$ GET http://localhost/cgi-bin/swish.cgi | head
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Tranitional//EN">
<html>
    <head>
       <title>
          Search Apache Documentation
       </title>
    </head>
    <body>
        <h2>
```

The script reports errors to stderr, so consult the web server's error log if problems occur. The message "Service currently unavailable", reported by running `swish.cgi`, typically indicates a configuration error; the exact problem will be listed in the web server's error log.

Detailed instructions on using the `swish.cgi` script and debugging tips can be found by running:

```
$ perldoc swish.cgi
```

while in the `cgi-bin` directory where `swish.cgi` was copied.

The spider program `spider.pl` also has a large number of configuration options.

Documentation is also available in the directory `$prefix/share/doc/swish-e` or at http://swish-e.org.

Note: Also check out the `search.cgi` script, found at the same location as the `swish.cgi` script. This is more of a skeleton script, for those that want to create a custom search script.

Now you are ready to search.

2.7 Indexing Other Types of Documents - Filtering

Swish-e can only index HTML, XML, and text documents. In order to index other documents, such as PDF or MS Word documents, you must use a utility to convert or "filter" those documents.

How documents are filtered with Swish-e has changed over time. This has resulting in a bit of confusion. It's also a somewhat complex process, as different programs need to communicate with each other.

You may wish to read the Swish-e FAQ question on filtering, before continuing here. How do I filter documents?

2.7.1 Filtering Overview

There are two ways to filter documents with Swish-e. Both are described in the SWISH-CONFIG man page. They use the `FileFilter` directive and the `SWISH::Filter` Perl module.

The `FileFilter` directive is a general-purpose method of filtering. It allows running of an external program for each document processed (based on file extension), and requires one or more external programs. These programs open an input file, convert as needed, and write their output to standard output.

Previous versions of Swish-e (before 2.4.0) used a collection of filter programs for converting files such as PDF or MS Word documents. The external programs call other program to do the work of filtering (e.g. *pdftotext* to extract the contents from PDF files). Although these filter programs are still included with the Swish-e distribution as examples, it is recommended to use the `SWISH::Filter` method, instead.

One disadvantage of using `FileFilter` is that the filter program is run once for every document that needs to be filtered. This can slow down the indexing process **substantially**.

The `SWISH::Filter` Perl module works very much like the old system and uses the same helper programs. Convieniently, however, it provides a single interface for filtering all types of documents. The primary advantage of `SWISH::Filter` is that it is built into the program used for spidering web sites (spider.pl), so all that's required is installing the filter programs that do the actual work of filtering (e.g. *catdoc*, *xpdf*). (The Windows binary includes some of the filter programs.)

But, Swish-e will not use `SWISH::Filter` by default when using the file system method of indexing. To use `SWISH::Filter` when indexing by file system method (-S fs), you can use a `FileFilter` directive with the `swish_filter.pl` filter (which is just a program that uses `SWISH::Filter`) or use the `-S prog` method of indexing and use the `DirTree.pl` program for fetching documents.

`DirTree.pl` is included with the Swish-e distribution and is designed to work with `SWISH::Filter`. Using DirTree.pl will likely be a faster way to index, since the `SWISH::Filter` set of modules does not need to be compiled for every document that needs to be filtered.

See the contents of `swish_filter.pl` and `DirTree.pl` for specifics on their use.

2.7.2 Filtering Examples

The `FileFilter` directive can be used in your config file to convert documents, based on their extensions. This is the old way of filtering, but provides an easy way to add filters to Swish-e.

For example:

```
    FileFilter .pdf  pdftotext   "'%p' -"
    IndexContents TXT* .pdf
```

will cause all `.pdf` files to be filtered through the *pdftotext* program (part of the *Xpdf* package) and to parse the resulting output (from *pdftotext*) with the text ("TXT") parser.

The other way to filter documents is to use a `-S prog` prograam and convert the documents before passing them onto Swish-e.

For example, `spider.pl` makes use of the `SWISH::Filter"` Perl module, included with the Swish-e distribution. `SWISH::Filter` is passed a document and the document's content type; it looks for modules and utilities to convert the document into one of the types that Swish-e can index.

Swish-e comes ready to index PDF, MS Word, MP3 ID3 tags, and MS Excel file types. But these filters need extra modules or tools to do the actual conversion.

For example, the Swish-e distribution includes a module called `SWISH::Filter::Pdf2HTML` that uses the *pdftotext* and *pdfinfo* utilities provided by the *Xpdf* package.

This means that if you are using `spider.pl` to spider your web site and you wish to index PDF documents, all that is needed is to install the Xpdf package and Swish-e (with the help of spider.pl) will begin indexing your PDF files.

Ok, so what does all that mean? For a very simple site, you should be able to run this:

```
    $ /usr/local/lib/swish-e/spider.pl default http://localhost/ | swish-e -S prog -i stdin
```

which is running the spider with default spider settings, indexing the Web server on localhost, and piping its output into Swish-e (using the default indexing settings). Documents will be filtered automatically, if you have the required helper applications installed.

Most people will not want to just use the default settings (for one thing, the spider will take a while because its default is to delay a few seconds between every request). So, read the documentation for `spider.pl`, to learn how to use a spider config file. Also read SWISH-CONFIG to learn about what configuration options can be used with Swish-e.

The `SWISH::Filter` documentation provides more details on filtering and hints for debugging problems when filtering.

2.8 Document Info

$Id: INSTALL.pod,v 1.40 2004/10/04 22:49:33 whmoseley Exp $

.

3 CHANGES - List of revisions

3.1 Revision History

This document contains list of bug fixes and feature additions to Swish-e.

3.1.1 Version 2.4.3 December 9, 2004

Improved error messsages when using incremental indexing

There was a bit of confusion on how to use incremental indexing (still experimental) so added better logic for error messages.

Also fixed a logic error when setting the incremental update mode. Caught by Paul Loner.

3.1.2 Version 2.4.3-pr1 - Wed Dec 1 09:52:50 PST 2004

"Fixed" libxml2's change in UTF8Toisolat1() return value

Bernhard Weisshuhn supplied a patch to parser.c for checking the return value of `UTF8Toisolat1().` Seems that libxml2 now returns the number of characters converted instead of zero for success.

```
http://bugzilla.gnome.org/show_bug.cgi?id=153937
```

Added swish-config and pkg-config

Swish now provides a swish-config script and config file for the pkg-config utility. These tools help when building programs that link with the swish-e library.

The SWISH::API Makefile.PL program uses swish-config to locate the installation directory of swish-e. This should make building SWISH::API easier when swish-e is installed in a non-standard location.

Fixed rank bias in merge

Peter van Dijk noticed that MetaNamesRank settings were not being copied to the output index when merging.

Added SwishFuzzy function

SwishFuzzy function (SWISH::API::Fuzzy) lets you stem a word without first searching. This might be helpful for playing with queries prior to the search.

Fixed translate character table

Michael Levy found an error in the table used to translate 8859-1 to ascii7. Luckily, it was an upper case translation and the table is only used on lower case characters.

MetaNamesRank documentation

Changed the ’not yet implemented’ caveat to ’implemented but experimental’.

Added Continuation option to config processing

You can now use continuation lines in the config file:

```
IgnoreWords \
    the \
    am \
    is \
    are \
    was
```

There may not be any characters following the backslash.

Fixed Buzzwords (and other word lists entered in the config)

Words entered in config were not converted to lower case before storing in the index.

Fixed metaname mapping problem in Merge

Peter Karman found an error when merging indexes where the source indexes had the same metanames, but listed in a different order in their config files. Words would then be indexed under the wrong metaID number in the output index.

SWISH::Filters and spider.pl updates

The web spider *spider.pl* was updated to work better with SWISH::Filter by default and also make it easier to use the spider default along with a spider config file. See spider.pl for details.

SWISH::Filter was updated. The way filters are created has changed. If you created your own filters you will need to update them. Take a look at SWISH::Filter and the filters included in the distribution.

Updates to Documentation

Richard Morin submitted formatting and punctuation dates to the README and INSTALL docs.

Added -R option to support IDF word weighting in ranking. (karman)

Added Inverse Document Frequency calculation to the `getrank()` routine. This will allow the relative frequency of a word in relationship to other words in the query to impact the ranking of documents.

Example: if ’foo’ is present twice as often as ’bar’ in the collection as a whole, a search for ’foo bar’ will weight documents with ’bar’ more heavily (i.e., higher rank) than those with ’foo’.

The impact is greatest when OR’ing words in a query rather than AND’ing them (which is the default).

Also added Rank discussion to the FAQ.

Updates to the example scripts

Updated PhraseHighlight.pm as suggested by Bill Schell for an optimization when all words in a document are highlighted.

Updated search.cgi and PhraseHighlight.pm to use the internal stemmers via the SWISH::API module as suggested by Jonas Wolf.

Leak when using C library

David Windmueller found a memory leak when calling multiple searches on a swish handle. The problem was swish loading the pre-sorted property index on every search, even after the table had been loaded into memory.

Swish.cgi now kills swish-e on time out

The example script *swish.cgi* uses an alarm (on platforms that support alarm) to abort processing after some number of seconds, but it was not killing the child process, swish-e. Bill Schell submitted a patch to kill the child when the alarm triggers.

The template search.tt was renamed to swish.tt

The template was renamed because it’s used by *swish.cgi*, not by *search.cgi*, which was confusing.

Updates to the search.cgi

The example script *search.cgi* was updated to work better with mod_perl and to use external template files and style sheets.

New MS Word Filter

James Job provided the SWISH::Filter::Doc2html filter that uses the wvWare (http://wvware.source-forge.net/) program for filtering MS Word documents. If both catdoc and wvWare are installed then wvWare will be used.

wvWare is reported to do a good job at converting MS Word docs to HTML. In a few tests it did work well, but other cases it failed to generate correct output. It was also much, much slower than catdoc. I tested with wvWare 0.7.3 on Debian Linux. Testing with both is recommended.

Change in way symbolic links are followed

John-Marc Chandonia pointed out that if a symlink is skipped by FileRules, then the actual file/directory is marked as "already seen" and cannot be indexed by other links or directly.

Now, files and directories are not marked "already seen" until after passing FileRules (i.e after a file is actually indexed or a directory is processed).

Could not set SwishSetSort() more than once

David Windmueller found a problem when trying to set the sort order more than once on an existing search object. Memory was not correctly reset after clearing the previous sort values.

Access MetaNames and PropertyNames from API

Patch provided by Jamie Herre to access the MetaNames and PropertyNames via the C API and to test via the testlib program. Swish::API also updated to access this data.

SwishResultPropertyULong() bug fixed

David Windmueller reported that `SwishResultPropertyULong()` was returning ULONG_MAX on all calls. This was fixed.

Null written to wrong location in file.c

Bill Schell with the help of valgrind found a null written past the end of a buffer in file.c in the code that supports the old parsers. This resulted in a segfault while indexing a large set of XML documents.

Fixed problem when indexing very large files

Steve Harris reported a problem when indexing a very large document that caused an integer overflow. José Ruiz updated to used unsigned integers.

Bump word position on block tags with HTML2 parser

Peter Karman pointed out the the libxml2 HTML parser was allowing phrase matches across block level html elements. Swish now bumps the word position on these elements.

3.1.3 Version 2.4.2 - March 09, 2004

- **UseStemming didn’t take no for an answer**

  UseStemming was coded as an alias for FuzzyIndexingMode when Snowball was compiled in (the default), but "no" doesn’t always mean no when the Norwegian stemmer is available.

- **Fixed problem building incremental version**

Fixed compile problem with building incremental indexing mode. This is an experimental option with swish-e to allow adding files to an index. See configure --help for build option. Incremental indexes are not compatible with standard indexes.

- **Updated build instructions in INSTALL**

  Added a few comments about use of CPPFLAGS and LDFLAGS.

- **Updated the index_hypermail.pl**

  Updated to work with latest version of hypermail (pre-2.1.9).

- **Time zone in ResultPropertyStr()**

  Format string for generating date did not include the time zone in location. Add strftime format string to config.h

- **Undefined and Blank Properties and (NULL)**

  Fixed a few problems with printing properties:

  1) Using -p and -x showed different results if a bad property value was given:

```
$ swish-e -w not dkdk -p badname -H0
err: Unknown Display property name "badname"
.
$ swish-e -w not dkdk -x '<badname>\n' -H0
(NULL)
```

  Now both return an error.

  2) Fixed bug where using a "fmt" string with -x output generated (bad) output if the result did not have the specified property.

```
$ swish-e -w not dkdk -x '<somedate>\n' -H0  # undefined value
```

```
$ swish-e -w not dkdk -x '<somedate fmt="%Y %B %d">\n' -H0
%Y %B 1075353525
```

  Now nothing is printed if the property does not exist.

  3) Updated SWISH::API to `croak()` on invalid property names, and to return undefined values for missing properties.

  4) Updated swish.cgi and search.cgi to not generate warnings on undefined values return as properties. Note that swish.cgi will now die on undefined properties. Previously would just display (NULL).

- **Fixed segfault when generating warnings while parsing**

  Parser.c was incorrectly calling `warning()` incorrectly. And -Wall was not catching this!

- **Added check for internal property names.**

  Parser was not checking for use of Swish-e reserved property names.

  ```
      <swishrank>foo</swishrank>
  ```

  This will now generate a warning.

3.1.4 Version 2.4.1 - December 17, 2003

- **Added new example CGI script**

  search.cgi is a new skeleton CGI script that uses SWISH::API for searching. It is installed in the same location as swish.cgi.

- **Add Fuzzy access to C and Perl interfaces**

  Added a number of functions to the C API (and SWISH::API) to access the stemmer used when indexing a given index.

- **Commas in numbers**

  Added commas to summary display at end of indexing.

- **Insert whitespace between tags**

  Parser.c was updated to flush the text buffer before and after every (non-inline HTML) tag.

  The problem was that:

  ```
      foo<tag>bar</tag>baz
  ```

  would index as a single word "foobarbaz".

- **DirTree.pl**

  DirTree.pl was updated to work with SWISH::Filter and to work on Windows. DirTree.pl is a program to fetch files from the file system and works with the -S prog input method.

- **Problem with --enable-incremental option**

Fixed configure script to build incremental option. Note that this is still experimental. But testers are welcome.

- **headers.c bug**

  Mark Fletcher with the help of valgrind found a bug in headers.c function SwishIndexHeaderNames used by the C API.

- **Clarify documentation regarding search order**

  At the prompting of Doralyn Rossmann updated SEARCH.pod to try and make the explanation of searching clearer, and to fix an error in the description of nested searches.

3.1.5 Version 2.4.0 - October 27, 2003

- **Note: Different Index Format**

  Swish-e version 2.4.0 has a different index file format from previous versions of Swish-e. Upgrading will **require** reindexing -- version 2.4.0 cannot read indexes created with previous versions.

3.1.6 Version 2.4.0 (Release Candidate 4) September 26, 2003

- **robots.txt not closed correctly**

  When using -S http method robots.txt was not closed and that caused the (last) .contents file to not be unlinked under Windows. Windows seems to think filenames are related to files.

- **SWISH::Filter and locating programs on Windows**

  SWISH::Filter now scans `$libexecdir` in addition to the PATH for programs (such at catdoc and pdftotext), and also checks for programs by adding the extensions ".exe" and ".bat" to the program name.

- **Install sample templates**

  The sample templates included with swish.cgi are now installed in `$pkgdatadir` (typically /usr/local/share/swish-e).

3.1.7 Version 2.4.0 (Release Candidate 3) September 11, 2003

- **Fix parser bug meta=(foo*)**

  Fixed bug in query parser caused in rc2’s (pr2) attempt to catch wildcards errors.

3.1.8 Version 2.4.0 (Release Candidate 2) September 10, 2003

- **Indexing HTML title**

  Fixed a problem when these were used in combination:

  ```
  MetaNames swishtitle
  MetaNameAlias swishtitle title
  ```

  That failed to correctly reset the metaname stack and indexed text under the wrong metaID.

- **Single Wildcards**

  Due to the way the query parser "works" a search of

  ```
   "foo *"
  ```

  would result in a search of "foo*". Now that results in:

  ```
   err: Single wildcard not allowed as word
  ```

- **Fixed search parsing bug**

  Brad Miele reported that the word "andes" was not being found. It was being stemmed to "and" when was then considered an operator. [moseley]

- **Add new directive PropertyNamesSortKeyLength**

  PropertyNamesSortKeyLength sets the sort key length to use when sorting string properties. The default is 100 characters. There was a hard-coded 100 char limit before, but that was a problem where people were not building from source (Windows). The value of this is questionable -- it's intended to limit how much memory is used when sorting while indexing and searching. [moseley]

- **Fixed sorting issues with multiple indexes and reverse sorting**

  Reworked much of the sorting code. Still to do is setting the character sort order. [moseley]

- **Fixed minor memory leak**

  Fixed leak of not releasing memory of index file name and swish_handle destroy, and fixed Swish-StemWord to default to the Stemmer_en. [moseley]

  Fixed libtest.c example program that was not cleaning up memory after an error condition.

- **Replaced Swish-e's Porter Stemmer with Snowball**

Swish-e now has support for Snowball stemmers (http://snowball.tartarus.org/). The stemmers are enabled for an index with FuzzyIndexingMode Stemming_* where "*" can be:

```
de, dk, en1, en2, es, fi, fr, it, nl, no, pt, ru, se
```

In addition, UseStemming yes or FuzzyIndexingMode Stemming_en will use the old stemmer.

3.1.9 Version 2.4.0 (Release Candidate 1) May 21, 2003

- **Security Fix: swish.cgi**

  The swish.cgi script was not correctly escaping HTML when searching by the right combination of metanames and highlighting module. This could lead to cross-site scripting if indexing un-trusted documents. [moseley]

- **Added Support for building a Debian Package**

  To build as a .deb unpack the distribution and chdir then run

  ```
    $ fakeroot debian/build binary
  ```

  Then install the generated .deb file with dpkg -i

- **Use SWISH::Filter by default with spider.pl**

  spider.pl is installed in the libexecdir directory as well as the SWISH::Filter modules. PDF, MS Word, MP3, and XML documents will be indexed automatically if the required helper applications (e.g. catdoc, pdftotext) or scripts (e.g. MP3::Tag) are installed.

  Swish also knows about libexecdir, so you you specify a relative path with -S prog swish-e will look for the program in libexecdir. This is mostly for spider.pl so indexing only requires:

  ```
      IndexDir spider.pl
      SwishProgParameters default http://localhost/index.html
  ```

  And swish-e will find spider.pl and SWISH::Filter will be used to convert docs.

- **Fixed Document-Type bug**

  Document-Type was not being reset after set input from a -S prog program causing the wrong parser to be used. [moseley]

- **New Directive: PropertyNamesNoStripChars**

Swish replaces all series of low ASCII chars with a single space character. This option instructs swish to store all chars in the property. [moseley]

- **Change HTTP access defaults**

  Defaults used with -S http access method were changed.

  Delay was reduced from one minute between start of each request to five seconds between requests.

  MaxDepth was changed from five to zero, meaning there is no limit to depth indexed by default. [moseley]

- **swishspider location and SpiderDirectory**

  The swishspider program is now installed in $prefix/lib/swish-e by default. This can be changed by the --libexecdir option to configure.

  The SpiderDirectory option now defaults to the value of libexecdir instead of the current directory. [moseley]

- **Added libtool and automake support**

  Replaces the build system with Autotools. Now builds libswish-e as a shared library on systems that support shared libraries. The swish-e binary links against this shared library. Can also build outside the source tree on platforms with GNU make. [moseley]

- **Updates to installation**

  Running "make install" now installs additional files. Files include the swish-e binary, the libswish-e search library, swish-e.h header, documentation files, the swishspider program, and Perl modules used for the example swish.cgi search script. Directories will be created if they do not already exist. Installation directories can be specified at build time.

- **Fixed bug when searching at end of inverted index**

  Swish was not correctly detecting the end of the inverted index when searching a wildcard word that was past the last word in the index. Caught by Frank Heasley. [moseley]

- **Increase sort key length from 50 to 100 characters**

  The setting MAX_SORT_STRING_LEN in *src/config.h* sets the max length used when sorting in swish-e. You may reduce this number to save memory while sorting, or increase it if you have very long properties to sort.

- **Remove &quot; entity from -p output**

  The -p option to print properties was escaping double quotes in properties with the &quot; entity. -x does not do that, so inconsistent. -p no longer converts double quotes. The user should pick a good delimiter with -d or preferably use the -x method for generating output.

- **XML parser and Windows**

  The XML parser was being passed the incorrect buffer length when used on Windows platform causing the parser to abort with an error.

- **Version Numbering**

  SWISH-E versions starting with 2.3.4 use kernel version numbering. Versions are in the form: Major.Minor.Build. Odd minor versions are development. Even minor versions are releases. 2.3.4 would be a development version. 2.4.0 would be a release version. 2.3.20 would be the 20th build of 2.3.

- **Added RPM support**

  RPMs can be built with:

  ```
  ./configure
  make dist
  ```

  Copy the resulting tarball to RPM's SOURCES directory and then run as a superuser:

  ```
  rpmbuild -ba rpm/swish-e.spec
  ```

  You should have swish-e packages in your RPMS/<arch> directory. [augur]

- **Changed default perl binary location**

  Most perl scripts provided with SWISH-E now use /usr/bin/perl by default. Note that some scripts are generated at build time, so those will look in the path for the location of the perl binary.

- **New Feature: MetaNamesRank**

  MetaNamesRank can be used to adjust the ranking for words based on the word's MetaName.

- **New Swish Library API and Perl Module**

  The Swish-e C library interface was rewritten to provide better memory management and better separation of data. Most indexing related code has been removed from the library. A new header file is provided for the API: swish-e.h.

  The Perl module SWISHE was replaced with the SWISH::API module in the Swish-e distribution.

  Previous versions of the SWISHE module will not work with this version of Swish-e.

  If you are using the SWISHE module from a previous version of Swish then you must either rewrite your code to use the new SWISH::API module (highly recommended) or use the replacement SWISHE module. The replacement SWISHE module is a thin interface to the SWISH::API module. It can be downloaded from

```
http://swish-e.org/Download/old/SWISHE-0.03.tar.gz
```

- **NoContents not working with libxml2 parser**

  Corrected problem when using NoContents with binary files and the HTML2 parser.

  Trying to index image file names with:

  ```
  IndexOnly .gif .jpeg
  NoContents .gif .jpeg
  ```

  failed to index the path names because the default parser (HTML2 when libxml2 is linked with swish-e) was not finding any text in the binary files. [moseley]

- **Updates to swish.cgi**

  The example/swish.cgi script can now use the SWISH::API module for searching an index. Combined with mod_perl this module can improve search performance considerably.

  The Perl modules used with the swish.cgi script have all been moved into the SWISH::* namespace. Hence, files in the *modules* directory were moved into the *modules::SWISH* directory.

3.1.10 Version 2.2.3 - December 11, 2002

Multiple -L options were ORing instead of ANDing. Catch by Patrick Mouret. [moseley]

3.1.11 Version 2.2.2 - November 14, 2002

Pass non- text/* files onto indexing code IF there is a FileFilter associated with the *extension* of the URL. Fixes the problem of not being able to index, say, pdf files by using the FileFilter configuation option.

Fixed bug where nulls were stripped when using FileFilter with -S prog. Catch by Greg Fenton. [moseley]

3.1.12 Version 2.2.1 - September 26, 2002

- **NoContents with -S prog**

  Failed to use the correct default parser when using the No-Contents header and libxml2 linked in. [moseley]

- **Add tests for IRIX and sparc machines**

  8-byte alignment in mem_zones is is required for these machine [moseley]

- **Fixed code when removing files**

  Was not correctly removing words from index when parser aborted [jmruiz]

- **Merge segfault**

  Fixed segfault caused by trying to print null dates while merging duplicate files. [moseley]

- **Documentation patches**

  Spelling corrections to the SWISH-CONFIG pod page [Steve Eckert]

- **Configure corrections**

  Fixed a zlib test error that used "==" in a test [Steve Eckert]

- **Updates to VMS build**

  The VMS build was updated [Jean-François PIÉRONNE]

- **MANIFEST corrections**

  Added missing filters and vms build file into MANIFEST [moseley]

3.1.13 Version 2.2 - September 18, 2002

- **Default parser**

  Swish-e will now use the HTML2 (libxml2) parser by default if libxml2 is installed and DefaultContents or IndexContents is not used.

- **Selecting parsers**

  Allow HTML*, XML*, and TXT* to automatically select the libxml2-based parsers if libxml2 is linked with Swish-e, otherwise fallback to the built-in parsers.

- **SwishSpider and Filters**

  Filters (FileFilter directive) did not work correctly when spidering with the -S http method. A new filter system was developed and now filtering of documents (e.g. pdf->html or MSWord->text) is handled by the src/SwishSpider program.

  When indexing with the -S http method only documents of content-type "text/*" are indexed. Other documents must be converted to text by using the filter system.

- **Buffer overflow in xml.c**

Fixed bug in xml.c reported by Rodney Barnett when very long words were indexed. [moseley]

- **configure script updates**

  Updated from _WIN32 checks to feature checks using autoconf [moseley, norris]

- **updates to run on Alpha (Linux 2.4 (Debian 3.0))**

  Fixed a cast error when calling zlib, and the calls to read/write a packed longs to disk. [jmruiz, moseley]

- **COALESCE_BUFFER_MAX_SIZE**

  Some people were seeing the following error:

  ```
  err: Buffer too short in coalesce_word_locations.
  Increase COALESCE_BUFFER_MAX_SIZE in config.h and rebuild.
  ```

  This was due to indexing binary data or files with very large number of words. The best solution is to not index binary data or files with a very large number of words.

  Swish-e will now automatically reallocate the buffer as needed. [jmruiz]

3.1.14 Version 2.2rc1 - August 29, 2002

Many large changes were made internally in the code, some for performance reasons, some for feature changes and additions, and some to prepare for new features in later versions of Swish-e.

- **Documentation!**

  Documentation is now included in the source distribution as .pod (perldoc) files, and as HTML files. In addition, the distribution can now generate PDF, postscript, and unix man pages from the source .pod files. See README for more information.

- **Indexing and searching speed**

  The indexing process has been imporoved. Depending on a number of factors, you may see a significant improvement in indexing speed, especially if upgrading from version 1.x.

  Searching speed has also been improved. Properties are not loaded until results are displayed, and properties are pre-sorted during indexing to speed up sorting results by properties while searching.

- **Properties are written to a sepearte file**

  Swish-e now stores document properties in a separate file. This means there are now two files that make up a Swish-e index. The default files are `index.swish-e` and `index.swish-e.prop`.

This change frees memory while indexing, allowing larger collections to be indexed in memory.

- **Internal data stored as Properties**

  Pre 2.2 some internal data was stored in fixed locations within the index, namely the file name, file size, and title. 2.2 introduced new internal data such as the last modified date, and document summaries. This data is considered *meta data* since it is data about a document.

  Instead of adding new data to the internal structure of the index file, it was decided to use the MetaNames and PropertyNames feature of Swish-e to store this meta information. This allows for new meta data to be added at a later time (e.g. Content-type), and provides an easy and customizable way to print results with the `-p` switch and the new `-x` switch. In addition, search results can now be sorted and limited by properties.

  For example, to sort by the rank and title:

  ```
      swish-e -w foo -s swishrank desc swishtitle asc
  ```

- **The header display has been slightly reorganized.**

  If you are parsing output headers in a program then you may need to adjust your code. There's a new switch <-H> to control the level of header output when searching.

- **Results are now combined when searching more than one index.**

  Swish-e now merges (and sorts) the results from multiple indexes when using `-f` to specify more than one index. This change effects the way maxhits (`-m`) works. Here's a summary of the way it works for the different versions.

  ```
      1.3.2 - MaxHits returns first N results starting from the first index.
              e.g. maxhits=20; 15 hits Index1, 40 hits Index2
              All 15 from Index1 plus first five from Index2 = 20 hits.
  ```

  ```
      2.0.0 - MaxHits returns first N results from each index.
              e.g. Maxhits=20; 15 hits Index1, 40 hits Index2
              All 15 from Index1 plus 15 from Index2.
  ```

  ```
      2.2.0 - Results are merged and first N results are returned.
              e.g. Maxhits=20; 15 hits Index1, 40 hits Index2
              Results are merged from each index and sorted
              (rank is the default sort) and only the first
              20 are returned.
  ```

- **New prog document source indexing method**

You can now use -S prog to use an external program to supply documents to Swish-e. This external program can be used to spider web servers, index databases, or to convert any type of document into html, xml, or text, so it can be indexed by Swish-e. Examples are given in the `prog-bin` directory.

- **The indexing parser was rewritten to be more logical.**

  TranslateCharacters now is done before WordCharacters is checked. For example,

  ```
          WordCharacters abcdefghijklmnopqrstuvwxyz
          TranslateCharacters ñ n
  ```

  Now `El Niño` will be indexed as El Nino (el and nino), even though `ñ` is not listed in WordCharacters.

  Previously, stopwords were checked after stemming and soundex conversions, as well as most of the other word checks (WordCharacters, min/max length and so on). This meant that the stopword list probably didn't work as expected when using stemming.

- **The search parser was rewritten to be more logical**

  The search parser was rewritten to correct a number of logic errors. Swish-e did not differentiate between meta names, Swish-e operators and search words when parsing the query. This meant, for example, that metanames might be broken up by the WordCharacters setting, and that they could be stemmed.

  Swish-e operator characters `"*()=` can now be searched by escaping with a backslash. For example:

  ```
          ./swish-e -w 'this\=odd\)word'
  ```

  will end up searching for the word `this=odd)word`. To search for a backslash character preceed it with a backslash.

  Currently, searching for:

  ```
          ./swish-e -w 'this\*'
  ```

  is the same as a wildcard search. This may be fixed in the future.

  Searching for buzzwords with those characters will still require backslashing. This also may change to allow some un-escaped operator characters, but some will always need to be escaped (e.g. the double-quote phrase character).

- **Quotes and Backslash escapes in strings**

  A bug was fixed in the `parse_line()` function (in *string.c*) where backslashes were not escaping the next character. `parse_line()` is used to parse a string of text into tokens (words). Normally splitting is done at whitespace. You may use quotes (single or double) to define a string (that might include whitespace) as a single parameter. The backslash can also be used to escape the following character when *within* quotes (e.g. to escape an embedded quote character).

```
        ReplaceRules append "foo bar"   <- define "foo bar" as a single word
        ReplaceRules append "foo\"bar"  <- escape the quotes
        ReplaceRules append 'foo"bar'   <- same thing
```

- **Example user.config file removed.**

  Previous versions of Swish-e included a configuration file called `user.config` which contained examples of all directives. This has been replaced by a series of example configuration files located in the `conf` directory. The configuration directives are now described in SWISH-CONFIG.

- **Ports to Win32 and VMS**

  David Norris has included the files required to build Swish-e under Windows. See `src/win32`. A self-extracting Windows version is available from the Download page of the swish-e.org web site.

  Jean-François Piéronne has provided the files required to build Swish-e under OpenVMS. See `src/vms` for more information.

- **String properties are concatenated**

  Multiple *string* properties of the same name in a document are now concatenated into one property. A space character is added between the strings if needed. A warning will be generated if multiple numeric or date properties are found in the same document, and the additional properties will be ignored.

  Previously, properties of the same name were added to the index, but could not be retrieved.

  To do: remove the `next` pointer, and allow user-defined character to place between properties.

- **regex type added to ReplaceRules**

  A more general purpose pattern replacement syntax.

- **New Parsers**

  Swish-e's XML parser was replaced with James Clark's expat XML parser library.

  Swish-e can now use Daniel Veillard's libxml2 library for parsing HTML and XML. This requires installation of the library before building Swish-e. See the INSTALL document for information. libxml2 is not required, but is strongly recommended for parsing HTML over Swish-e's internal HTML parser, and provides more features for both HTML and XML parsing.

- **Support for zlib**

Swish-e can be compiled with zlib. This is useful for compressing large properties. Building Swish-e with zlib is stronly recommended if you use its `StoreDescription` feature.

- **LST type of document no longer supported**

  LST allowed indexing of files that contained multiple documents.

- **Temporary files**

  To improve security Swish-e now uses the `mkstemp(3)` function to create temporary files. Temporary files are used while indexing only. This may result in some portability issues, but the security issues were overriding.

  (Currently this does not apply to the -S http indexing method.)

  `mkstemp` opens the temporary with O_EXCL|O_CREAT flags. This prevents overwriting existing files. In addition, the name of the file created is a lot harder to guess by attackers. The temporary file is created with only owner permissions.

  Please report any portability issues on the Swish-e discussion list.

- **Temporary file locations**

  Swish-e now uses the environment variables `TMPDIR`, `TMP`, and `TEMP` (in that order) to decide where to write temporary files. The configuration setting of TmpDir will be used if none of the environment variables are set. Swish-e uses the current directory otherwise; there is no default temporary directory.

  Since the environment variables override the configuration settings, a warning will be issued if you set TmpDir in the configuration file and there's also an environment variable set.

  Temporary files begin with the letters "swtmp" (which can be changed in *config.h*), followed by two or more letters that indicate the type of temporary file, and some random characters to complete the file name. If indexing is aborted for some reason you may find these temporary files left behind.

- **New Fuzzy indexing method Double Metaphone**

  Based on Lawrence Philips' Metaphone algorithm, add two new methods of creating a fuzzy index (in addition to Stemming and Soundex).

Changes to Configuration File Directives. Please see SWISH-CONFIG for more info.

- **New directives: IndexContents and DefaultContents**

  The IndexContents directive assigns internal Swish-e document parsers to files based on their file type. The DefaultContents directive assigns a parser to be used on file that are not assigned a parser with IndexContents.

- **New directive: UndefinedMetaTags [error|ignore|index|auto]**

  This describes what to do when a meta tag is found in a document that is not listed in the MetaNames directive.

- **New directive: IgnoreTags**

  Will ignore text with the listed tags.

- **New directive: SwishProgParameters *list of words***

  Passes words listed to the external Swish-e program when running with `-S prog` document source method.

- **New directive: ConvertHTMLEntities [yes|no]**

  Controls parsing and conversion of HTML entities.

- **New directive: DontBumpPositionOnMetaTags**

  The word position is now bumped when a new metatag is found -- this is to prevent phrases from matching across meta tags. This directive will disable this behavior for the listed tags.

  This directive works for HTML and XML documents.

- **Changed directive: IndexComments**

  This has been changed such that comments are not indexed by default.

- **Changed directive: IgnoreWords**

  The builtin list of stopwords has been removed. Use of the SwishDefault word will generate a warning, and no stop words will be used. You must now specify a list of stopwords, or specify a file of stopwords.

  A sample file `stopwords.txt` has been included in the *conf/stopwords* directory of the distribution, and can be used by the directive:

  ```
      IgnoreWords File: /path/to/stopwords.txt
  ```

- **Change of the default for IgnoreTotalWordCountWhenRanking**

  The default is now "yes".

- **New directive: Buzzwords**

  Buzzwords are words that should be indexed as-is, without checking for stopwords, word length, WordCharacters, or any other of the word limiting features. This allows indexing of things like `C++` when "+" is not listed in WordCharacters.

Currenly, IgnoreFirstChar and IgnoreLastChar will be stripped before processing Buzzwords.

In the future we may use separate IgnoreFirst/Last settings for buzzwords since, for example, you may wish to index all `+` within Swish-e words, but strip `+` from the start/end of Swish-e words, but not from the buzzword `C++`.

- **New directives: PropertyNamesNumeric PropertyNamesDate**

  Before Swish-e 2.2 all user-defined document properties were stored in the index as strings. PropertyNamesNumeric and PropertyNamesDate tell it that a property should be stored in binary format. This allows for correct sorting of numeric properties.

  Currenly, only integers can be stored, such as a unix timestamp. (Swish-e uses `strtoul` to convert the number to an unsigned long internally.)

  PropertyNamesDate only indicates to Swish-e that a number is a unix timestamp, and to display the property as a formatted time when printing results. Swish does not currently parse date strings; you must provide a unix timestamp.

- **New directive: MetaNameAlias**

  You may now create alias names for MetaNames. This allow you to map or group multiple names to the same MetaName.

- **New directive: PropertyNameAlias**

  Creates aliases for a PropertyName.

- **New directive: PropertyNamesMaxLength**

  Sets the max length of a text property.

- **New directive: HTMLLinksMetaName**

  Defines a metaname to use for indexing href links in HTML documents. Available only with libxml2 parser.

- **New directive: ImageLinksMetaName**

  Defines a metaname to use for indexing src links in <img> tags. Allow you to search image pathnames within HTML pages. Available only with libxml2 parser.

- **New directive: IndexAltTagMetaName**

  Allows indexing of image ALT tags. Only available when using the libxml2 parser.

- **New directive: AbsoluteLinks**

Attempts to convert relative links indexed with HTMLLinksMetaName and ImageLinksMetaName to absolute links. Available only with libxml2 parser.

- **New directive: ExtractPath**

  Allows you to use a regular expression to extract out part of the path of each file and index it with a meta name. For example, this allows searches to be limited to parts of your file tree.

- **New directive: FileMatch**

  FileMatch is similar to FileRules. Where FileRules is used to exclude files and directoires, FileMatch is used to *include* files.

- **New directive: PreSortedIndex**

  Controls which properties are pre-sorted while indexing. All properties are sorted by default.

- **New directive: ParserWarnLevel**

  Sets the level of warning printed when using libxml2.

- **New directive: obeyRobotsNoIndex [yes|NO]**

  When using libxml2 to parse HTML, Swish-e will skip files marked as NOINDEX.

  ```
      <meta name="robots" content="noindex">
  ```

  Also, comments may be used within HTML and XML source docs to block sections of content from indexing:

  ```
          <!-- SwishCommand noindex -->
          <!-- SwishCommand index -->
  ```

  and/or these may be used also:

  ```
          <!-- noindex -->
          <!-- index -->
  ```

- **New directive: UndefinedXMLAttributes**

  This describes how the content of XML attributes should be indexed, if at all. This is similar to UndefinedMetaTags, but is only for XML attributes and when parsed by libxml2. The default is to not index XML attributes.

- **New directive: XMLClassAttributes**

XMLClassAttributes can specify a list of attribute names whose content is combined with the element name to form metanames.

- **New directive: PropCompressionLevel [0-9]**

  If compiled with zlib, Swish-e uses this setting to control the level of compression applied to properties. Properties must be long enough (defined in config.h) to be compressed. Useful for StoreDescription.

- **Experimental directive: IgnoreNumberChars**

  Defines a set of characters. If a word is made of of *only* those characters the word will not be indexed.

- **New directive: FuzzyIndexingMode**

  This configuration directive is used to define the type of "fuzzy" index to create. Currently the options are:

  ```
      None
      Stemming
      Soundex
      Metaphone
      DoubleMetaphone
  ```

Changes to command line arguments. See SWISH-RUN for documentation on these switches.

- **New command line argument -H**

  Controls the level (verbosity) of header information printed with search results.

- **New command line argument -x**

  Provides additional header output and allows for a *format string* to describe what data to print.

- **New command line argument -k**

  Prints words stored in the Swish-e index.

- **New command line argument -N**

  Provides a way to do incremental indexing by comparing last modification dates. You pass `-N` a path to a file and only files newer than the last modified date of that file will be indexed.

- **Removed command line argument -D**

  `-D` no longer dumps the index file data. Use `-T` instead.

- **New command line argument -T**

  `-T` is used for debugging indexing and searching.

- **Enhanced command line argument -d**

  Now `-d` can accept some back-slashed characters to be used as output separators.

- **Enhanced command line argument -P**

  Now -P sets the phrase delimiter character in searches.

- **New command line argument -L**

  Swish-e 2.2 contains an **experimental** feature to limit results by a range of property values. This behavior of this feature may change in the future.

- **Modified command line argument -v**

  Now the argument `-v 0` results in *no* output unless there is an error. This is a bit more handy when indexing with cron.

4 SWISH-CONFIG - Configuration File Directives

4.1 Swish-e CONFIGURATION FILE

What files Swish-e indexes and how they are indexed, and where the index is written can be controlled by a configuration file.

The configuration file is a text file composed of comments, blank lines, and **configuration directives**. The order of the directives is not important. Some directives may be used more than once in the configuration file, while others can only be used once (e.g. additional directives will overwrite preceding directives). Case of the directive is not important -- you may use upper, lower, or mixed case.

Comments are any line that begin with a "#".

```
# This is a comment
```

As of 2.4.3 lines may be continued by placing a backslas as the last character on the line:

```
IgnoreWords \
    am \
    the \
    foo
```

Directives may take more than one parameter. Enclose single parameters that include whitespace in quotes (single or double). Inside of quotes the backslash escapes the next character.

```
ReplaceRules append "foo bar"   <- define "foo bar" as a single parameter
```

If you need to include a quote character in the value either use a backslash to escape it, or enclose it in quotes of the other type.

For example, under unix you can use quotes to include white space in a single parameter. Here, to protect against path names (%p) that might have white space embedded use single quotes (this also protects against shell expansion or metacharacters):

```
FileFilter .foo foofilter "'%p'"  <- parameter passed through the shell in single quotes
FileFilter .foo foofilter '"%p"'  <- windows uses double-quotes
FileFilter .foo foofilter '\'%p\''<- silly example
```

Backslashes also have special meaning in regular expressions.

```
FileFilterMatch pdftotext "'%p' -" /\.pdf$/
```

This says that the dot is a real dot (instead of matching any character). If you place the regular expression in quotes then you must use double-backslashes.

```
    FileFilterMatch pdftotext "'%p' -" "/\\.pdf$/"
```

Swish-e will convert the double backslash into a single backslash before passing the parameter to the regular expression compiler.

Commented example configuration files are included in the *conf* directory of the Swish-e distribution.

Some command line arguments can override directives specified in the configuration file. Please see also the SWISH-RUN for instructions on running Swish-e, and the SWISH-SEARCH page for information and examples on how to search your index.

The configuration file is specified to Swish-e by the `-c` switch. For example,

```
    swish-e -c myconfig.conf
```

You may also split your directives up into different configuration files. This allows you to have a master configuration file used for many different indexes, and smaller configuration files for each separate index. You can specify the different configuration files when running from the command line with the `-c` switch (see SWISH-RUN), or you may include other Configuration file with the **IncludeConfigFile** directive below.

Typically, in a configuration file the directives are grouped together in some logical order -- that is, directives that control the source of the documents would be grouped together first, and directives that control how each document is filtered or its words index in another group of directives. (The directives listed below are grouped in this order).

The configuration file directives are listed below in these groups:

- Administrative Headers Directives -- You may add administrative information to the header of the index file.
- Document Source Directives -- Directives for selecting the source documents and the location of the index file.
- Document Contents Directives -- Directives that control how a document content is indexed.
- Directives for the File Access method only -- These directives are only applicable to the File Access indexing method.
- Directives for the HTTP Access Method Only -- Likewise, these only apply to the HTTP Access method.
- Directives for the prog Access Method Only -- These only apply to the prog Access method.
- Document Filter Directives -- This is a special section that describes using document filters with Swish-e.

4.1.1 Alphabetical Listing of Directives

- AbsoluteLinks [yes|NO]
- BeginCharacters `*string` of characters*
- BumpPositionCounterCharacters *string*
- Buzzwords [*list of buzzwords*|File: path]
- ConvertHTMLEntities [YES|no]
- DefaultContents [TXT|HTML|XML|TXT2|HTML2|XML2|TXT*|HTML*|XML*]
- Delay *seconds*
- DontBumpPositionOnEndTags `*list` of names*
- DontBumpPositionOnStartTags `*list` of names*
- EnableAltSearchSyntax [yes|NO]
- EndCharacters `*string` of characters*
- EquivalentServer `*server` alias*
- ExtractPath *metaname* [replace|remove|prepend|append|regex]
- FileFilter *suffix* *program* [options]
- FileFilterMatch *program* *options* *regex* [*regex* ...]
- FileInfoCompression [yes|NO]
- FileMatch [contains|is|regex] `*regular` expression*
- FileRules [contains|is|regex] `*regular` expression*
- FuzzyIndexingMode [NONE|Stemming|Soundex|Metaphone|DoubleMetaphone]
- FollowSymLinks [yes|NO]
- HTMLLinksMetaName *metaname*
- IgnoreFirstChar `*string` of characters*
- IgnoreLastChar `*string` of characters*

- IgnoreLimit *`integer` integer*
- IgnoreMetaTags *`list` of names*
- IgnoreNumberChars *`list` of characters*
- IgnoreTotalWordCountWhenRanking [YES|no]
- IgnoreWords [*list of stop words*|File: path]
- ImageLinksMetaName *metaname*
- IncludeConfigFile
- IndexAdmin *text*
- IndexAltTagMetaName *tagname*|as-text
- IndexComments [yes|NO]
- IndexContents [TXT|HTML|XML|TXT2|HTML2|XML2|TXT*|HTML*|XML*] *`file` extensions*
- IndexDescription *text*
- IndexDir [URL|directories or files]
- IndexFile *path*
- IndexName *text*
- IndexOnly *`list` of file suffixes*
- IndexPointer *text*
- IndexReport [0|1|2|3]
- MaxDepth *integer*
- MaxWordLimit *integer*
- MetaNameAlias *`meta` name* *`list` of aliases*
- MetaNames *`list` of names*
- MinWordLimit *integer*
- NoContents *`list` of file suffixes*

- obeyRobotsNoIndex [yes|NO]
- ParserWarnLevel [0|1|2|3]
- PreSortedIndex *`list` of property names*
- PropCompressionLevel [0-9]
- PropertyNameAlias *`property` name* *`list` of aliases*
- PropertyNames *`list` of meta names*
- PropertyNamesCompareCase *`list` of meta names*
- PropertyNamesIgnoreCase *`list` of meta names*
- PropertyNamesNoStripChars *`list` of meta names*
- PropertyNamesDate *`list` of meta names*
- PropertyNamesNumeric *`list` of meta names*
- PropertyNamesMaxLength integer *`list` of meta names*
- PropertyNamesSortKeyLength integer *`list` of meta names*
- ReplaceRules [replace|remove|prepend|append|regex]
- ResultExtFormatName name -x format string
- SpiderDirectory *path*
- StoreDescription [XML <tag>|HTML <meta>|TXT size]
- "SwishProgParameters *`list` of parameters*
- SwishSearchDefaultRule [<AND-WORD>|<or-word>]
- SwishSearchOperators <and-word> <or-word> <not-word>
- TmpDir *path*
- TranslateCharacters [*string1 string2*|:ascii7:]
- TruncateDocSize *`number` of characters*
- UndefinedMetaTags [error|ignore|INDEX|auto]

- UndefinedXMLAttributes [DISABLE|error|ignore|index|auto]
- UseStemming [yes|NO]
- UseSoundex [yes|NO]
- UseWords [*list of words*|File: path]
- WordCharacters `*string` of characters*
- XMLClassAttributes `*list` of XML attribute names*

4.1.2 Directives that Control Swish

These configuration directives control the general behavior of Swish-e.

IncludeConfigFile *path to config file*

This directive can be used to include configuration directives located in another file.

```
        IncludeConfigFile /usr/local/swish/conf/site_config.config
```

IndexReport [0|1|2|3]

This is how detailed you want reporting while indexing. You can specify numbers 0 to 3. 0 is totally silent, 3 is the most verbose. The default is 1.

This may be overridden from the command line via the `-v` switch (see SWISH-RUN).

ParserWarnLevel [0|1|2|3]

Sets the error level when using the libxml2 parser for XML and HTML. libxml2 will point out structural errors in your documents.

```
        0 = no report
        1 = fatal errors
        2 = errors
        3 = warnings
```

The exception to this is UTF-8 to Latin-1 conversion errors are reported at level 1. This is because words may be indexed incorrectly in these cases.

Note that unlike other errors generated by Swish-e, these errors are sent to stderr.

IndexFile *path*

Index file specifies the location of the generated index file. If not specified, Swish-e will create the file *index.swish-e* in the current directory.

```
IndexFile /usr/local/swish/site.index
```

obeyRobotsNoIndex [yes|NO]

When enabled, Swish-e will not index any HTML file that contains:

```
<meta name="robots" content="noindex">
```

The default is to ignore these meta tags and index the document. This tag is described at http://www.robotstxt.org/wc/exclusion.html.

Note: This feature is only available with the libxml2 HTML parser.

Also, if you are using the libxml2 parser (HTML2 and XML2) then you can use the following comments in your documents to prevent indexing:

```
    <!-- SwishCommand noindex -->
    <!-- SwishCommand index -->
```

and/or these may be used also:

```
    <!-- noindex -->
    <!-- index -->
```

For example, these are very helpful to prevent indexing of common headers, footers, and menus.

NOTE: This following items are currently not available. These items require Swish-e to parse the configuration file while searching.

EnableAltSearchSyntax [yes|NO]

NOTE: This following item is currently not available.

Enable alternate search syntax. Allows the usage of a basic "Altavista(c)", "Lycos(c)", etc. like search syntax. This means a search query can contain "+" and "-" as syntax parameter.

Example:

```
swish-e -w "+word1 +word2 -word3  word4 word5"
"+"  = following word has to be in all found documents
"-"  = following word may not be in any document found
" "  = following word will be searched in documents
```

SwishSearchOperators <and-word> <or-word> <not-word>

NOTE: This following item is currently not available.

Using this config directive you can change the boolean search operators of Swish-e, e.g. to adapt these to your language. The default is: AND OR NOT

Example (german):

```
SwishSearchOperators   UND  ODER  NICHT
```

SwishSearchDefaultRule [<AND-WORD>|<or-word>]

NOTE: This following item is currently not available.

SwishSearchDefaultRule defines the default Boolean operator to use if none is specified between words or phrases. The default is `AND`.

The word you specify must match one of the available SwishSearchOperators.

Example:

```
SwishSearchOperators   UND  ODER  NICHT
# Make it act like a web search engine
SwishSearchDefaultRule ODER
```

ResultExtFormatName name -x format string

NOTE: This following item is currently not available.

The output of Swish-e can be defined by specifying a format string with the `-x` command line argument. Using ResultExtFormatName you can assign a predefined format string to a name.

Examples:

```
ResultExtFormatName  moreinfo   "%c|%r|%t|%p|<author>|<publishyear>\n"
```

Then when searching you can specify the format string's name

```
swish-e   ...  -x moreinfo  ...
```

See the `-x` switch in SWISH-RUN for more information about output formats.

4.1.3 Administrative Headers Directives

Swish-e stores configuration information in the header of the index file. This information can be retrieved while searching or by functions in the Swish-e C library. There are a number of fields available for your own use. None of these fields are required:

IndexName *text*

IndexDescription *text*

IndexPointer *text*

IndexAdmin *text*

These variables specify information that goes into index files to help users and administrators. IndexName should be the name of your index, like a book title. IndexDescription is a short description of the index or a URL pointing to a more full description. IndexPointer should be a pointer to the original information, most likely a URL. IndexAdmin should be the name of the index maintainer and can include name and email information. These values should not be more than 70 or so characters and should be contained in quotes. Note that the automatically generated date in index files is in D/M/Y and 24-hour format.

Examples:

```
    IndexName "Linux Documentation"
    IndexDescription "This is an index of /usr/doc on our Linux machine."
    IndexPointer http://localhost/swish/linux/index.html
    IndexAdmin webmaster
```

4.1.4 Document Source Directives

These directives control *what* documents are indexed and *how* they are accessed. See also Directives for the File Access method only and Directives for the HTTP Access Method Only for directives that are specific to those access methods.

IndexDir [directories or files|URL|external program]

IndexDir defines the source of the documents for Swish-e. Swish-e currently supports three file access methods: **File system**, **HTTP** (also called **spidering**), and **prog** for reading files from an external program.

The `-S` command line argument is used to select the file access method.

```
    swish-e -c swish.config -S fs    - file system
    swish-e -c swish.config -S http  - internal http spider
    swish-e -c swish.config -S prog  - external program of any type
```

For the **fs** method of access **IndexDir** is a space-separated list of files and directories to index. Use a forward slash as the path separator in MS Windows.

For the **http** method the **IndexDir** setting is a list of space-separated URLs.

For the **prog** method the **IndexDir** setting is a list of space-separated programs to run (which generate documents for swish to index).

You may specify more than one **IndexDir** directive.

Any sub-directories of any listed directory will also be indexed.

Note: While *processing* directories, Swish-e will ignore any files or directories that begin with a dot ("."). You may index files or directories that begin with a dot by specifying their name with IndexDir or `-i`.

Examples:

```
    # Index this directory an any subdirectories
    IndexDir /usr/local/home/http
```

```
    # Index the docs directory in current directory
    IndexDir ./docs
```

```
    # Index these files in the current directory
    IndexDir ./index.html ./page1.html ./page2.html
    # and index this directory, too
    IndexDir ../public_html
```

For the **HTTP** method of access specify the URL's from which you want the spidering to begin.

Example:

```
    IndexDir http://www.my-site.com/index.html
    IndexDir http://localhost/index.html
```

Obviously, using the **HTTP** method to index is **much** slower than indexing local files. Be well aware that some sites do not appreciate spidering and may block your IP address. You may wish to contact the remote site before spidering their web site. More information about spidering can be found in Directives for the HTTP Access Method Only below.

For the prog method of access **IndexDir** specifies the path to the `program(s)` to execute. The external program must correctly format the documents being passed back to Swish-e. Examples of external programs are provided in the *prog-bin* directory.

```
IndexDir ./myprogram.pl
```

See prog for details.

Note: Not all directives work with all methods.

NoContents *list of file suffixes*

Files with these suffixes will **not** have their contents indexed, but will have their path name (file name) indexed instead.

If the file's type is HTML or HTML2 (as set by IndexContents or DefaultContents) then the file will be parsed for a HTML title and that title will be indexed. Note that you must set the file's type with IndexContents or DefaultContents: `.html` and `.htm` are NOT type HTML by default. For example:

```
IndexContents HTML* .htm .html
```

If a title is found, it will still be checked for `FileRules title`, and the file will be skipped if a match is found. See FileRules.

If the file's type is not HTML, or it is HTML and no title is found, then the file's path will be indexed.

For example, this will allow searching by image file name.

```
NoContents .gif .xbm .au .mov .mpg .pdf .ps
```

Note: Using this directive will **not** cause files with those suffixes to be indexed. That is, if you use IndexOnly to limit the types of files that are indexed, then you must specify in IndexOnly the same suffixes listed in NoContents.

This does **not** work:

```
# Wrong!
IndexOnly .htm .html
NoContents .gif .xbm .au .mov .mpg .pdf .ps
```

A `-S prog` program may set the No-Contents: header to enable this feature for a specific document (although it would be smarter for the `-S prog` program to simply only send the pathname or title to be indexed.

ReplaceRules [replace|remove|prepend|append|regex]

ReplaceRules allows you to make changes to file pathnames before they're indexed. These changed file names or URLs will be returned in search results.

For example, you may index your files locally (with the File system indexing method), yet return a URL in search results. This directive can be used to map the file names to their respective URLs on your web server.

There are five operations you can specify: **replace**, **append**, **remove**, **prepend**, and **regex** They will parse the pathname in the order you've typed these commands.

This directive uses C library regex.h regular expressions.

```
replace "the string you want replaced" "what to change it to"
remove "a string to remove"
prepend "a string to add before the result"
append "a string to add after the result"
regex  "/search string/replace string/options"
```

Remember, quotes are needed if an expression contains white space, and backslashes have special meaning.

Regex is an Extended Regular Expression. The first character found is the delimiter (but it's not smart enough to use matched chars such as [], (), and {}).

The **replace** string may use substitution variables:

```
 $0      the entire matched (sub)string
 $1-$9   returns patterns captured in "(" ")" pairs
 $‘      the string before the matched pattern
 $’      the string after the matched pattern
```

The **options** change the behavior of expression:

```
 i       ignore the case when matching
 g       repeat the substitution for the entire pattern
```

Examples:

```
 ReplaceRules replace testdir/ anotherdir/
 ReplaceRules replace [a-z_0-9]*_m.*\.html index.html
```

```
 ReplaceRules remove testdir/
```

```
 ReplaceRules prepend http://localhost/
 ReplaceRules append .html
```

```
ReplaceRules regex  !^/web/(.+)/!http://$1.domain.com/!
replaces a file path:
    /web/search/foo/index.html
with
    http://search.domain.com/foo/index.html
```

```
ReplaceRules regex  #^#http://localhost/www#
ReplaceRules prepend http://localhost/www  (same thing)
```

```
# Remove all extensions from C source files
ReplaceRules remove .c     # ERROR! That "." is *any char*
ReplaceRules remove \.c    # much better...
```

```
ReplaceRules remove "\\.c" # if in quotes you need double-backslash!
ReplaceRules remove "\.c"  # ERROR! "\." -> "." and is *any char*
```

IndexContents [TXT|HTML|XML|TXT2|HTML2|XML2|TXT*|HTML*|XML*] *file extensions*

The IndexContents directive assigns one of Swish-e's document parsers to a document, based on the its extension. Swish-e currently knows how to parse TXT, HTML, and XML documents.

The XML2, HTML2, and TXT2 parsers are currently only available when Swish-e is configured to use libxml2.

You may use XML*, HTML*, and TXT* to select the parser automatically. If libxml2 is installed then it will be used to parse the content. Otherwise, Swish-e's internal parsers will be used.

Documents that are not assigned a parser with IndexContents will, by default, use the HTML2 parser if libxml2 is installed, otherwise will use Swish-e's internal HTML parser. The DefaultContents directive may be used to assign a parser to documents that do not match a file extension defined with the IndexContents directive.

Example:

```
IndexContents HTML* .htm .html .shtml
IndexContents TXT*  .txt .log .text
IndexContents XML*  .xml
```

HTML* is the default type for all files, unless otherwise specified (and this default can be changed by the **DefaultContents** directive. Swish-e parses titles from HTML files, if available, and keeps track of the context of the text for context searching (see `-t` in SWISH-RUN).

If using filters (with the FileFilter directive) to convert documents you should include those extensions, too. For example, if using a filter to convert .pdf to .html, you need to tell Swish-e that .pdf should be indexed by the internal HTML parser:

```
        FileFilter  .pdf   pdf2html
        IndexContent  HTML  .pdf
```

See also Document Filter Directives.

Note: Some of this may be changed in the future to use content-types instead of file extensions. See SWISH-3.0

DefaultContents [TXT|HTML|XML|TXT2|HTML2|XML2|TXT*|HTML*|XML*]

This sets the default parser for documents that are not specified in **IndexContents**. If not specified the default is HTML.

The XML2, HTML2, and TXT2 parsers are currently only available when Swish-e is configured to use libxml2.

You may use XML*, HTML*, and TXT* to select the parser automatically. If libxml2 is installed then it will be used to parse the content. Otherwise, Swish-e's internal parsers will be used.

Example:

```
        DefaultContents HTML
```

The DefaultContents directive *should* be used when spidering, as HTML files may be returned without a file extension (such as when requesting a directory and the default index.html is returned).

FileInfoCompression [yes|NO]

** This directive is currently not supported **

Setting **FileInfoCompression** to `yes` will compress the index file to save disk space. This may result in longer indexing times. The default is `no`.

Also see the `-e` switch in SWISH-RUN for saving RAM during indexing.

4.1.5 Document Contents Directives

These directives control what information is extracted from your source documents, and how that information is made available during searching.

ConvertHTMLEntities [YES|no]

ASCII *entities* can be converted automatically while indexing documents of type HTML (not for HTML2). For performance reasons you may wish to set this to `no` if your documents do not contain HTML entities. The default is `yes`.

If ConvertHTMLEntities is set `no` the entities will be indexed without conversion.

NOTE: Entities within XML files and files parsed with libxml2 (HTML2) are converted regardless of this setting.

MetaNames *list of names*

META names are a way to define "fields" in your XML and HTML documents. You can use the META names in your queries to limit the search to just the words contained in that META name of your document. For example, you might have a META tagged field in your documents called `subjects` and then you can search your documents for the word "foo" but only return documents where "foo" is within the `subjects` META tag.

```
swish-e -w subjects=foo
```

(See also the `-t` switch in SWISH-RUN for information about *context* searching in HTML documents.)

The **MetaNames** directive is a space separated list. For example:

```
MetaNames meta1 meta2 keywords subjects
```

You may also use UndefinedMetaTags to specify automatic extraction of meta names from your HTML and XML documents, and also to ignore indexing content of meta tags.

META tags can have two formats in your **HTML** source documents:

```
<META NAME="meta1" CONTENT="some content">
```

and (if using the HTML2/libxml2 parser)

```
<meta1>
    some content
</meta1>
```

But this second version is invalid HTML, and will generate a warning if ParserWarningLevel is set (libxml2 only).

And in **XML** documents, use the format:

```
<meta1>
    Some Content
</meta1>
```

Then you can limit your search to just META **meta1** like this:

```
swish-e -w 'meta1=(apples or oranges)'
```

You may nest the XML and the start/end tag versions:

```
<keywords>
    <tag1>
        some content
    </tag1>
    <tag2>
        some other content
    </tag2>
<keywords>
```

Then you can search in both tag2 and tag2 with:

```
swish-e -w 'keywords=(query words)'
```

Swish-e indexes all text as some metaname. The default is `swishdefault`, so these two queries are the same:

```
swish-e -w foo
swish-e -w swishdefault=foo
```

When indexing HTML Swish-e indexes the HTML title as default text, so when searching Swish-e will find matches in both the HTML body and the HTML title. Swish also, by default, indexes content of meta tags. So:

```
swish-e -w foo
```

will find "foo" in the body, the title, or any meta tags.

Currently, there's no way to prevent Swish-e from indexing the title contents along with the body contents, but see UndefinedMetaTags for how to control the indexing of meta tags.

If you would like to search just the title text, you may use:

```
MetaNames swishtitle
```

This will index the title text separately under the built-in swish internal meta name "swishtitle". You may then search like

```
swish-e -w foo  -- search for "foo" in title, body (and undefined meta tags)
swish-e -w swishtitle=foo -- search for "foo" in title only
```

In addition to swishtitle, you can limit searches to documents' path with:

```
MetaNames swishdocpath
```

Then to search for "foo" but also limit searches to documents that include "manual" or "tutorial" in their path:

```
swish-e -w foo swishdocpath=(manual or tutorial)
```

See also ExtractPath.

MetaNameAlias *meta name* *list of aliases*

MetaNameAlias assigns aliases for a meta name. For example, if your documents contain meta tags "description", "summary", and "overview" that all give a summary of your documents you could do this:

```
MetaNames summary
MetaNameAlias summary description overview
```

Then all three tags will get indexed as meta tag "summary". You can then search all the fields as:

```
-w summary=foo
```

The Alias work at search time, too. So these will also limit the search to the "summary" meta name.

```
-w description=foo
-w overview=foo
```

MetaNamesRank integer *list of meta names*

You can assign a bias to metanames that will affect how ranking is calculated. The range of values is from -10 to +10, with zero being no bias.

```
MetaNamesRank 4 subject
MetaNamesRank 3 swishdefault
MetaNamesRank 2 author publisher
MetaNamesRank -5 wrongwords
```

This feature is still considered experimental. If you use it, please send feedback to the discussion list.

HTMLLinksMetaName *metaname*

Allows indexing of HTML links. Normally, HTML links (href tags) are not indexed by Swish-e. This directive defines a metaname, and links will be indexed under this meta name.

Example:

```
    HTMLLinksMetaName links
```

Now, to limit searches to files with a link to "home.html" do this:

```
    -w links='"home.html"'
```

The double quotes force a phrase search.

To make Swish-e index links as normal text, you may use:

```
    HTMLLinksMetaName swishdefault
```

This feature is only available with the libxml2 HTML parser.

ImageLinksMetaName *metaname*

Allows indexing of image links under a metaname. Normally, image URLs are not indexed.

Example:

```
    ImagesLinksMetaName images
```

Now, if you would like to find pages that include a nice image of a beach:

```
    -w images='beach'
```

To make Swish-e index links as normal text, you may use:

```
    ImageLinksMetaName swishdefault
```

This feature is only available with the libxml2 HTML parser.

IndexAltTagMetaName *tagname*|as-text

Allows indexing of images <IMG> ALT tag text. Specify either a tag name which will be used as a metaname, or the special text "as-text" which says to index the ALT text as if it were plain text at the current location.

For example, by specifying a tag name:

```
 IndexAltTagMetaName bar
```

would make this markup:

```
  <foo>
      <img src="/someimage.png" alt="Alt text here">
  </foo>
```

appear like

```
  <foo>
      <bar>Alt text here</bar>
  </foo>
```

Then the normal rules (MetaNames and PropertyNames) apply to how that text is indexed.

If you use the special tag "as-text" then

```
  <foo>
      <img src="/someimage.png" alt="Alt text here">
  </foo>
```

simply becomes

```
  <foo>
      Alt text here
  </foo>
```

This feature is only available when using the libxml2 parser (HTML2 and XML2).

AbsoluteLinks [yes|NO]

If this is set true then Swish-e will attempt to convert relative URIs extracted from HTML documents for use with HTMLLinksMetaName and ImageLinksMetaName into absolute URIs. Swish-e will use any <BASE> tag found in the document, otherwise it will use the file’s pathname. The pathname used will be the pathname *after* ReplaceRules has been applied to the document’s pathname.

For example, say you wish to index image links under the metaname "images".

```
  ImageLinksMetaName images
```

If an image is located in http://localhost/vacations/france/index.html and AbsoluteLinks is set to no, then a image within that document:

```
        <img src="beach.jpeg">
```

will only index "beach.jpeg".

But, if you want more detail when searching, you can enable AbsoluteLinks and Swish-e will index "http://localhost/vacations/france/beach.jpeg". You can then look for images of beaches, but only in France:

```
        -w images=(beach and france)
```

This also means you can search for any images within France:

```
        -w images=(france)
```

This feature is only available with the libxml2 HTML parser.

UndefinedMetaTags [error|ignore|INDEX|auto]

This directive defines the behavior of Swish-e during indexing when a meta name is found but is **not** listed in **MetaNames**. There are four choices:

UndefinedMetaTags error

If a meta name is found that is not listed in **MetaNames** then indexing will be halted and an error reported.

UndefinedMetaTags ignore

The contents of the meta tag are ignored and **not** indexed unless a metaname has been defined with the MetaNames directive.

UndefinedMetaTags index

The contents of the meta tag are indexed, but placed in the main index unless there’s an enclosing metatag already in force. This is the default.

UndefinedMetaTags auto

This method create meta tags automatically for HTML meta names and XML elements. Using this is the same as specifying all the meta names explicitly in a **MetaNames** directive.

UndefinedXMLAttributes [DISABLE|error|ignore|index|auto]

This is similar to UndefinedMetaTags, but only applies to XML documents (parsed with libxml2). This allows indexing of attribute content, and provides a way to index the content under a metaname. For example, UndefinedXMLAttributes can make

```
<person age="23">
      John Doe
</person>
```

look like the following to swish:

```
<person>
    <person.age>
        23
    </person.age>
    John Doe
</person>
```

What happens to the text "23" will depend on the setting of UndefinedXMLAttributes:

disable

XML attributes are not parsed and not indexed. This is the default.

error

If the concatenated meta name (e.g. person.age) is not listed in **MetaNames** then indexing will be halted and an error reported.

ignore

The contents of the meta tag are ignored and **not** indexed unless a metaname has been defined with the MetaNames directive.

index

The contents of the meta tag are indexed, but placed in the main index unless there's an enclosing metatag already in force.

auto

This method will create meta tags from the combined element and attributes (and XML Class name) This options should be used with caution as it can generate a lot of metaname entries.

See also the example below `XMLClassAttribues`.

XMLClassAttributes *list of XML attribute names*

Combines an XML class name with the element name to make up a metaname. For example:

```
XMLClassAttributes class
```

```
<person class="first">
    John
</person>
<person class="last">
    Doe
</person>
```

Will appear to Swish-e as:

```
<person>
    <person.first>
    John
    </person.first>
</person>
<person>
    <person.last>
    Doe
    </person.last>
</person>
```

How the data is indexed depends on MetaNames and UndefinedMetaTags.

Here's an example using the following configuration which combines the two directives XMLClassAttributes and UndefinedXMLAttributes.

```
XMLClassAttributes class
UndefinedMetaTags auto
UndefinedXMLAttributes auto
IndexContents XML2 .xml
```

The source XML file looks like:

```
<xml> <person class="student" phone="555-1212" age="102"> John </person>
<person greeting="howdy">Bill</person> </xml>
```

Swish-e parses as:

```
./swish-e -c 2 -i 1.xml -T parsed_tags  parsed_text  -v 0
Indexing Data Source: "File-System"
```

```
<xml> (MetaName)
```

```
        <person> (MetaName)
            <person.student> (MetaName)
                <person.student.phone> (MetaName)
                    555-1212
                </person.student.phone>
                <person.student.age> (MetaName)
                    102
                </person.student.age>
                John
        </person>
```

```
        <person> (MetaName)
            <person.greeting> (MetaName)
                howdy
            </person.greeting>
            Bill
        </person>
```

```
    </xml>
    Indexing done!
```

One thing to note is that the first <person> block finds a class name "student" so all metanames that are created from attributes use the combined name "person.student". The second <person> block doesn't contain a "class" so, the attribute name is combined directly with the element name (e.g. "person.greeting").

ExtractPath *metaname* [replace|remove|prepend|append|regex]

This directive can be used to index extracted parts of a document's path. A common use would be to limit searches to specific areas of your file tree.

The extracted string will be indexed under the specified meta name.

See ReplaceRules for a description of the various pattern replacement methods, but you will use the *regex* method.

For example, say your file system (or web tree) was organized into departments:

```
    /web/sales/foo...
    /web/parts/foo...
    /web/accounting/foo...
```

And you wanted a way to limit searches to just documents under "sales".

```
    ExtractPath department regex !^/web/([^/]+)/.*$!$1!
```

Which says, extract out the department name (as substring $1) and index it as meta name `department`. Then to limit a search to the sales department:

```
    swish-e -w foo AND department=sales
```

Note that the `regex` method uses a substitution pattern, so to index only a sub-string match the *entire* document path in the regular expression, as shown above. Otherwise any part that is not matched will end up in the substitution pattern.

See the ExtractPathDefault option for a way to set a value if not patterns match.

Although unlikely, you may use more than one ExtractPath directive. More than one directive of the *same* meta name will operate successively (in order listed in the configuration file) on the path. This allows you to use regular expressions on the results of the previous pattern substitution (as if piping the output from one expression to the patter of the next).

```
    ExtractPath foo regex !^(...).+$!$1!
    ExtractPath foo regex !^.+(.)$!$1!
```

So, the third letter is indexed as meta name "foo" if both patterns match.

```
    ExtractPath foo regex !^X(...).+$!$1!
    ExtractPath foo regex !^.+(.)$!$1!
```

Now (not the "X"), if the first pattern doesn’t match, the last character of the path name is indexed. You must be clear on this behavior if you are using more than one ExtractPath directive with the same metaname.

The document path operated on is the real path swish used to access the document. That is, the ReplaceRules directive has no effect on the path used with ExtractPath.

The full path is used for each meta name if more than one ExtractPath directive is used. That is, changes to the path used in `ExtractPath foo` do not affect the path used by `ExtractPath bar`.

ExtractPathDefault *metaname* default_value

This can be used with ExtractPath to set a default string to index under the given metaname if none of the ExtractPath patterns match.

For example, say your want to index each document with a metaname "department" based on the following path examples:

```
    /web/sales/foo...
    /web/parts/foo...
    /web/accounting/foo...
```

But you are also indexing documents that do not follow that pattern and you want to search those separately, too.

```
ExtractPath department regex !^/web/([^/]+)/.*$!$1!
ExtractPathDefault department other
```

Now, you may search like this:

```
-w foo department=(sales)      - limit searches to the sales documents
-w foo department=(parts)      - limit searches to the parts documents
-w foo department=(accounting) - limit searches to the accounting documents
-w foo department=(other)      - everything but sales, parts, and accounting.
```

This basically is a shortcut for:

```
-w foo not department=(sales or parts or accounting)
```

but you don't need to keep track of what was extracted.

PropertyNames *list of meta names*

PropertyNamesCompareCase *list of meta names*

PropertyNamesIgnoreCase *list of meta names*

Swish-e allows you to specify certain META tags that can be used as **document properties**. The contents of any META tag that has been identified as a document property can be returned as part of the search results along with the rank, file name, title, and document size (see the `-p` and `-x` switches in SWISH-RUN).

Properties are useful for returning additional data from documents in search results -- this saves the effort of reading and parsing the source files while reading Swish-e search results, and is especially useful when the source documents are no longer available or slow to access (e.g. over http).

Another feature of properties is that Swish-e can use the PropertyNames for sorting the search results (see the `-s` switch).

```
PropertyNames author subjects
```

Two variations are available. PropertyNamesCompareCase and PropertyNamesIgnoreCase. These tell Swish-e to either ignore or compare case when sorting results. The default for PropertyNames is to ignore the case.

```
PropertyNamesIgnoreCase subject
PropertyNamesCompareCase keyword
```

The defaults for "internal" properties are:

```
    swishtitle          --  ignore the case
    swishdocpath        --  compare case
    swishdescription    --  compare case
```

These can be overridden with PropertyNamesCompareCase and PropertyNamesIgnoreCase.

```
    PropertyNamesCompareCase swishtitle
```

Use of PropertyNames will increase the size of your index files, sometimes significantly. Properties will be compressed if Swish-e is compiled with zlib as described in the INSTALL manual page.

If Swish-e finds more than one property of the same name in a document the property's contents will be concatinated for strings, and a warning issues for numeric (or date) properties.

PropertyNamesNoStripChars

PropertyNamesNoStripChars specifies that the listed properties should not have strings of low ASCII characters replaced with a space character. Properties will be stored as found in the document.

When printing properties with the swish-e binary newlines are replaced with a space character. Use the swish-e library (or SWISH::API perl module) to fetch properties without newlines replaced.

PropertyNamesNumeric

This directive is similar to PropertyNames, but it flags the property as being a string of digits (integer value) that will be stored as binary data instead of a string. This allows sorting with `-s` and limiting with `-L` to sort and limit the property correctly.

Swish-e uses `strtoul(3)` to convert the string into an unsigned long integer. Therefore, only positive integers can be stored.

Future versions of Swish-e may be able to store different property types (such as negative integers and real numbers). This directive may change in future releases of Swish.

PropertyNamesDate

This directive is exactly like PropertyNamesNumeric, but it also flags the number as a machine timestamp (seconds since Epoch), and will print a formatted date when returning this property. See `-x` in SWISH-RUN.

Swish-e will not parse dates when indexing; you must use a timestamp.

PropertyNameAlias *property name* *list of aliases*

This allows aliases for a property name. For example, if you are indexing HTML files, plus XML files that are written in English, German, and Spanish and thus use the tags "title", "titel", and "título" you can use:

```
PropertyNameAlias swishtitle title titel título titulo
```

Note that "swishtitle" is the built-in property used to store the title of a document, and therefore you do not need to specify it as a PropertyName before use.

PropertyNamesMaxLength integer *list of meta names*

This option will set the max length of the text stored in a property. You must specify a number between 0 and the max integer size on your platform, and a list of properties. The properties specified must not be aliases.

If any of the property names do not exist they will be created (e.g. you do not need to define the property with PropertyNames first).

In general, this feature will only be useful when parsing HTML or XML with the libxml2 parser.

For example:

```
PropertyNamesMaxLength 1000 swishdescription
PropertyNameAlias swishdescription body
```

Is somewhat like

```
StoreDescription HTML <body> 1000
StoreDescription XML <body> 1000
StoreDescription HTML2 <body> 1000
StoreDescription XML2 <body> 1000
```

but StoreDescription allows setting the tag for each parser type.

```
PropertyNamesMaxLength 1000 headings
PropertyNameAlias headings h1 h2 h3 h4
```

collects all the heading text into a single property called "headings", not to exceed 1000 characters.

PropertyNamesSortKeyLength integer *list of meta names*

Sets the length of the string used when sorting. The default is 100 characters. The -T metanames debugging option will list the current values for an index.

This setting is used when sorting during indexing, and perhaps when sorting while searching. It also effects the order when limiting to a range of values with the -L option.

PreSortedIndex *list of property names*

By default Swish-e generates presorted tables while indexing for each property name. This allows faster sorting when generating results. On large document collections this presorting may add to the indexing time, and also adds to the total size of the index. This directive can be used to customize exactly which properties will be presorted.

If PreSortedIndex it is *not* present in the config file (default action), all the properties will be presorted at indexing time. If it is present without any parameter, no properties will be presorted. Otherwise, only the property names specified will be presorted.

For example, if you only wish to sort results by a property called `title`:

```
PropertyNames title age time
PreSortedIndex  title
```

StoreDescription [XML <tag> size|HTML <meta> size|TXT size]

StoreDescription allows you to store a document description in the index file. This description can be returned in your search results when the `-x` switch is used to include the *swishdescription* for extended results, or by using `-p swishdescription`.

The document type (XML, HTML and TXT) must match the document type currently being indexed as set by IndexContents or DefaultContents. See those directives for possible values. A common problem is using StoreDescription yet not setting the document's type with IndexContents or DefaultContents. Another problem is different types:

```
IndexContents HTML2 .html
StoreDescription HTML <body>
```

Then .html documents are assigned a type of HTML2 (and parsed by the libxml2 parser), but the description will not be stored since it is type HTML instead of HTML2.

For text documents you specify the type TXT (or TXT2 or TXT*) and the number of *characters* to capture.

```
StoreDescription TXT 20
```

The above stores only the first twenty characters from the text file in the Swish-e index file.

For HTML, and XML file types, specify the tag to use for the description, and optionally the number of characters to capture. If not specified will capture the entire contents of the tag.

```
StoreDescription HTML <body> 20000
StoreDescription XML  <desc> 40
```

Again, note that documents must be assigned a document type with IndexContents or DefaultContents to use this feature.

Swish-e will compress the descriptions (or any other large property) if compiled to use zlib (see INSTALL). This is recommended when using StoreDescription and a large number of documents. Compression of 30% to 50% is not uncommon with HTML files.

PropCompressionLevel [0-9]

This directive sets the compression level used when storing properties to disk. A setting of zero is no compression, and a setting of nine is the most compression.

The default depends on the default setting compiled with zlib, but is typically six.

This option is useful when using StoreDescription to store a large amount text in properties (or if using PropertyNames with large property sizes).

Properties must be over a value defined in *config.h* (100 is the default) before compression will be attempted. Swish-e will never store the results of the compression if the compressed data is larger than the original data.

This option is only available when Swish-e is compiled with zlib support.

TruncateDocSize *number of characters*

TruncateDocSize limits the size of a document while indexing documents and/or using filters. This config directive truncates the numbers of read bytes of a document to the specified size. This means: if a document is larger, read only the specified numbers of bytes of the document.

Example:

```
    TruncateDocSize    10000000
```

The default is zero, which means read all data.

Warning: If you use TruncateDocSize, use it with care! TruncateDocSize is a safety belt only, to limit e.g. filteroutput, when accessing databases, or to limit "runnaway" filters. Truncating doc input may destroy document structures for Swish-e (e.g. swish may miss closing tags for XML or HTML documents).

TruncateDocSize does not currently work with the `prog` input source method.

FuzzyIndexingMode NONE|Stemming|Soundex|Metaphone|DoubleMetaphone

Selects the type of index to create. Only one type of index may be created.

It’s a good idea to create both a normal index and a fuzzy index and allow your search interface select which index to use. Many people find the fuzzy searches to be too fuzzy.

The available fuzzy indexing options can be displayed by running

```
    swish-e -T LIST_FUZZY_MODES
```

Available options include:

None

Words are stored in the index without any conversion. This is the default.

Stemming_*

This options uses one of the installed Snowball stemmers (http://snowball.tartarus.org/).

The installed stemmers can be viewed by running

```
    swish-e -T LIST_FUZZY_MODES
```

For example, to use the Spanish stemming module:

```
    FuzzyIndexingMode Stemming_es
```

Stem or Stemming_en

Selects the legacy Swish-e English stemmer.

This is deprecated in favor of the Snowball English stemmers (Stemming_en1, Stemming_en2). Future versions of Swish-e will likely use the Stemming_en2 stemmer by default.

Words are converted using the Porter stemming algorithm.

From: http://www.tartarus.org/~martin/PorterStemmer/

```
     The Porter stemming algorithm (or Porter stemmer) is a
     process for removing the commoner morphological and inflexional
     endings from words in English. Its main use is as part of a
     term normalisation process that is usually done when setting up
     Information Retrieval systems.
```

This will help a search for "running" to also find "run" and "runs", for example.

The stemming function does not convert words to their root, rather programmatically removes endings on words in an attempt to make similar words with different endings stem to the same string of characters. It’s not a perfect system, and searches on stemmed indexes often return curious results. For example, two entirely different words may stem to the same word.

Stemming also can be confusing when used with a wildcard (truncation). For example, you might expect to find the word "running" by searching for "runn*". But this fails when using a stemmed index, as "running" stems to "run", yet searching for "runn*" looks for words that start with "runn".

Soundex

Soundex was developed in the 1880s so records for people with similar sounding names could be found more readily. Soundex is a coded surname based on the way a surname sounds rather than spelling. Surnames that sound similar, like Smith and Smyth, are filed together under the same Soundex code. This is mostly useful for US English.

Soundex should not be used to search for sound-alike words. Metaphone would be more appropriate for generic sound matching of words. Soundex should only be used where you need to search multiple documents for proper names which sound similar. This is primarily used for indexing genealogical records. This may be useful for indexing other collections of data consisting mostly of names. Many common name variations are matched by Soundex. The only notable exception is the first letter of the name. The first letter is not matched for sound.

Metaphone and DoubleMetaphone

Words are transformed into a short series of letters representing the sound of the word (in English). Metaphone algorithms are often used for looking up mis-spelled words in dictionary programs.

From: http://aspell.sourceforge.net/metaphone/

```
Lawrence Philips’ Metaphone Algorithm is an algorithm which returns
the rough approximation of how an English word sounds.
```

The `DoubleMetaphone` mode will sometimes generate two different metaphones for the same word. This is supposed to be useful when a word may be pronounced more than one way.

A metaphone index should give results somewhere in between Soundex and Stemming.

UseStemming [yes|NO]

Put yes to apply word stemming algorithm during indexing, else no.

```
UseStemming no
UseStemming yes
```

When UseStemming is set to `yes` every word is stemmed before placing it in to the index.

This option is deprecated. It has been superceded by FuzzyIndexingMode.

UseSoundex [yes|NO]

When UseSoundex is set to `yes` every word is converted to a Soundex code before placing it in to the index.

This option is deprecated. It has been superceded by FuzzyIndexingMode.

IgnoreTotalWordCountWhenRanking [YES|no]

Put yes to ignore the total number of words in the file when calculating ranking. Often better with merges and small files. Default is yes.

```
IgnoreTotalWordCountWhenRanking no
```

The default was changed from no to yes in version 2.2.

NOTE: must be set to **no** if you intend to use the -R 1 option when searching.

MinWordLimit *integer*

Set the minimum length of an word. Shorter words will not be indexed. The default is 1 (as defined in *src/config.h*).

```
MinWordLimit 5
```

MaxWordLimit *integer*

Set the maximum length of an indexable word. Every longer word will not be indexed. The Default is 40 (as defined in *src/config.h*).

WordCharacters *string of characters*

IgnoreFirstChar *string of characters*

IgnoreLastChar *string of characters*

BeginCharacters *string of characters*

EndCharacters *string of characters*

These settings define what a word consists of to the Swish-e indexing engine. Compiled in defaults are in *src/config.h*.

When indexing Swish-e uses **WordCharacters** to split up the document into words. Words are defined by any string of non-blank characters that contain only the characters listed in WordCharacters. If a string of characters includes a character that is not in WordCharacters then the word will be spit into two or more separate words.

For example:

```
WordCharacters abde
```

Would turn "abcde" into two words "ab" and "de".

Next, of these words, any characters defined in **IgnoreFirstChar** are stripped off the start of the word, and **IgnoreLastChar** characters are stripped off the end of the word. This allows, for example, periods within a word (www.slashdot.com), but not at the end of a word. Characters in IgnoreFirstChar and IgnoreLastChar must be in WordCharacters.

Finally, the resulting words MUST begin with one of the characters listed in **BeginCharacters** and end with one of the characters listed in **EndCharacters**. BeginCharacters and EndCharacters must be a subset of the characters in WordCharacters. Often, WordCharacters, BeginCharacters and EndCharacters will all be the same.

Note that the same process applies to the query while searching.

Getting these settings correct will take careful consideration and practice. It's helpful to create an index of a single test file, and then look at the words that are placed in the index (see the `-v 4`, `-D` and `-k` searching switches).

Currently there is only support for eight-bit characters.

Example:

```
WordCharacters  .abcdefghijklmnopqrstuvwxyz
BeginCharacters abcdefghijklmnopqrstuvwxyz
EndCharacters   abcdefghijklmnopqrstuvwxyz
IgnoreFirstChar .
IgnoreLastChar  .
```

So the string

```
Please visit http://www.example.com/path/to/file.html.
```

will be indexed as the following words:

```
please
visit
http
www.example.com
path
to
file.html
```

Which means that you can search for `www.example.com` as a single word, but searching for just `example` will not find the document.

Note: when indexing HTML documents HTML entities are converted to their character equivalents before being processed with these directives. This is a change from previous versions of Swish-e where you were required to include the characters `0123456789&#;` to index entities. See also ConvertHTMLEntities

Buzzwords [*list of buzzwords*|File: path]

The Buzzwords option allows you to specify words that will be indexed regardless of WordCharacters, BeginCharacters, EndCharacters, stemming, soundex and many of the other checks done on words while indexing.

Buzzwords are case insensitive.

Buzzwords should be separated by spaces and may span multiple directives. If the special format `File:filename` is used then the Buzzwords will be read from an external file during indexing.

Examples:

```
        Buzzwords C++ TCP/IP
```

```
        Buzzwords File: ./buzzwords.lst
```

If a Buzzword contains search operator characters they must be backslashed when searching. For example:

```
        Buzzwords C++ TCP/IP web=http
```

```
        ./swish-e -w ’web\=http’
```

Buzzwords are found by splitting the text on whitespace, removing IgnoreFirstChar and IgnoreLastChar characters from the word, and then comparing with the list of Buzzwords. Therefore, if adding Buzzwords to an index you will probably want to define IgnoreFirstChar and IgnoreLastChar settings.

Note: Buzzwords specific settings for IgnoreFirstChar and IgnoreLastChar may be used in the future.

IgnoreWords [*list of stop words*|File: path]

The IgnoreWords option allows you to specify words to ignore, called *stopwords*. The default is to not use any stopwords.

Words should be separated by spaces and may span multiple directives. If the special format `File:filename` is used then the stop words will be read from an external file during indexing.

In previous versions of Swish-e you could use the directive

```
IgnoreWords swishdefault - obsolete!
```

to include a default list of compiled in stopwords. This keyword is no longer supported.

Examples:

```
IgnoreWords www http a an the of and or
```

```
IgnoreWords File: ./stopwords.de
```

UseWords [*list of words*|File: path]

UseWords defines the words that Swish-e will index. **Only** the words listed will be indexed.

You can specify a list of words following the directive (you may specify more than one UseWords directive in a config file), and/or use the `File:` form to specify a path to a file containing the words:

```
UseWords perl python pascal fortran basic cobal php
UseWords File: /path/to/my/wordlist
```

Please drop the Swish-e list a note if you actually use this feature. It may be removed from future versions.

IgnoreLimit *integer integer*

This automatically omits words that appear too often in the files (these words are called stopwords). Specify a whole percentage and a number, such as "80 256". This omits words that occur in over 80% of the files and appear in over 256 files. Comment out to turn off auto-stopwording.

```
IgnoreLimit 50 1000
```

Swish-e must do extra processing to adjust the entire index when this feature is used. It is recommended that instead of using this feature that you decided what words are stopwords and add them to **IngoreWords** in your configuration file. To do this, use IgnoreLimit one time and note the stop words that are found while indexing. Add this list to IgnoreWords, and then remove IgnoreLimit from the configuration file.

IgnoreMetaTags *list of names*

IgnoreMetaTags defines a list of metatags to ignore while indexing XML files (and HTML files if using libxml2 for parsing HTML). All text within the tags will be ignored -- both for indexing (MetaNames) and properties (PropertyNames). To still parse properties, yet do not index the text, see UndefinedMetaTags.

This option is useful to avoid indexing specific data from a file. For example:

```
<person>
    <first_name>
        William
    </first_name> <last_name>
        Shakespeare
    </last_name> <updated_date>
        April 25, 1999
    </updated_date>
</person>
```

In the above example you might **not** want to index the updated date, and therefore prevent finding this record by searching

```
-w 'person=(April)'
```

This is solved by:

```
IgnoreMetaTags updated_date
```

See also UndefinedMetaTags.

IgnoreNumberChars *list of characters*

Experimental Feature

This experimental feature can be used to define a set of characters that describe a number. If a word is found to contain only those characters it will not be indexed. The characters listed must be part of WordCharacters settings. In other words, the "word" checked is a word that Swish-e would otherwise index.

For example,

```
IgnoreNumberChars 0123456789$.,
```

Then Swish-e would not index the following:

```
123
123,456.78
$123.45
```

You might be tempted to avoid indexing hex numbers with:

```
        IgnoreNumberChars 0123456789abcdef
```

which will not index 0D31, but will also not index the word "bad".

This is an experimental feature that may change in future versions. One possible change is to use regular expressions instead.

IndexComments [NO|yes]

This option allows the user decide if to index the contents of HTML comments. Default is no. Set to yes if comment indexing is required.

```
        IndexComments yes
```

Note: This is a change in the default behavior prior to version 2.2.

TranslateCharacters [*string1 string2*|:ascii7:]

The TranslateCharacters directive maps the characters in string1 to the characters listed in string2.

For example:

```
        # This will index a_b as a-b and ámo as amo
        TranslateCharacters _á -a
```

`TranslateCharacters :ascii7:` is a predefined set of characters that will translate eight bit characters to ascii7 characters. Using the :ascii7: rule will translate "Ääç" to "aac". This means: searching "Çelik", "çelik" or "celik" will all match the same word.

TranslateCharacters is done early in the indexing process, after converting HTML entities but before splitting the input text into words based on **WordCharacters**. So characters you are translating *from* do not need to be listed in word characters.

The same character translations take place when searching.

BumpPositionCounterCharacters *string*

When indexing Swish-e assigns a word position to each word. This enables phrase searching. There may be cases where you would like to prevent phrase matching. The BumpPositionCounterCharacters directive allows you to specify a set of characters that when found in the text will increment the word position -- effectively preventing phrase matches across that character.

For example, if you have a tag:

```
<subjects>
    computer programming | apple computers
</subjects>
```

You might want to prevent matching "programming apple" in that meta name.

```
BumpPositionCounterCharacters |
```

There is no default, and you may list a string of characters.

DontBumpPositionOnEndTags *list of names*

DontBumpPositionOnStartTags *list of names*

Since metatags are typically separate data fields, the word position counter is automatically bumped between metatags (actually, bumped when a start tag is found and when an end tag is found). This prevents matching a phrase that spans more than one metaname. DontBumpPositionOnEndTags and DontBumpPositionOnStartTags disables this feature for the listed metanames.

For example,

```
<person>
    <first_name>
        William
    </first_name>
    <last_name>
        Shakespeare
    </last_name>
    <updated_date>
        April 25, 1999
    </updated_date>
</person>
```

In the configuration file:

```
DontBumpPositionOnEndTags first_name
DontBumpPositionOnStartTags last_name
```

This configuration allows this phrase search

```
-w 'person=("william shakespeare")'
```

but this phrase search will fail

```
-w 'person=("shakespeare april")'
```

4.1.6 Directives for the File Access method only

Some directives have different uses depending on the source of the documents. These directives are only valid when using the **File system** method of indexing.

IndexOnly *list of file suffixes*

This directive specifies the allowable file suffixes (extensions) while indexing. The default is to index all files specified in **IndexDir**.

```
# Only index .html .htm and .q files
IndexOnly .html .htm .q
```

IndexOnly checks that the file end in the characters listed. It does not check "extensions". IndexOnly is tested right before FileRules is processed.

FollowSymLinks [yes|NO]

Put "yes" to follow symbolic links in indexing, else "no". Default is no.

```
FollowSymLinks no
FollowSymLinks yes
```

Note that when set to `no` extra `stat(2)` system calls must be made for each file. For large number of files you may see a small reduction in indexing time by setting this to `yes`.

See also the `-l` switch in SWISH-RUN.

FileRules [type] [contains|is|regex] *regular expression*

FileMatch [type] [contains|is|regex] *regular expression*

FileRules and FileMatch are used to, respectively, exclude and include files and directories to index. Since, by default, Swish-e indexes all files and recurses all directories (but see also FollowSymLinks) you will typically only use FileRules to exclude files or directories. FileMatch is useful in a few cases, for example, to override the behavior of IndexOnly. Some examples are included below.

Except for `FileRules title ...`, this feature is only available for file access method (-S fs), which is the default indexing mode. Also, any pathname modification with ReplaceRules happens after the check for FileRules. (It’s unlikely that you would exclude files with FileRules based on text you added with ReplaceRules!)

The regular expression is a C regex.h extended regular expression. You may supply more than one regular expression per line, or use separate directives. Preceding the regular expression with the word "not" negates the match.

The regular expression is compared against **[type]** as described below.

For historical reasons, you can specify `contains` or `is`. `is` simply forces the regular expression to match at the start and end of the string (by internally prepending "^" and appending "$" to the regular expression).

The `regex` option requires delimiter characters:

```
    FileRules title regex /^private/i
```

The only advantage of `regex` is if you want to do case insensitive matches, or simply like your regular expressions to look like perl regular expressions. You must use matching delimiters; (), {}, and [], are not currently supported for no good reason other than laziness.

Use quotes (" or ’) around a pattern if it contains any white space. Note that the backslash character becomes the escape character within quotes.

For example, these sets generate the same regular expressions.

```
    FileRules title is hello
    FileRules title contains ^hello$
    FileRules title regex /^hello$/
```

These all need quotes due to the included space character

```
    FileRules title is "hello there"
    FileRules title contains "^hello there$"
    FileRules title regex "!^hello there$!"
```

These show how the backslash must be doubled inside of quotes. Swish-e converts a double-backslash into a single backslash, and then passes that single onto the regular expression compiler.

```
    FileRules filename regex /\.pdf/
    FileRules filename regex "/\\.pdf/"
```

```
    FileRules filename regex !hello\\there!     # need double for real backslash
    FileRules filename regex "!hello\\\\there!" # need double-double inside of quotes
```

Matching Types

The following types of match strings my be supplied:

```
    FileRules pathname
    FileRules dirname
    FileRules filename
    FileRules directory
    FileRules title
```

```
FileMatch pathname
FileMatch filename
FileMatch dirname
FileMatch directory
```

pathname matches the regular expression against the current pathname. The pathname may or may not be absolute depending on what you supplied to IndexDir.

Example:

```
# Don’t index paths that contain private or hidden
FileRules pathname contains (private|hidden)
```

```
# Same thing
FileRules pathname regex /(private|hidden)/
```

```
# Don’t index exe files
FileRules pathname contains \.exe$
```

dirname and **filename** split the path name by the last delimiter character into a directory name, and a file name. Then these are compared against the patterns supplied. Directory names do **not** have a trailing slash. All path names use the forward slash as a delimiter within Swish-e.

Example:

```
# Same as last example - don’t index *.exe files.
FileRules filename contains \.exe$
```

```
# Don’t index any file called test.html files
FileRules filename contains ^test\.html$
```

```
# Same thing
FileRules filename is test\.html
```

```
# Don’t index any directories that contain "old"  (/usr/local/myold/docs)
FileRules dirname contains old
```

```
# Don’t index any directories that contain the path segment "old" (/usr/local/old/foo)
FileRules dirname contains /old/
```

```
# Index only .htm, .html, plus any all-digit file names
IndexOnly .htm .html
FileMatch filename contains ^\d+$
```

```
# Same as previous, but maybe a little slower
FileRules filename regex not !\.(htm|html)$!
FileMatch filename contains ^\d+$
```

Swish-e checks these settings in the order of `pathname`, `dirname`, and `filename`, and FileMatch patterns are checked before FileRules, in general. This allows you to exclude most files with FileRules, yet allow in a few special cases with FileMatch. For example:

```
# Exclude all files of .exe, .bin, and .bat
FileRules filename contains \.(exe|bin|bat)$
# But, let these two in
FileMatch filename is baseball\.bat incoming_mail\.bin
```

```
# Same, but as a single pattern
FileMatch filename is (baseball\.bat|incoming_mail\.bin)
```

The `directory` type is somewhat unique. When Swish-e recurses into a directory it will compare all the *files* in the directory with the pattern and then decide if that entire directory should or should not be indexed (or recursed). Note that you are matching against file names in a directory -- and some of those names may be directory names.

A `FileRules directory` match will cause Swish-e to ignore all files and sub-directories in the current directory.

Warning: A match with `FileMatch directory` says to index **everything** in the *current* directory and **ignore** any FileRules for this directory.

Example:

```
# Don't index any directories (and sub directories) that contain
# a file (or sub-directory) called "index.skip"
FileRules directory contains ^index\.skip$
```

```
# Don't index directories that contain a .htaccess file.
FileRules directory contains ^\.htaccess
```

Note: While *processing* directories, Swish-e will ignore any files or directories that begin with a dot ("."). You may index files or directories that begin with a dot by specifying their name with IndexDir or `-i`.

`title` checks for a pattern match in an HTML title.

Example:

```
FileRules title contains construction example pointers
```

```
# This example says to ignore case
FileRules title regex "/^Internal document/i"
```

Note: `FileRules title` works for any input method (fs, prog, or http) that is parsed as HTML, and where a title was found in the document.

In case all this seems a bit confusing, processing a directory happens in the following order.

First the directory name is checked:

```
FileRules dirname - reject entire directory if matches
```

Next the directory is scanned and each file name (which might be the name of a sub-directory) is checked:

```
FileRules directory - reject entire dir if *any* files match
FileMatch directory - accept entire dir if *any* files match
```

Then, unless `FileMatch directory` matched, each file is tested with FileMatch. A match says to index the file without further testing (i.e. overrides FileRules and IndexOnly):

```
FileMatch pathname  \
FileMatch dirname   - file is accepted if any match
FileMatch filename  /
```

otherwise

```
IndexOnly - file is checked for the correct file extension
```

```
FileRules pathname  \
FileRules dirname   - file is rejected if any match
FileRules filename  /
```

finally, the file is indexed.

Files (not directories) listed with IndexDir or `-i` are processed in a similar way:

```
FileMatch pathname  \
FileMatch dirname   - file is accepted if any match
FileMatch filename  /
```

otherwise, the file is rejected if it doesn’t have the correct extension or a FileRules matches.

```
    IndexOnly - file is checked for the correct file extension
```

```
    FileRules pathname  \
    FileRules dirname   - file is rejected if any match
    FileRules filename  /
```

Note: If things are not indexing as you expect, create a directory with some test files and use the `-T regex` trace option to see how file names are checked. Start with very simple tests!

4.1.7 Directives for the HTTP Access Method Only

The HTTP Access method is enabled by the "-S http" switch when indexing. It works by running a Perl program called SwishSpider which fetches documents from a web server.

Only text files (content-type of "text/*") are indexed with the HTTP Access Method. Other document types (e.g. PDF or MSWord) may be indexed as well. The SwishSpider will attempt to make use of the SWISH::Filter module (included with the Swish-e distribution) to convert documents into a format that Swish-e can index.

Note: The -S prog method of spidering (using spider.pl) can be a replacement for the -S http method. It offers more configuration options and better spidering speed.

These directives below are available when using the HTTP Access Method of indexing.

MaxDepth *integer*

MaxDepth defines how many links the spider should follow before stopping. A value of 0 configures the spider to traverse all links. The default is MaxDepth 0.

```
    MaxDepth 5
```

Note: The default was changed from 5 to 0 in release 2.4.0

Delay *seconds*

The number of seconds to wait between issuing requests to a server. This setting allows for more friendly spidering of remote sites. The default is 5 seconds.

```
    Delay 1
```

Note: The default was changed from 60 to 5 seconds in release 2.4.0

TmpDir *path*

The location of a writable temp directory on your system. The HTTP access method tells the Perl helper to place its files in this location, and the `-e` switch causes Swish-e to use this directory while indexing. There is no default.

```
    TmpDir /tmp/swish
```

If this directory does not exist or is not writable Swish-e will fail with an error during indexing.

Note, the environment variables of `TMPDIR`, `TMP`, and `TEMP` (in that order) will **override** this setting.

SpiderDirectory *path*

The location of the Perl helper script called *swishspider*. If you use a relative directory, it is relative to your directory when you run Swish-e, not to the directory that Swish-e is in. The default is the location swishspider was installed. Normally this does not need to be set.

```
    SpiderDirectory /usr/local/swish
```

EquivalentServer *server alias*

Often times the same site may be referred to by different names. A common example is that often http://www.some-server.com and http://some-server.com are the same. Each line should have a list of all the method/names that should be considered equivalent. Multiple EquivalentServer directives may be used. Each directive defines its own set of equivalent servers.

```
    EquivalentServer http://library.berkeley.edu http://www.lib.berkeley.edu
    EquivalentServer http://sunsite.berkeley.edu:2000 http://sunsite.berkeley.edu
```

4.1.8 Directives for the prog Access Method Only

This section details the directives that are only available for the "prog" document source feature of Swish-e. The "prog" access method runs an external program that "feeds" documents to Swish-e. This allows indexing and filtering of documents from any source.

See prog - general purpose access method in the SWISH-RUN man page for more information.

A number of example programs for use with the "prog" access method are provided in the *prog-bin* directory. Please see those example if you have questions about implementing a "prog" input program.

SwishProgParameters *list of parameters*

This is a list of parameters that will be sent to the external program when running with the "prog" document source method.

```
SwishProgParameters /path/to/config hello there
IndexDir /path/to/program.pl
```

Then running:

```
swish-e -c config -S prog
```

Swish-e will execute `/path/to/program.pl` and pass `/path/to/config hello there` as three command line arguments to the program. This directive makes it easy to pass settings from the Swish-e configuration file to the external program.

For example, the `spider.pl` program (included in the `prog-bin` directory) uses the SwishProgParameters to specify what file to read for configuration information.

```
SwishProgParameters spider.config
IndexDir ./spider.pl
```

The `spider.pl` program also has a default action so you can avoid using a configuration file:

```
SwishProgParameters default http://www.swishe.org/ http://some.other.site/
IndexDir ./spider.pl
```

And the spider program will use default settings for spidering those sites.

Swish-e can read documents from standard input, so another way to run an external program with parameters is:

```
./spider.pl spider.conf | ./swish-e -S prog -i stdin
```

Notes when using MS Windows

You should use unix style path separators to specify your external program. Swish will convert forward slashes to backslashes before calling the external program. This is only true for the program name specified with IndexDir or the `-i` command line option.

In addition, Swish-e will make sure the program specified actually exists, which means you need to use the full name of the program.

For example, to run the perl spider program *spider.pl* you would need a Swish-e configuration file such as:

```
IndexDir e:/perl/bin/perl.exe
SwishProgParameters prog-bin/spider.pl default http://swish-e.org
```

and run indexing with the command:

```
    swish-e -c swish.cfg -S prog -v 9
```

The IndexDir command tells Swish-e the name of the program to run. Under unix you can just specify the name of the script, since unix will figure out the program from the first line of the script.

The SwishProgParameters are the parameters passed to the program specified by IndexDir (perl.exe in this case). The first parameter is the perl script to run (*prog-bin/spider.pl*). Perl passes the rest of the parameters directly to the perl script. The second parameter *default* tells the *spider.pl* program to use default settings for spidering (or you could specify a spider config file -- see `perldoc spider.pl` for details), and lastly, the URL is passed into the spider program.

4.1.9 Document Filter Directives

Internally, Swish-e knows how to parse only text, HTML, and XML documents. With "filters" you can index other types of documents. For example, if all your web pages are in gzip format a filter can uncompress these on the fly for indexing.

You may wish to read the Swish-e FAQ question on filtering before continuing here. How Do I filter documents?

There are two suggested methods for filtering.

4.1.9.1 Filtering with SWISH::Filter

The Swish-e distribution includes a Perl module called SWISH::Filter and individual filters located in the *filters* directory. This system uses plug-in filters to extend the types of documents that Swish-e can index. The plug-in filters do not actually do the filtering, but rather provide a standard interface for accessing programs that can filter or convert documents. The programs that do the filtering are not part of the Swish-e distribution; they must be downloaded and installed separately.

The advantage of this method is that new filtering methods can be installed easily.

This system is designed to work with the -S http and -prog methods, but may also be used with the FileFilter feature and -S fs indexing method. See *$prefix/share/doc/swish-e/examples/filter-bin/swish_filter.pl* for an example.

See the *filters/README* file for more information.

4.1.9.2 Filtering with the FileFilter feature

A filter is an external program that Swish-e executes while processing a document of a given type. Swish-e will execute the filter program for each file that matches the file suffix (extension) set in the **FileFilter** or **FileFilterMatch** directives. **FileFilterMatch** matches using regular expressions and is described below.

Filters may be used with any type of input method (i.e. -S fs, -S http, or -S prog). But because

Swish-e calls the external program passing as **default** arguments:

$0

the name of the filter program

$1

the physical path name of the file to read. This may be a temporary file location if indexing by the http method.

$2

When indexing under the file system this will be the same as `$1` (the path to the source file), but when indexing under the http method this will be the URL of the source document.

Swish-e can also pass other parameters to the filter program. These parameters can be defined using the **FileFilter** or **FileFilterMatch** directives. See Filter Options below.

The filter program must open the file, process its contents, and return it to Swish-e by printing to STDOUT.

Note that this can add a significant amount of time to the indexing process if your external program is a perl or shell script. If you have many files to filter you should consider writing your filter in C instead of a shell or perl script, or using the "prog" Access Method.

FilterDir *path-to-directory*

This is the path to a directory where the filter programs are stored. Swish-e looks in this directory to find the filter specified in the **FileFilter** directive. If this directive is omitted, you have to specify the full path to the filterscript on each FileFilter directive.

This feature does *not* apply to the FileFilterMatch directive.

Example:

```
        FilterDir /usr/local/swish/filters
```

FileFilter *suffix* "filter-prog" ["filter-options"]

This maps file suffix (extension) to a filter program. If *filter-prog* starts with a directory delimiter (absolute path), Swish-e doesn't use the FilterDir settings, but uses the given *filter-prog* path directly.

Filter options:

Filter options are a string passed as arguments to the *filter-prog*. Filter options can contain variables, replaced by Swish-e. If you omit *filter-options* Swish-e will use default parameters for the options listed above.

```
Default:      "'%p' '%P'"
Which means:  pass   "workfile path" and "documentfile path" to filter (each quoted).
```

Variables in filter options:

```
%%   =  %
%P   =  Full document pathname (e.g. URL, or path on filesystem)
%p   =  Full pathname to work file (maybe a tmpfile or the real document path on filesystem)
%F   =  Filename stripped from full document pathname
%f   =  Filename stripped from "work" pathname
%D   =  Directoryname stripped from full document pathname
%d   =  Directoryname stripped from full "work" pathname
```

Examples of strings passed:

```
%P =  document pathname:  http://myserver/path1/mydoc.txt
%p =  work pathname:      /tmp/tmp.1234.mydoc.txt
%F =     mydoc.txt
%f =     tmp.1234.mydoc.txt
%D =     http://myserver/path1
%d =     /tmp
```

Important hint for security:

When using variable substitution, use quotes to ensure filename integrity.

```
        e.g. "'%f'"  -->  'file name with spaces.doc'.
```

If you don't use this, your system security may be compromised, or filtering may not work for these files.

Notes when using MS Windows

Windows uses double quotes to escape shell metacharacters, so reverse the quotes in the examples above. e.g.:

```
        '"%f"' --> "file name with spaced.doc"
```

You can specify the filter program using forward slashes (unix style). Swish will convert the slashes to backslashes before running your program.

```
FileFilter .mydoc     c:/some/path/mydocfilter.exe  '-d "%d" -example -url "%P" "%f"'
```

Examples of filters:

```
FileFilter .doc       /usr/local/bin/catdoc "-s8859-1 -d8859-1 '%p'"
FileFilter .pdf       pdftotext   "'%p' -"
FileFilter .html.gz   gzip  "-c '%p'"
FileFilter .mydoc     "/some/path/mydocfilter"  "-d '%d' -example -url '%P' '%f'"
```

The above examples are running a *binary* filter program. For more complicated filtering needs you may use a scripting language such as Perl or a shell script. Here's some examples of calling a shell and perl script:

```
FileFilter .pdf       pdf2html.sh
FileFilter .ps        ghostscript-filter.pl
```

Using a scripting language (or any language that has a large startup cost) can **greatly increase the indexing time**. For small indexing jobs, this may not be an issue, but for large collections of files that require processing by a scripting language, you may be better off using the `-S prog` access method where the script will only be compiled once, instead of for each document.

Filters are probably easier to write than a `-S prog` program. Which you decide to use depends on your requirements. Examples of filter scripts can be found in the *filter-bin* directory, and examples of `-S prog` programs can be found in the *prog-bin* directory.

FileFilterMatch *filter-prog* *filter-options* *regex* [*regex* ...]

This is similar to FileMatch except uses regular expressions to match against the file name. *filter-prog* is the path to the program. Unlike FileFilter this does **not** use the FilterDir option. Also unlike FileFilter you **must** specify the *filter-options*.

Examples:

```
FileFilterMatch ./pdftotext "'%p' -" /\.pdf$/
```

Note that will also match a file called ".pdf", so you may want to use something that requires a file-name that has more than just an extension. For example:

```
FileFilterMatch ./pdftotext "'%p' -" /.\.pdf$/
```

To specify more than one extension:

```
FileFilterMatch ./check_title.pl "%p" /\.html$/  /\.htm$/
```

Or a few ways to do the same thing:

```
    FileFilterMatch ./check_title.pl %p /\.(html|html)$/
    FileFilterMatch ./check_title.pl %p /\.html?$/
```

And to ignore case:

```
    FileFilterMatch ./check_title.pl %p /\.html?$/i
```

You may also precede an expression with "not" to negate regular expression that follow. For example, to match files that do not have an extension:

```
    FileFilterMatch ./convert "%p %P" not /\..+$/
```

4.2 Document Info

$Id: SWISH-CONFIG.pod,v 1.81 2004/10/25 13:57:17 karman Exp $

.

5 SWISH-RUN - Running Swish-e and Command Line Switches

5.1 OVERVIEW

The Swish-e program is controlled by command line arguments (called *switches*). Often, it is run manually from a shell (command prompt), or from a program such as a CGI script that passes the command line arguments to swish.

Note: A number of the command line switches may be specified in the Swish-e configuration file specified with the `-c` command line argument. Please see SWISH-CONFIG for a complete description of available configuration file directives.

There are two basic operating modes of Swish-e: indexing and searching. There are command line arguments that are unique to each mode, and others that apply to both (yet may have different meaning depending on the operating mode). These command line arguments are listed below, grouped by:

INDEXING -- describes the command line arguments used while indexing.

SEARCHING -- lists the command line arguments used while searching.

OTHER SWITCHES -- lists switches that don't apply to searching or indexing.

Beginning with Swish-e version 2.1, you may embed its search engine into your applications. Please see SWISH-LIBRARY.

5.2 INDEXING

Swish-e indexing is initiated by passing *command line arguments* to swish. The command line arguments used for *searching* are described in SEARCHING. Also, see SWISH-SEARCH for examples of searching with Swish-e.

Swish-e usage:

```
    swish-e [-i dir file ... ] [-c file] [-f file] [-l] \
            [-v (num)] [-S method(fs|http|prog)] [-N path]
```

The `-h` switch (help) will list the available Swish-e command line arguments:

```
    swish-e -h
```

Typically, most if not all indexing settings are placed in a configuration file (specified with the `-c` switch). Once the configuration file is setup indexing is initiated as:

```
    swish-e -c /path/to/config/file
```

See SWISH-CONFIG for information on the configuration file.

Security Note: If the swish binary is named *swish-search* then swish will not allow any operation that would cause swish to write to the index file.

When indexing it may be advisable to index to a temporary file, and then after indexing has successfully completed rename the file to the final location. This is especially important when replacing an index that is currently in use.

```
swish-e -c swish.config -f index.tmp
[check return code from swish or look for err: output]
mv index.tmp index.swish-e
```

5.2.1 Indexing Command Line Arguments

-i *directories and/or files* (input file)

This specifies the directories and/or files to index. Directories will be indexed recursively. This is typically specified in the configuration file with the **IndexDir** directive instead of on the command line. Use of this switch overrides the configuration file settings.

-S [fs|http|prog] (document source/access mode)

This specifies the method to use for accessing documents to index. Can be either fs for local indexing via the file system (the default), http for spidering, or prog for reading documents from an external program.

Located in the `conf` directory are example configuration files that demonstrate indexing with the different document source methods.

See the SWISH-FAQ for a discussion on the different indexing methods, and the difference between spidering with the http method vs. using the file system method.

fs - file system

The fs method simply reads files from a local (or networked) drive. This is the default method if the `-S` switch is not specified. See SWISH-CONFIG for configuration directives specific to the fs method.

http - spider a web server

The http method is used to spider web servers. It uses an included helper program called *swish-spider*. See SWISH-CONFIG for configuration directives specific to the http method.

Security Note: Under Windows swish passes the URLs fetched from remote documents through the shell (swish uses the `system()` command for running *swishspider* under Windows), and this may be considered an additional security risk.

The http method is deprecated (or at least not very well appreciated). Consider using the prog method described below for spidering. There's a spider program available in the *prog-bin* directory for use with the prog method. Here's a number of limitation with this method that are solved with the prog method:

- swishspider only spiders standard <a href="..."> links. Frames and other links are not followed.
- By default, this method of spidering only indexes files that have a content type of "text/*" (e.g. text/plain, text/html, text/xml). You should use DefaultContents and IndexContents to map file extensions to parsers used by swish (e.g. `IndexContents HTML* .html .htm`), but this will fail where a document does not have a file extension.
- Swish-e's FileFilter directive can be used with the http access method, although it requires a separate process (in addition to the swsihspider process) for each document filtered.
- The SWISH::Filter modules can be used with the swishspider program. SWISH::Filter provides a general purpose filtering system (see SWISH::Filter documentation). To use SWISH::Filter set PERL5LIB to point to the location of the SWISH module name space (typically /usr/local/lib/swish-e under Unix). For example:

  ```
  export PERL5LIB=/usr/local/lib/swish-e  # bash, bourne shells
  setenv PERL5LIB /usr/local/lib/swish-e  # csh, tcsh
  ```

  or under Windows

  ```
  set PERL5LIB=c:\program files\swish-e2.4\lib\swish-e
  ```

  SWISH::Filter is not enabled by default due to the overhead of loading the modules for every document fetched.

  The Swish-e distribution includes perl modules in the SWISH::Filters::* namespace to make converting non-text documents into a format that Swish-e can parse easy. As mentioned above, the helper script *swishspider* will use these modules if can be found via PERL5LIB. These modules only provide an interface to programs that do the conversion. For example, you will need to download and install the "catdoc" program to convert MSWord documents into text for indexing. Please see *filters/README* to see how to use this filter system.

prog - general purpose access method

The prog method is new to Swish-e version 2.2. It's designed as a general purpose method to feed documents to swish from an external program.

For example, the external program can read a database (e.g. MySQL), spider a web server, or convert documents from one format to another (e.g. pdf to html). Or, you can simply use it to read the files of the file system (like `-S fs`), yet provide you with full control of what files are indexed.

The external program name to run is passed to swish either by the IndexDir directive, or via the `-i` option.

The program specified should be an absolute path as swish-e will attempt to `stat()` the program to make sure it exists. Swish does this to help in error reporting.

If the program specified with -i or IndexDir is not an absolute path (i.e. does not include "/") then swish-e will append the "libexecdir" directory defined during configuration. Typically, libexecdir is set to "$prefix/lib/swish-e" (/usr/local/lib/swish-e), but is platform and installation dependent. Running swish-e -h will report the directory.

For example, the -S prog program "spider.pl" is a Perl helper program for use with -S prog and is installed in libexecdir.

```
IndexDir spider.pl
SwishProgParameters default http://localhost/index.html
```

and swish-e will find spider.pl in libexecdir.

Additional parameters may be passed to the external program via the SwishProgParameters directive. In the example above swish-e will pass two parameters to spider.pl, "default" and "http://localhost/index.html".

A special name "stdin" may be used with `-i` or IndexDir which tells swish to read from standard input instead of from an external program. See example below.

The external program prints to standard output (which swish captures) a set of headers followed by the content of the file to index. The output looks similar to an email message or a HTTP document returned by a web server in that it includes name/value pairs of headers, a blank line, and the content.

The content length is determined by a content-length header supplied to swish by the program; there is no "end of record" character or flag sent between documents. Therefore, it is critical that the content-length header is correct. This is a common source of errors.

One advantage of this method (over using filters, for example) is that the external program is run only once for the entire indexing job, instead of once for every document. This avoids forking and creating a new process for every document, and makes a huge difference when your external program is something like perl that has a large startup cost.

Here’s a simple example written in Perl:

```
#!/usr/local/bin/perl -w
use strict;
```

```
    # Build a document
    my $doc = <<EOF;
    <html>
    <head>
        <title>Document Title</title>
    </head>
        <body>
            This is the text.
        </body>
    </html>
    EOF
```

```
    # Prepare the headers for swish
    my $path = 'Example.file';
    my $size = length $doc;
    my $mtime = time;
```

```
    # Output the document (to swish)
    print <<EOF;
    Path-Name: $path
    Content-Length: $size
    Last-Mtime: $mtime
    Document-Type: HTML*
```

```
    EOF
```

```
        print $doc;
```

The external program passes to swish a header. The header is separated from the body of the document with a blank line. The available headers are:

Path-Name:

This is the name of the file you are indexing. This can be any string, so for example it could be an ID of a record in a database, a URL or a simple file name.

This header is required.

Content-Length:

This header specifies the length in bytes of the document that follows the header. This length must be exactly the length of the document -- do not make the mistake of adding an extra line feed at the end of the document.

This header is required.

Last-Mtime:

Thi parameter is the last modification time of the file, and must be a time stamp (seconds since the Epoch on your platform).

This header is not required.

Document-Type:

You may override swish's determination of document type (`Indexcontents`) by using the `Document-Type:` header. The document type is used to select which parser Swish-e uses to parse the document's contents.

For example, a spider program might map the content-type returned from a web server to one of the types Swish-e understands. For example,

```
    my $doc_type = 'HTML*' if $response->content_type =~ m!text/html!'
```

This header is not required.

The above example program only returns one document and exits, which is not very useful. Normally, your program would read data from some source, such as files or a database, format as XML, HTML, or text, and pass them to swish, one after another. The `Content-Length:` header tells swish where each document ends -- there is not any special "end of record" character or marker.

To index with the above example you need to make sure that the program is executable (and that the path to perl is correct), and then call swish telling to run in prog mode, and the name of the program to use for input.

```
    % chmod 755 example.pl
    % ./swish-e -S prog -i ./example.pl
```

Programs can and should be tested prior to running swish. For example:

```
    % ./example.pl > test.out
```

A few more useful example programs are provided in the swish-e distribution located in the *prog-bin* directory. Some include documentation:

```
    % cd prog-bin
    % perldoc spider.pl
```

Others are small examples that include comments:

```
% cd prog-bin
% less DirTree.pl
```

The *spider.pl* program can be used as a replacement for the *-S http* method. It is far more feature-rich and offers much more control over indexing.

If you use the special program name "stdin" with `-i` or IndexDir then swish-e will read from standard input instead of from a program. For example:

```
% ./example.pl --count=1000 /path/to/data | ./swish-e -S prog -i stdin
```

This is basically the same as using a swish-e configuration file of:

```
SwishProgParameters --count=1000 /path/to/data
IndexDir ./example.pl
```

in a config file and running

```
% ./swish-e -S prog -c swish.conf
```

This gives an easy way to run swish without a configuration file with a `-S prog` program that requires parameters. It also means you can capture data to a file and then index more once with the same data:

```
% ./example.pl /path/to/data --count=1000 > docs.txt
% cat docs.txt | ./swish-e -S prog -i stdin -c normal_index
% cat docs.txt | ./swish-e -S prog -i stdin -c fuzzy_index
```

Using "stdin" might also be useful for programs that call swish (instead of swish calling the program).

(The reason "stdin" is used instead of the more common "-" dash is due to the rotten way swish parses the command line. This should be fixed in the future.)

The prog method bypasses some of the configuration parameters available to the file system method -- settings such as IndexOnly, FileRules, FileMatch and FollowSymLinks are ignored when using the prog method. It's expected that these operations are better accomplished in the external program before passing the document onto swish. In other words, when using the prog method, only send the documents to swish that you want indexed.

You may use swish's filter feature with the prog method, but performance will be better if you run filtering programs from within your external program. See also *filters/README* for an example how to easily add document converstion and filtering into your Perl-based programs.

Notes when using -S prog on MS Windows

Windows does not use the shebang (#!) line of a program to determine the program to run. So, when running, for example, a perl program you may need to specify the perl.exe binary as the program, and use the SwishProgParameters to name the file.

```
    IndexDir e:/perl/bin/perl.exe
    SwishProgParameters read_database.pl
```

Swish will replace the forward slashes with backslashes before running the command specified with IndexDir. Swish uses the `popen(3)` command which passes the command through the shell.

-f *indexfile* (index file)

If you are indexing, this specifies the file to save the generated index in, and you can only specify one file. See also **IndexFile** in the configuration file.

If you are searching, this specifies the index files (one or more) to search from. The default index file is index.swish-e in the current directory.

-c *file ...* (configuration files)

Specify the configuration `file(s)` to use for indexing. This file contains many directives that control how Swish-e proceeds. See SWISH-CONFIG for a complete listing of configuration file directives.

Example:

```
    swish-e -c docs.conf
```

If you specify a directory to index, an index file, or the verbose option on the command-line, these values will override any specified in the configuration file.

You can specify multiple configuration files. For example, you may have one configuration file that has common site-wide settings, and another for a specific index.

Examples:

```
    1) swish-e -c swish-e.conf
    2) swish-e -i /usr/local/www -f index.swish-e -v -c swish-e.conf
    3) swish-e -c swish-e.conf stopwords.conf
```

1. The settings in the configuration file will be used to index a site.

2. These command-line options will override anything in the configuration file.
3. The variables in swish-e.conf will be read, then the variable in stopwords.conf will be read. Note that if the same variables occur in both files, older values may be written over.

-e (economy mode)

For large sites indexing may require more RAM than is available. The `-e` switch tells swish to use disk space to store data structures while indexing, saving memory. This option is recommended if swish uses so much RAM that the computer begins to swap excessively, and you cannot increase available memory. The trade-off is slightly longer indexing times, and a busy disk drive.

-l (symbolic links)

Specifying this option tells swish to follow symbolic links when indexing. The configuration file value **FollowSymLinks** will override the command-line value.

The default is not to follow symlinks. A small improvement in indexing time my result from enabling FollowSymLinks since swish does not need to stat every directory and file processed to determine if it is a symbolic link.

-N path (index only newer files)

The `-N` option takes a path to a file, and only files *newer* than the specified file will be indexed. This is helpful for creating incremental indexes -- that is, indexes that contain just files added since the last full index was created of all files.

Example (bad example)

```
    swish-e -c config.file -N index.swish-e -f index.new
```

This will index as normal, but only files with a modified date newer than *index.swish-e* will be indexed.

This is a bad example because it uses *index.swish-e* which one might assume was the date of last indexing. The problem is that files might have been added between the time indexing read the directory and when the *index.swish-e* file was created -- which can be quite a bit of time for very large indexing jobs.

The only solution is to prevent any new file additions while full indexing is running. If this is impossible then it will be slightly better to do this:

Full indexing:

```
    touch indexing_time.file
    swish-e -c config.file -f index.tmp
    mv index.tmp index.full
```

Incremental indexing:

```
swish-e -c config.file -N indexing_time.file -f index.tmp
mv index.tmp index.incremental
```

Then search with

```
swish-e -w foo -f index.full index.incremental
```

or merge the indexes

```
swish-e -M index.full index.incremental index.tmp
mv index.tmp index.swish-e
swish-e -w foo
```

-v [0|1|2|3] (verbosity level)

The `-v` option can take a numerical value from 0 to 3. Specify 0 for completely silent operation and 3 for detailed reports.

If no value is given then 1 is assumed. See also **IndexReport** in the configuration file.

Warnings and errors are reported regardless of the verbosity level. In addition, all error and warnings are written to standard out. This is for historical reasons (many scripts exist that parse standard out for error messages).

5.3 SEARCHING

The following command line arguments are available when searching with Swish-e. These switches are used to select the index to search, what fields to search, and how and what to print as results.

This section just lists the available command line arguments and their usage. Please see SWISH-SEARCH for detailed searching instructions.

Warning: If using Swish-e via a CGI interface, please see CGI Danger!

Security Note: If the swish binary is named *swish-search* then swish will not allow any operation that would cause swish to write to the index file.

5.3.1 Searching Command Line Arguments

-w *word1 word2 ...* (query words)

This performs a case-insensitive search using a number of keywords. If no index file to search is specified (via the `-f` switch), swish-e will try to search a file called index.swish-e in the current directory.

```
swish-e -w word
```

Phrase searching is accomplished by placing the quote delimiter (a double-quote by default) around the search phrase.

```
swish-e -w 'word or "this phrase"'
```

Search would should be protected from the shell by quotes. Typically, this is single quotes when running under Unix.

Under Windows *command.com* you may not need to use quotes, but you will need to backslash the quotes used to delimit phrases:

```
swish-e -w \"a phrase\"
```

The phrase delimiter can be set with the `-P` switch.

The search may be limited to a *MetaName*. For example:

```
swish-e -w meta1=(foo or baz)
```

will only search within the **meta1** tag.

Please see SWISH-SEARCH for a description of MetaNames

-f *file1 file2 ...* (index files)

Specifies the index `file(s)` used while searching. More than one file may be listed, and each file will be searched. If no `-f` switch is specified then the file *index.swish-e* in the current directory will be used as the index file.

-m *number* (max results)

While searching, this specifies the maximum number of results to return. The default is to return all results.

This switch is often used in conjunction with the `-b` switch to return results one page at a time (strongly recommended for large indexes).

-b *number* (beginning result)

Sets the *begining* search result to return (records are numbered from 1). This switch can be used with the `-m` switch to return results in groups or pages.

Example:

```
swish-e -w 'word' -b 1 -m 20    # first 'page'
swish-e -w 'word' -b 21 -m 20   # second 'page'
```

-t HBthec (context searching)

The `-t` option allows you to search for words that exist only in specific HTML tags. Each character in the string you specify in the argument to this option represents a different tag in which to search for the word. H means all HEAD tags, B stands for BODY tags, t is all TITLE tags, h is H1 to H6 (header) tags, e is emphasized tags (this may be B, I, EM, or STRONG), and c is HTML comment tags

search only in header (<H*>) tags

```
swish-e -w word -t h
```

-d *string* (delimiter)

Set the delimiter used when printing results. By default, Swish-e separates the output fields by a space, and places double-quotes around the document title. This output may be hard to parse, so it is recommended to use `-d` to specify a character or string used as a separator between fields.

The string `dq` means "double-quotes".

```
swish-e -w word -d ,    # single char
swish-e -w word -d ::   # string
swish-e -w word -d '"'  # double quotes under Unix
swish-e -w word -d \"   # double quotes under Windows
swish-e -w word -d dq   # double quotes
```

The following control characters may also be specified: `\t \r \n \f`.

Warning: This string is passed directly to `sprintf()` and therefore exposes a securty hole. Do not allow user data to set -d format strings directly.

-P *character*

Sets the delimiter used for phrase searches. The default is double quotes `"`.

Some examples under bash: (be careful about you shell metacharacters)

```
swish-e -P ^ -w 'title=^words in a phrase^'
swish-e -P \' -w "title='words in a pharse"'
```

-p *property1 property2 ...* (display properties)

This causes swish to print the listed property in the search results. The properties are returned in the order they are listed in the `-p` argument.

Properties are defined by the **ProperNames** directive in the configuration file (see SWISH-CONFIG) and properties must also be defined in **MetaNames**. Swish stores the text of the meta name as a *property*, and then will return this text while searching if this option is used.

Properties are very useful for returning data included in a source documnet without having to re-read the source document while searching. For example, this could be used to return a short document description. See also see **Document Summeries** and PropertyNames in SWISH-CONFIG.

To return the subject and category properties while indexing.

```
    swish-e -w word -p subject category
```

Properties are returned in double quotes. If a property contains a double quote it is HTML escaped ("). See the `-x` switch for a more advanced method of returning a list of properties.

NOTE: it is necessary to have indexed with the proper PropertyNames directive in the user config file in order to use this option.

-s *property [asc|desc] ...* (sort)

Normally, search results are printed out in order of relevancy, with the most relevant listed first. The `-s` sort switch allows you to sort results in order of a specified *property*, where a *property* was defined using the **MetaNames** and **PropertyNames** directives during indexing (see SWISH-CONFIG).

The string passed can include the strings `asc` and `desc` to specify the sort order, and more than one property may be specified to sort on more than one key.

Examples:

sort by title property ascending order

```
    -s title
```

sort descending by title, ascending by name

```
    -s title desc name asc
```

Note: Swish limits sort keys to 100 characters. This limit can be changed by changing MAX_SORT_STRING_LEN in src/config.h and rebuilding swish-e.

-L limit to a range of property values (Limit)

This is an experimental feature!

The `-L` switch can be used to limit search results to a range of property values

Example:

```
swish-e -w foo -L swishtitle a m
```

finds all documents that contain the word `foo`, and where the document's title is in the range of `a` to `m`, inclusive. By default, the case of the property is ignored, but this can be changed by using PropertyNamesCompareCase configuation directive.

Limiting may be done with user-defined properties, as well.

For example, if you indexed documents that contain a created timestamp in a meta tag:

```
<meta name="created_on" content="982648324">
```

Then you tell Swish that you have a property called `created_on`, and that it's a timestamp.

```
PropertyNamesDate created_on
```

After indexing you will be able to limit documents to a range of timestamps:

```
-w foo -L created_on  946684800 949363199
```

will find documents containing the word foo and that have a created_on date from the start of Jan 1, 2000 to the end of Jan 31, 2000.

Note: swish currently does not parse dates; Unix timestamps must be used.

Two special formats can be used:

```
-L swishtitle <= m
-L swishtitle >= m
```

Finds titles less than or equal, or grater than or equal to the letter `m`.

This feature will not work with `swishrank` or `swishdbfile` properties.

This feature takes advantages of the pre-sorted tables built by swish during indexing to make this feature fast while searching. You should see in the indexing output a line such as:

```
    6 properties sorted.
```

That indicates that six pre-sorted tables were built during indexing. By default, all properties are presorted while indexing. What properties are pre-sorted can be controlled by the configuration parameter PreSortedIndex.

Using the `-L` switch on a property that was not pre-sorted will still work, but may be *much* slower during searching.

Note that the PropertyNamesSortKeyLength setting is used for sorting properties. Using too small a PropertyNamesSortKeyLength could result in -L selecting the wrong properties due to incomplete sorting.

This is an experimental feature, and its use and interface are subject to change.

-x formatstring (extended output format)

The `-x` switch defines the output format string. The format string can contain plain text and property names (including swish-defined internal property names) and is used to generate the output for every result. In addition, the output format of the property name can be controlled with C-like printf format strings. This feature overrides the cmdline switches `-d` and `-p`, and a warning will be generated if `-d` or `-p` are used with `-x`.

Warning: The format string (fmt) is passed directly to `sprintf()` and therefore exposes a securty hole. Do not allow user data to set -x format strings directly.

For example, to return just the title, one per line, in the search results:

```
    swish-e  -w ...   -x '<swishtitle>\n' ...
```

Note: the `\n` may need to be protected from your shell.

See also ResultExtFormatName for a way to define *named* format strings in the swish configuration file.

Format of "formatstring":

```
    "text<propertyname>text<propertyname fmt=propfmtstr>text..."
```

Where **propertyname** is:

- the name of a user property as specified with the config file directive "PropertyNames"
- the name of a swish Auto property (see below). These properties are defined automatically by swish -- you do not need to specify them with PropertyNames directive. (This may change in the future.)

propertynames must be placed within "<" and ">".

User properties:

Swish-e allows you to specify certain META tags within your documents that can be used as **document properties**. The contents of any META tag that has been identified as a document property can be returned as part of the search results. Doucment properties must be defined while indexing using the **PropertyNames** configuration directive (see SWISH-CONFIG).

Examples of user-defined PropertyNames:

```
<keywords>
<author>
<deliveredby>
<reference>
<id>
```

Auto properties:

Swish defines a number of "Auto" properties for each document indexed. These are available for output when using the `-x` format.

```
Name               Type     Contents
--------------     -------  ---------------------------------------------
swishreccount      Integer  Result record counter
swishtitle         String   Document title
swishrank          Integer  Result rank for this hit
swishdocpath       String   URL or filepath to document
swishdocsize       Integer  Document size in bytes
swishlastmodified  Date     Last modified date of document
swishdescription   String   Description of document (see:StoreDescription)
swishdbfile        String   Path of swish database indexfile
```

The Auto properties can also be specified using shortcuts:

```
Shortcut    Property Name
--------    --------------
  %c        swishreccount
  %d        swishdescription
  %D        swishlastmodified
  %I        swishdbfile
  %p        swishdocpath
  %r        swishrank
  %l        swishdocsize
  %t        swishtitle
```

For example, these are equivalent:

```
    -x '<swishrank>:<swishdocpath>:<swishtitle>\n'
    -x '%r:%p:%t\n'
```

Use a double percent sign "%%" to enter a literal percent sign in the output.

Formatstrings of properties:

Properties listed in an `-x` format string can include format control strings. These "propertyformats" are used to control how the contents of the associated property are printed. Property formats are used like C-language printf formats. The property format is specified by including the attribute "fmt" within the property tag.

Format strings cannot be used with the "%" shortcuts described above.

General syntax:

```
     -x '<propertyname fmt="propfmtstr">'
```

where `subfmt` controls the output format of `propertyname`.

Examples of property format strings:

```
         date type:    <swishlastmodified fmt="%d.%m.%Y">
         string type:  <swishtitle fmt="%-40.35s">
         integer type: <swishreccount fmt=/%8.8d/>
```

Please see the manual pages for `strftime(3)` and `sprintf(3)` for an explanation of format strings. Note: some versions of strftime do not offer the `%s` format string (number of seconds since the Epoch), so swish provides a special format string "%ld" to display the number of seconds since the Epoch.

The first character of a property format string defines the delimiter for the format string. For example,

```
     -x  "<author  fmt=[%20s]> ...\n"
     -x  "<author  fmt='%20s'> ...\n"
     -x  "<author  fmt=/%20s/> ...\n"
```

Standard predefined formats:

If you ommit the sub-format, the following formats are used:

```
     String type:       "%s"  (like printf char *)
     Integer type:      "%d"  (like printf int)
     Float type:        "%f"  (like printf double)
     Date type:         "%Y-%m-%d %H:%M:%S" (like strftime)
```

Text in "formatstring" or "propfmtstr":

Text will be output as-is in format strings (and property format strings). Special characters can be escaped with a backslash. To get a new line for each result hit, you have to include the Newline-Character "\n" at the end of "fmtstr".

```
-x "<swishreccount>|<swishrank>|<swishdocpath>\n"
-x "Count=<swishreccount>, Rank=<swishrank>\n"
-x "Title=\<b\><swishtitle>\</b\>"
-x 'Date: <swishlastmodified fmt="%m/%d/%Y">\n'
-x 'Date in seconds: <swishlastmodified fmt=/%ld/>\n'
```

Control/Escape charcters:

you can use C-like control escapes in the format string:

```
known controls:      \a, \b, \f, \n, \r, \t, \v,
digit escapes:       \xhexdigits   \0octaldigits
character escapes:   \anychar
```

Example,

```
swish -x "%c\t%r\t%p\t\"<swishtitle fmt=/%40s/>\"\n"
```

Examples of -x format strings:

```
-x "%c|%r|%p|%t|%D|%d\n"
-x "%c|%r|%p|%t|<swishdate fmt=/%A, %d. %B %Y/>|%d\n"
-x "<swishrank>\t<swishdocpath>\t<swishtitle>\t<keywords>\n
-x "xml_out: \<title\><swishtitle>\>\</title\>\n"
-x "xml_out: <swishtitle fmt='<title>%s</title>'>\n"
```

-H [0|1|2|3|<n>] (header output verbosity)

The `-H n` switch generates extened *header* output. This is most useful when searching more than one index file at a time by specifying more than one index file with the `-f` switch. `-H 2` will generate a set of headers specific to each index file. This gives access to the settings used to generate each index file.

Even when searching a single index file, `-H n` will provided additional information about the index file, how it was indexed, and how swish is interperting the query.

```
-H 0 : print no header information, output only search result entries.
-H 1 : print standard result header (default).
-H 2 : print additional header information for each searched index file.
-H 3 : enhanced header output (e.g. print stopwords).
-H 9 : print diagnostic information in the header of the results (changed from: C<-v 4>)
```

-R [0|1] (Ranking Scheme)

This is an experimental feature!

The default ranking scheme in SWISH-E evaluates each word in a query in terms of its frequency and position in each document. The default scheme is 0.

New in version 2.4.3 you may optionally select an experimental ranking scheme that, in addition to document frequency and position, uses Inverse Document Frequency (IDF), or the relative frequency of each word across all the indexes being searched, and Relative Density, or the normalization of the frequency of a word in relationship to the number of words in the document.

NOTE: IgnoreTotalWordCountWhenRanking must be set to **no** or **0** in your `index(es)` for -R 1 to work.

Specify -R 1 to turn on IDF ranking. See the API documentation for how to set the ranking scheme in your Perl or C program.

5.4 OTHER SWITCHES

-V (version)

Print the current version.

-k *letter* (print out keywords)

The `-k` switch is used for testing and will cause swish to print out all keywords in the index beginning with that letter. You may enter `-k '*'` to generate a list of all words indexed by swish.

-D *index file* (debug index)

The -D option is no longer supported in version 2.2.

-T *options* (trace/debug swish)

The -T option is used to print out information that may be helpful when debugging swish-e's operation. This option replaced the `-D` option of previous versions.

Running `-T help` will print out a list of available *options*

5.5 Merging Index Files

In previous versions of Swish-e indexing would require a very large amount of memory and the indexing process could be very slow. Merging provided a way to index in chunks and then combine the indexes together into a single index.

Indexing is much faster now and uses much less memory, and with the `-e` switch very little memory is needed to index a large site.

Still, at times it can be useful to merge different index files into one file for searching. This could be because you want to keep separate site indexes and a common one for a global search, or you have separate collections of documents that you wish to search all at one time, but manage separately.

-M *index1 index2 ... indexN out_index

Merges the indexes specified on the command line -- the last file name entered is the output file. The output index must not exist (otherwise merge will not proceed).

Only indexes that were indexed with common settings may be merged. (e.g. don’t mix stemming and non-stemming indexes, or indexes with different WordCharacter settings, etc.).

Use the `-e` switch while merging to reduce memory usage.

Merge generates progress messages regardless of the setting of `-v`.

-c *configuration file*

Specify a configuration file while indexing to add administrative information to the output index file.

5.6 Document Info

$Id: SWISH-RUN.pod,v 1.35 2004/08/31 03:15:03 karman Exp $

.

6 SWISH-SEARCH - Swish-e Searching Instructions

6.1 OVERVIEW

This page describes the process of searching with Swish-e. Please see the SWISH-CONFIG page for information the Swish-e configuration file directives, and SWISH-RUN for a complete list of command line arguments.

Searching a Swish-e index involves passing command line arguments to it that specify the index file to use, and the query (or search words) to locate in the index. Swish-e returns a list of file names (or URLs) that contain the matched search words. Perl is often used as a front-end to Swish-e such as in CGI applications, and perl modules exist to for interfacing with Swish-e.

6.2 Searching Syntax and Operations

The `-w` command line argument is used specify the search query to Swish-e.

```
    swish-e -w airplane
```

will find all documents that contain the word **airplane**.

When running Swish-e from a shell prompt, be careful to protect your query from shell metacharacters and shell expansions. This often means placing single or double quotes around your query. See Searching with Perl if you plan to use Perl as a front end to Swish-e. In the examples below single quotes are used to protect the search from the shell.

The following section describes various aspects of searching with Swish-e.

6.2.1 Boolean Operators

You can use the Boolean operators **and**, **or** or **not** in searching. Without these Boolean operators Swish-e will assume you're **and**'ing the words together. The operators are not case sensitive. These three searches are the same:

```
    swish-e -w foo bar
    swish-e -w bar foo
    swish-e -w foo AND bar
```

[Note: you can change the default to **or**ing by changing the variable DEFAULT_RULE in the config.h file and recompiling Swish-e.]

The **not** operator inverts the results of a search.

```
   swish-e -w not foo
```

finds all the documents that do not contain the word foo.

Parentheses can be used to group searches.

```
swish-e -w ’not (foo and bar)’
```

The result is all documents that have none or one term, but not both.

To search for the words **and**, **or**, or **not**, place them in a double quotes. Remember to protect the quotes from the shell:

```
swish-e -w ’"not"’
swish-e -w \"not\"
```

will search for the word "not".

Other examples:

```
swish-e -w smilla or snow
```

Retrieves files containing either the words "smilla" or "snow".

```
swish-e -w smilla snow not sense
swish-e -w ’(smilla and snow) and not sense’  (same thing)
```

retrieves first the files that contain both the words "smilla" and "snow"; then among those the ones that do not contain the word "sense".

6.2.2 Truncation

The wildcard (*) is available, however it can only be used at the end of a word: otherwise is is considered a normal character (i.e. can be searched for if included in the WordCharacters directive).

```
swish-e -w librarian
```

this query only retrieves files which contain the given word.

On the other hand:

```
swish-e -w ’librarian*’
```

retrieves "librarians", "librarianship", etc. along with "librarian".

Note that wildcard searches combined with word stemming can lead to unexpected results. If stemming is enabled, a search term with a wildcard will be stemmed internally before searching. So searching for `running*` will actually be a search for `run*`, so `running*` would find `runway`. Also, searching for `runn*` will not find `running` as you might expect, since `running` stems to `run` in the index, and thus `runn*` will not find `run`.

6.2.3 Order of Evaluation

In general, the order of evaluation is not important. Internally swish-e processes the search terms from left to right. Parenthesis can be used to group searches together, effectively changing the order of evaluation. For example these three are the same:

```
swish-e -w foo not bar baz
swish-e -w not bar foo baz
swish-e -w baz foo not bar
```

but these two are **not** the same:

```
swish-e -w foo not bar baz
swish-e -w foo not (bar baz)
```

The first finds all documents that contain both foo and baz, but do not contain bar. The second finds all that contain foo, and contain either bar or baz, but not both.

It is often helpful in understanding searches to use the boolean terms and parenthesis. So the above two become:

```
swish-e -w foo AND (not bar) AND baz
swish-e -w foo AND (not (bar AND baz))
```

These four examples are all the same search (assuming that AND is the default search type):

```
swish-e -w ’juliet not ophelia and pac’
swish-e -w ’(juliet) AND (NOT ophelia) AND (pac)’
swish-e -w ’juliet not ophelia pac’
swish-e -w ’pac and juliet and not ophelia’
```

Looking at the the first three searches, first Swish-e finds all the documents with "juliet". Then it finds all documents that do not contain "ophelia". Those two lists are then combined with the boolean AND operator resulting with a list of documents that include "juliet" but not "ophelia". Finally, that list is ANDed with the list of documents that contain "pac" resulting.

However it is always possible to force the order of evaluation by using parenthesis. For example:

```
    swish-e -w 'juliet not (ophelia and pac)'
```

retrieves files with "juliet" that do not contain both words "ophelia" and "pac".

6.2.4 Meta Tags

MetaNames are used to represent *fields* (called *columns* in a database) and provide a way to search in only parts of a document. See SWISH-CONFIG for a description of MetaNames, and how they are specified in the source document.

To limit a search to words found in a meta tag you prefix the keywords with the name of the meta tag, followed by the equal sign:

```
    metaname = word
    metaname = (this or that)
    metaname = ( (this or that) or "this phrase" )
```

It is not necessary to have spaces at either side of the "=", consequently the following are equivalent:

```
    swish-e -w "metaName=word"
    swish-e -w "metaName = word"
    swish-e -w "metaName= word"
```

To search on a word that contains a "=", precede the "=" with a "\" (backslash).

```
    swish-e -w "test\=3 = x\=4 or y\=5"
```

this query returns the files where the word "x=4" is associated with the metaName "test=3" or that contains the word "y=5" not associated with any metaName.

Queries can be also constructed using any of the usual search features, moreover metaName and plain search can be mixed in a single query.

```
     swish-e -w "metaName1 = (a1 or a4) not (a3 and a7)"
```

This query will retrieve all the files in which "a1" or "a2" are found in the META tag "metaName1" and that do not contain the words "a3" and "a7", where "a3" and "a7" are not associated to any meta name.

6.2.5 Phrase Searching

To search for a phrase in a document use double-quotes to delimit your search terms. (The phrase delimiter is set in src/swish.h.)

You must protect the quotes from the shell.

For example, under Unix:

```
     swish-e -w '"this is a phrase" or (this and that)'
     swish-e -w 'meta1=("this is a phrase") or (this and that)'
```

Or under Windows:

```
     swish-e -w \"this is a phrase\" or (this and that)
```

You can not use boolean search terms inside a phrase. That is:

```
     swish-e -w 'this and that'
```

finds documents with both words "this" and "that", but:

```
     swish-e -w '"this and that"'
```

finds documents that have the phrase "that and that". A phrase can consist of a single word, so this is how to search for the words used as boolean operators:

```
    swish-e -w 'this "and" that'
```

finds documents that contain all three words, but in any order.

You can use the `-P` switch to set the phrase delimiter character. See SWISH-RUN for examples.

6.2.6 Context

At times you might not want to search for a word in every part of your files since you know that the `word(s)` are present in a particular tag. The ability to search according to context greatly increases the chances that your hits will be relevant, and Swish-e provides a mechanism to do just that.

The -t option in the search command line allows you to search for words that exist only in specific HTML tags. Each character in the string you specify in the argument to this option represents a different tag in which the word is searched; that is you can use any combinations of the following characters:

```
     H means all<HEAD> tags
     B stands for <BODY> tags
     t is all <TITLE> tags
     h is <H1> to <H6> (header) tags
     e is emphasized tags (this may be <B>, <I>, <EM>, or <STRONG>)
     c is HTML comment tags (<!-- ... -->)
```

```
# This search will look for files with these two words in their titles only.
swish-e -w "apples oranges" -t t
```

```
# This search will look for files with these words in comments only.
swish-e -w "keywords draft release" -t c
```

```
This search will look for words in titles, headers, and emphasized tags.
swish-e -w "world wide web" -t the
```

6.3 Searching with Perl

Perl (http://www.perl.com/) is probably the most common programming language used with Swish-e, especially in CGI interfaces. Perl makes searching and parsing results with Swish-e easy, but if not done properly can leave your server vulnerable to attacks.

When designing your CGI scripts you should carefully screen user input, and include features such as paged results and a timer to limit time required for a search to complete. These are to protect your web site against a denial of service (DoS) attack.

Included with every distribution of Perl is a document called perlsec -- Perl Security. *Please* take time to read and understand that document before writing CGI scripts in perl.

Type at your shell/command prompt:

```
perldoc perlsec
```

If nothing else, start every CGI program in perl as such:

```
#!/usr/local/bin/perl -wT
use strict;
```

That alone won’t make your script secure, but may help you find insecure code.

6.3.1 CGI Danger!

There are many examples of CGI scripts on the Internet. Many are poorly written and insecure. A commonly seen way to execute Swish-e from a perl CGI script is with a *piped open*. For example, it is common to see this type of `open()`:

```
open(SWISH, "$swish -w $query -f $index|");
```

This `open()` gives shell access to the entire Internet! Often an attempt is made to strip `$query` of *bad* characters. But, this often fails since it’s hard to guess what every *bad* character is. Would you have thought about a null? A better approach is to only allow *in* known safe characters.

Even if you can be sure that any user supplied data is safe, this *piped open* still passes the command parameters through the shell. If nothing else, it's just an extra unnecessary step to running Swish-e.

Therefore, the recommended approach is to fork and exec `swish-e` directly without passing through the shell. This process is described in the perl man page `perlipc` under the appropriate heading **Safe Pipe Opens**.

Type:

```
    perldoc perlipc
```

If all this sounds complicated you may wish to use a Perl module that does all the hard work for you.

6.3.2 Perl Modules

The Swish-e distribution includes a Perl module called SWISH::API. SWISH::API provides access to the Swish-e C Library.

The SWISH::API module is *not* installed by default.

The SWISH::API module will *embed* Swish-e into your perl program so that searching does not require running an external program. Embedding the Swish-e program into your perl program results in faster Swish-e searches, especially when running under a persistent environment like mod_perl since it avoids the cost of opening the index file for every request (mod_perl is much also much faster than CGI because it avoids the need to compile Perl code for every request).

See the README file in the *perl* directory of the Swish-e distribution for installation instructions. Documentation for the SWISH::API module is available at http://swish-e.org and is installed along with other HTML documentation on your computer.

6.4 Document Info

$Id: SWISH-SEARCH.pod,v 1.6 2003/12/18 05:00:39 whmoseley Exp $

.

7 The Swish-e FAQ - Answers to Common Questions

7.1 Frequently Asked Questions

7.1.1 General Questions

7.1.1.1 What is Swish-e?

Swish-e is **S**imple **W**eb **I**ndexing **S**ystem for **H**umans **- E**nhanced. With it, you can quickly and easily index directories of files or remote web sites and search the generated indexes for words and phrases.

7.1.1.2 So, is Swish-e a search engine?

Well, yes. Probably the most common use of Swish-e is to provide a search engine for web sites. The Swish-e distribution includes CGI scripts that can be used with it to add a *search engine* for your web site. The CGI scripts can be found in the *example* directory of the distribution package. See the *README* file for information about the scripts.

But Swish-e can also be used to index all sorts of data, such as email messages, data stored in a relational database management system, XML documents, or documents such as Word and PDF documents -- or any combination of those sources at the same time. Searches can be limited to fields or *MetaNames* within a document, or limited to areas within an HTML document (e.g. body, title). Programs other than CGI applications can use Swish-e, as well.

7.1.1.3 Should I upgrade if I’m already running a previous version of Swish-e?

A large number of bug fixes, feature additions, and logic corrections were made in version 2.2. In addition, indexing speed has been drastically improved (reports of indexing times changing from four hours to 5 minutes), and major parts of the indexing and search parsers have been rewritten. There’s better debugging options, enhanced output formats, more document meta data (e.g. last modified date, document summary), options for indexing from external data sources, and faster spidering just to name a few changes. (See the CHANGES file for more information.

Since so much effort has gone into version 2.2, support for previous versions will probably be limited.

7.1.1.4 Are there binary distributions available for Swish-e on platform foo?

Foo? Well, yes there are some binary distributions available. Please see the Swish-e web site for a list at http://swish-e.org/.

In general, it is recommended that you build Swish-e from source, if possible.

7.1.1.5 Do I need to reindex my site each time I upgrade to a new Swish-e version?

At times it might not strictly be necessary, but since you don’t really know if anything in the index has changed, it is a good rule to reindex.

7.1.1.6 What’s the advantage of using the libxml2 library for parsing HTML?

Swish-e may be linked with libxml2, a library for working with HTML and XML documents. Swish-e can use libxml2 for parsing HTML and XML documents.

The libxml2 parser is a better parser than Swish-e’s built-in HTML parser. It offers more features, and it does a much better job at extracting out the text from a web page. In addition, you can use the `ParserWarningLevel` configuration setting to find structural errors in your documents that could (and would with Swish-e’s HTML parser) cause documents to be indexed incorrectly.

Libxml2 is not required, but is strongly recommended for parsing HTML documents. It’s also recommended for parsing XML, as it offers many more features than the internal Expat xml.c parser.

The internal HTML parser will have limited support, and does have a number of bugs. For example, HTML entities may not always be correctly converted and properties do not have entities converted. The internal parser tends to get confused when invalid HTML is parsed where the libxml2 parser doesn’t get confused as often. The structure is better detected with the libxml2 parser.

If you are using the Perl module (the C interface to the Swish-e library) you may wish to build two versions of Swish-e, one with the libxml2 library linked in the binary, and one without, and build the Perl module against the library without the libxml2 code. This is to save space in the library. Hopefully, the library will someday soon be split into indexing and searching code (volunteers welcome).

7.1.1.7 Does Swish-e include a CGI interface?

Yes. Kind of.

There’s two example CGI scripts included, swish.cgi and search.cgi. Both are installed at *$prefix/lib/swish-e*.

Both require a bit of work to setup and use. Swish.cgi is probably what most people will want to use as it contains more features. Search.cgi is for those that want to start with a small script and customize it to fit their needs.

An example of using swish.cgi is given in the INSTALL man page, and it the swish.cgi documentation. Like often is the case, it will be easier to use if you first read the documentation.

Please use caution about CGI scripts found on the Internet for use with Swish-e. Some are not secure.

The included example CGI scripts were designed with security in mind. Regardless, you are encouraged to have your local Perl expert review it (and all other CGI scripts you use) before placing it into production. This is just a good policy to follow.

7.1.1.8 How secure is Swish-e?

We know of no security issues with using Swish-e. Careful attention has been made with regard to common security problems such as buffer overruns when programming Swish-e.

The most likely security issue with Swish-e is when it is run via a poorly written CGI interface. This is not limited to CGI scripts written in Perl, as it's just as easy to write an insecure CGI script in C, Java, PHP, or Python. A good source of information is included with the Perl distribution. Type `perldoc perlsec` at your local prompt for more information. Another must-read document is located at `http://www.w3.org/Security/faq/wwwsf4.html`.

Note that there are many *free* yet insecure and poorly written CGI scripts available -- even some designed for use with Swish-e. Please carefully review any CGI script you use. Free is not such a good price when you get your server hacked...

7.1.1.9 Should I run Swish-e as the superuser (root)?

No. Never.

7.1.1.10 What files does Swish-e write?

Swish writes the index file, of course. This is specified with the IndexFile configuration directive or by the `-f` command line switch.

The index file is actually a collection of files, but all start with the file name specified with the IndexFile directive or the `-f` command line switch.

For example, the file ending in *.prop* contains the document properties.

When creating the index files Swish-e appends the extension *.temp* to the index file names. When indexing is complete Swish-e renames the *.temp* files to the index files specified by IndexFile or `-f`. This is done so that existing indexes remain untouched until it completes indexing.

Swish-e also writes temporary files in some cases during indexing (e.g. `-s http`, `-s prog` with filters), when merging, and when using `-e`). Temporary files are created with the `mkstemp(3)` function (with 0600 permission on unix-like operating systems).

The temporary files are created in the directory specified by the environment variables `TMPDIR` and `TMP` in that order. If those are not set then swish uses the setting the configuration setting TmpDir. Otherwise, the temporary file will be located in the current directory.

7.1.1.11 Can I index PDF and MS-Word documents?

Yes, you can use a *Filter* to convert documents while indexing, or you can use a program that "feeds" documents to Swish-e that have already been converted. See `Indexing` below.

7.1.1.12 Can I index documents on a web server?

Yes, Swish-e provides two ways to index (spider) documents on a web server. See `Spidering` below.

Swish-e can retrieve documents from a file system or from a remote web server. It can also execute a program that returns documents back to it. This program can retrieve documents from a database, filter compressed documents files, convert PDF files, extract data from mail archives, or spider remote web sites.

7.1.1.13 Can I implement keywords in my documents?

Yes, Swish-e can associate words with *MetaNames* while indexing, and you can limit your searches to these MetaNames while searching.

In your HTML files you can put keywords in HTML META tags or in XML blocks.

META tags can have two formats in your source documents:

```
<META NAME="DC.subject" CONTENT="digital libraries">
```

And in XML format (can also be used in HTML documents when using libxml2):

```
<meta2>
    Some Content
</meta2>
```

Then, to inform Swish-e about the existence of the meta name in your documents, edit the line in your configuration file:

```
MetaNames DC.subject meta1 meta2
```

When searching you can now limit some or all search terms to that MetaName. For example, to look for documents that contain the word apple and also have either fruit or cooking in the DC.subject meta tag.

7.1.1.14 What are document properties?

A document property is typically data that describes the document. For example, properties might include a document’s path name, its last modified date, its title, or its size. Swish-e stores a document’s properties in the index file, and they can be reported back in search results.

Swish-e also uses properties for sorting. You may sort your results by one or more properties, in ascending or descending order.

Properties can also be defined within your documents. HTML and XML files can specify tags (see previous question) as properties. The *contents* of these tags can then be returned with search results. These user-defined properties can also be used for sorting search results.

For example, if you had the following in your documents

```
    <meta name="creator" content="accounting department">
```

and `creator` is defined as a property (see PropertyNames in SWISH-CONFIG) Swish-e can return `accounting department` with the result for that document.

```
     swish-e -w foo -p creator
```

Or for sorting:

```
     swish-e -w foo -s creator
```

7.1.1.15 What’s the difference between MetaNames and PropertyNames?

MetaNames allows keywords searches in your documents. That is, you can use MetaNames to restrict searches to just parts of your documents.

PropertyNames, on the other hand, define text that can be returned with results, and can be used for sorting.

Both use *meta tags* found in your documents (as shown in the above two questions) to define the text you wish to use as a property or meta name.

You may define a tag as **both** a property and a meta name. For example:

```
    <meta name="creator" content="accounting department">
```

placed in your documents and then using configuration settings of:

```
     PropertyNames creator
     MetaNames creator
```

will allow you to limit your searches to documents created by accounting:

```
     swish-e -w ’foo and creator=(accounting)’
```

That will find all documents with the word `foo` that also have a creator meta tag that contains the word `accounting`. This is using MetaNames.

And you can also say:

```
swish-e -w foo -p creator
```

which will return all documents with the word `foo`, but the results will also include the contents of the `creator` meta tag along with results. This is using properties.

You can use properties and meta names at the same time, too:

```
swish-e -w creator=(accounting or marketing) -p creator -s creator
```

That searches only in the `creator` *meta name* for either of the words `accounting` or `marketing`, prints out the contents of the contents of the `creator` *property*, and sorts the results by the `creator` *property name*.

(See also the `-x` output format switch in SWISH-RUN.)

7.1.1.16 Can Swish-e index multi-byte characters?

No. This will require much work to change. But, Swish-e works with eight-bit characters, so many characters sets can be used. Note that it does call the ANSI-C `tolower()` function which does depend on the current locale setting. See `locale(7)` for more information.

7.1.2 Indexing

7.1.2.1 How do I pass Swish-e a list of files to index?

Currently, there is not a configuration directive to include a file that contains a list of files to index. But, there is a directive to include another configuration file.

```
IncludeConfigFile /path/to/other/config
```

And in `/path/to/other/config` you can say:

```
IndexDir file1 file2 file3 file4 file5 ...
IndexDir file20 file21 file22
```

You may also specify more than one configuration file on the command line:

```
./swish-e -c config_one config_two config_three
```

Another option is to create a directory with symbolic links of the files to index, and index just that directory.

7.1.2.2 How does Swish-e know which parser to use?

Swish can parse HTML, XML, and text documents. The parser is set by associating a file extension with a parser by the IndexContents directive. You may set the default parser with the DefaultContents directive. If a document is not assigned a parser it will default to the HTML parser (HTML2 if built with libxml2).

You may use Filters or an external program to convert documents to HTML, XML, or text.

7.1.2.3 Can I reindex and search at the same time?

Yes. Starting with version 2.2 Swish-e indexes to temporary files, and then renames the files when indexing is complete. On most systems renames are atomic. But, since Swish-e also generates more than one file during indexing there will be a very short period of time between renaming the various files when the index is out of sync.

Settings in *src/config.h* control some options related to temporary files, and their use during indexing.

7.1.2.4 Can I index phrases?

Phrases are indexed automatically. To search for a phrase simply place double quotes around the phrase.

For example:

```
    swish-e -w 'free and "fast search engine"'
```

7.1.2.5 How can I prevent phrases from matching across sentences?

Use the BumpPositionCounterCharacters configuration directive.

7.1.2.6 Swish-e isn't indexing a certain word or phrase.

There are a number of configuration parameters that control what Swish-e considers a "word" and it has a debugging feature to help pinpoint any indexing problems.

Configuration file directives (SWISH-CONFIG) WordCharacters, BeginCharacters, EndCharacters, IgnoreFirstChar, and IgnoreLastChar are the main settings that Swish-e uses to define a "word". See SWISH-CONFIG and SWISH-RUN for details.

Swish-e also uses compile-time defaults for many settings. These are located in *src/config.h* file.

Use of the command line arguments `-k`, `-v` and `-T` are useful when debugging these problems. Using `-T INDEXED_WORDS` while indexing will display each word as it is indexed. You should specify one file when using this feature since it can generate a lot of output.

```
    ./swish-e -c my.conf -i problem.file -T INDEXED_WORDS
```

You may also wish to index a single file that contains words that are or are not indexing as you expect and use -T to output debugging information about the index. A useful command might be:

```
      ./swish-e -f index.swish-e -T INDEX_FULL
```

Once you see how Swish-e is parsing and indexing your words, you can adjust the configuration settings mentioned above to control what words are indexed.

Another useful command might be:

```
       ./swish-e -c my.conf -i problem.file -T PARSED_WORDS INDEXED_WORDS
```

This will show white-spaced words parsed from the document (PARSED_WORDS), and how those words are split up into separate words for indexing (INDEXED_WORDS).

7.1.2.7 How do I keep Swish-e from indexing numbers?

Swish-e indexes words as defined by the WordCharacters setting, as described above. So to avoid indexing numbers you simply remove digits from the WordCharacters setting.

There are also some settings in *src/config.h* that control what "words" are indexed. You can configure swish to never index words that are all digits, vowels, or consonants, or that contain more than some consecutive number of digits, vowels, or consonants. In general, you won't need to change these settings.

Also, there's an experimental feature called IgnoreNumberChars which allows you to define a set of characters that describe a number. If a word is made up of **only** those characters it will not be indexed.

7.1.2.8 Swish-e crashes and burns on a certain file. What can I do?

This shouldn't happen. If it does please post to the Swish-e discussion list the details so it can be reproduced by the developers.

In the mean time, you can use a FileRules directive to exclude the particular file name, or pathname, or its title. If there are serious problems in indexing certain types of files, they may not have valid text in them (they may be binary files, for instance). You can use NoContents to exclude that type of file.

Swish-e will issue a warning if an embedded null character is found in a document. This warning will be an indication that you are trying to index binary data. If you need to index binary files try to find a program that will extract out the text (e.g. `strings(1)`, `catdoc(1)`, `pdftotext(1))`.

7.1.2.9 How to I prevent indexing of some documents?

When using the file system to index your files you can use the FileRules directive. Other than `FileRules title`, FileRules only works with the file system (`-S fs`) indexing method, not with `-S prog` or `-S http`.

If you are spidering, use a *robots.text* file in your document root. This is a standard way to excluded files from search engines, and is fully supported by Swish-e. See http://www.robotstxt.org/

You can also modify the *spider.pl* spider perl program to skip, index content only, or spider only listed web pages. Type `perldoc spider.pl` in the `prog-bin` directory for details.

If using the libxml2 library for parsing HTML, you may also use the Meta Robots Exclusion in your documents:

```
    <meta name="robots" content="noindex">
```

See the obeyRobotsNoIndex directive.

7.1.2.10 How do I prevent indexing parts of a document?

To prevent Swish-e from indexing a common header, footer, or navigation bar, AND you are using libxml2 for parsing HTML, then you may use a fake HTML tag around the text you wish to ignore and use the IgnoreMetaTags directive. This will generate an error message if the `ParserWarningLevel` is set as it's invalid HTML.

IgnoreMetaTags works with XML documents (and HTML documents when using libxml2 as the parser), but not with documents parsed by the text (TXT) parser.

If you are using the libxml2 parser (HTML2 and XML2) then you can use the the following comments in your documents to prevent indexing:

```
        <!-- SwishCommand noindex -->
        <!-- SwishCommand index -->
```

and/or these may be used also:

```
        <!-- noindex -->
        <!-- index -->
```

7.1.2.11 How do I modify the path or URL of the indexed documents.

Use the ReplaceRules configuration directive to rewrite path names and URLs. If you are using `-S prog` input method you may set the path to any string.

7.1.2.12 How can I index data from a database?

Use the "prog" document source method of indexing. Write a program to extract out the data from your database, and format it as XML, HTML, or text. See the examples in the `prog-bin` directory, and the next question.

7.1.2.13 How do I index my PDF, Word, and compressed documents?

Swish-e can internally only parse HTML, XML and TXT (text) files by default, but can make use of *filters* that will convert other types of files such as MS Word documents, PDF, or gzipped files into one of the file types that Swish-e understands.

Please see SWISH-CONFIG and the examples in the *filters* and *filter-bin* directory for more information.

See the next question to learn about the filtering options with Swish-e.

7.1.2.14 How do I filter documents?

The term "filter" in Swish-e means the converstion of a document of one type (one that swish-e cannot index directly) into a type that Swish-e can index, namely HTML, plain text, or XML. To add to the confusion, there are a number of ways to accomplish this in Swish-e. So here's a bit of background.

The FileFilter directive was added to swish first. This feature allows you to specify a program to run for documents that match a given file extension. For example, to filter PDF files (files that end in .pdf) you can specify the configuation setting of:

```
    FileFilter .pdf pdftotext   "'%p' -"
```

which says to run the program "pdftotext" passing it the pathname of the file (%p) and a dash (which tells pdftotext to output to stdout). Then for each .pdf file Swish-e runs this program and reads in the filtered document from the output from the filter program.

This has the advantage that it is easy to setup -- a single line in the config file is all that is needed to add the filter into Swish-e. But it also has a number of problems. For example, if you use a Perl script to do your filtering it can be very slow since the filter script must be run (and thus compiled) for each processed document. This is exacerbated when using the -S http method since the -S http method also uses a Perl script that is run for every URL fetched. Also, when using -S prog method of input (reading input from a program) using FileFilter means that Swish-e must first read the file in from the external program and then write the file out to a temporary file before running the filter.

With -S prog it makes much more sense to filter the document in the program that is fetching the documents than to have swish-e read the file into memory, write it to a temporary file and then run an external program.

The Swish-e distribution contains a couple of example -S prog programs. *spider.pl* is a reasonably full-featured web spider that offers many more options than the -S http method. And it is much faster than running -S http, too.

The spider has a perl configuration file, which means you can add programming logic right into the configuration file without editing the spider program. One bit of logic that is provided in the spider's configuration file is a "call-back" function that allows you to filter the content. In other words, before the spider passes a fetched web document to swish for indexing the spider can call a simple subroutine in the spider's configuration file passing the document and its content type. The subroutine can then look at the content type and decide if the document needs to be filtered.

For example, when processing a document of type "application/msword" the call-back subroutine might call the doc2txt.pm perl module, and a document of type "appliation/pdf" could use the pdf2html.pm module. The *prog-bin/SwishSpiderConfig.pl* file shows this usage.

This system works reasonably well, but also means that more work is required to setup the filters. First, you must explicitly check for specific content types and then call the appropriate Perl module, and second, you have to know how each module must be called and how each returns the possibly modified content.

In comes SWISH::Filter.

To make things easier the SWISH::Filter Perl module was created. The idea of this module is that there is one interface used to filter all types of documents. So instead of checking for specific types of content you just pass the content type and the document to the SWISH::Filter module and it returns a new content type and document if it was filtered. The filters that do the actual work are designed with a standard interface and work like filter "plug-ins". Adding new filters means just downloading the filter to a directory and no changes are needed to the spider's configuation file. Download a filter for Postscript and next time you run indexing your Postscript files will be indexed.

Since the filters are standardized, hopefully when you have the need to filter documents of a specific type there will already be a filter ready for your use.

Now, note that the perl modules may or may not do the actual conversion of a document. For example, the PDF conversion module calls the pdfinfo and pdftotext programs. Those programs (part of the Xpfd package) must be installed separately from the filters.

The SwishSpiderConfig.pl examle spider configuration file shows how to use the SWISH::Filter module for filtering. This file is installed at $prefix/share/doc/swish-e/examples/prog-bin, where `$prefix` is normally /usr/local on unix-type machines.

The SWISH::Filter method of filtering can also be used with the -S http method of indexing. By default the *swishspider* program (the Perl helper script that fetches documents from the web) will attempt to use the SWISH::Filter module if it can be found in Perls library path. This path is set automatically for spider.pl but not for swishspider (because it would slow down a method that's already slow and spider.pl is recommended over the -S http method).

Therefore, all that's required to use this system with -S http is setting the `@INC` array to point to the filter directory.

For example, if the swish-e distribution was unpacked into ~/swish-e:

```
    PERL5LIB=~/swish-e/filters swish-e -c conf -S http
```

will allow the -S http method to make use of the SWISH::Filter module.

Note that if you are not using the SWISH::Filter module you may wish to edit the *swishspider* program and disable the use of the SWISH::Filter module using this setting:

```
    use constant USE_FILTERS  => 0;  # disable SWISH::Filter
```

This prevents the program from attempting to use the SWISH::Filter module for every non-text URL that is fetched. Of course, if you are concerned with indexing speed you should be using the -S prog method with spider.pl instead of -S http.

If you are not spidering, but you still want to make use of the SWISH::Filter module for filtering you can use the DirTree.pl program (in $prefix/lib/swish-e). This is a simple program that traverses the file system and uses SWISH::Filter for filtering.

Here's two examples of how to run a filter program, one using Swish-e's FileFilter directive, another using a prog input method program. See the *SwishSpiderConfig.pl* file for an example of using the SWISH::Filter module.

These filters simply use the program `/bin/cat` as a filter and only indexes .html files.

First, using the FileFilter method, here's the entire configuration file (swish.conf):

```
    IndexDir .
    IndexOnly .html
    FileFilter .html "/bin/cat"   "'%p'"
```

and index with the command

```
    swish-e -c swish.conf -v 1
```

Now, the same thing with using the `-S prog` document source input method and a Perl program called catfilter.pl. You can see that's it's much more work than using the FileFilter method above, but provides a place to do additional processing. In this example, the prog method is only slightly faster. But if you needed a perl script to run as a FileFilter then prog will be significantly faster.

```
    #!/usr/local/bin/perl -w
    use strict;
    use File::Find;  # for recursing a directory tree
```

```
    $/ = undef;
    find(
        { wanted => \&wanted, no_chdir => 1, },
        '.',
    );
    sub wanted {
        return if -d;
        return unless /\.html$/;

        my $mtime  = (stat)[9];

        my $child = open( FH, '-|' );
        die "Failed to fork $!" unless defined $child;
        exec '/bin/cat', $_ unless $child;

        my $content = <FH>;
        my $size = length $content;

        print <<EOF;
    Content-Length: $size
    Last-Mtime: $mtime
    Path-Name: $_

    EOF

        print <FH>;
    }
```

And index with the command:

```
    swish-e -S prog -i ./catfilter.pl -v 1
```

This example will probably not work under Windows due to the '-|' open. A simple piped open may work just as well:

That is, replace:

```
    my $child = open( FH, '-|' );
    die "Failed to fork $!" unless defined $child;
    exec '/bin/cat', $_ unless $child;
```

with this:

```
    open( FH, "/bin/cat $_ |" ) or die $!;
```

Perl will try to avoid running the command through the shell if meta characters are not passed to the open. See `perldoc -f open` for more information.

7.1.2.15 Eh, but I just want to know how to index PDF documents!

See the examples in the *conf* directory and the comments in the *SwishSpiderConfig.pl* file.

See the previous question for the details on filtering. The method you decide to use will depend on how fast you want to index, and your comfort level with using Perl modules.

Regardless of the filtering method you use you will need to install the Xpdf packages available from http://www.foolabs.com/xpdf/.

7.1.2.16 I’m using Windows and can’t get Filters or the prog input method to work!

Both the `-S prog` input method and filters use the `popen()` system call to run the external program. If your external program is, for example, a perl script, you have to tell Swish-e to run perl, instead of the script. Swish-e will convert forward slashes to backslashes when running under Windows.

For example, you would need to specify the path to perl as (assuming this is where perl is on your system):

```
    IndexDir e:/perl/bin/perl.exe
```

Or run a filter like:

```
    FileFilter .foo e:/perl/bin/perl.exe ’myscript.pl "%p"’
```

It’s often easier to just install Linux.

7.1.2.17 How do I index non-English words?

Swish-e indexes 8-bit characters only. This is the ISO 8859-1 Latin-1 character set, and includes many non-English letters (and symbols). As long as they are listed in WordCharacters they will be indexed.

Actually, you probably can index any 8-bit character set, as long as you don’t mix character sets in the same index and don’t use libxml2 for parsing (see below).

The TranslateCharacters directive (SWISH-CONFIG) can translate characters while indexing and searching. You may specify the mapping of one character to another character with the TranslateCharacters directive.

`TranslateCharacters :ascii7:` is a predefined set of characters that will translate eight-bit characters to ascii7 characters. Using the `:ascii7:` rule will, for example, translate "Ääç" to "aac". This means: searching "Çelik", "çelik" or "celik" will all match the same word.

Note: When using libxml2 for parsing, parsed documents are converted internally (within libxml2) to UTF-8. This is converted to ISO 8859-1 Latin-1 when indexing. In cases where a string can not be converted from UTF-8 to ISO 8859-1 (because it contains non 8859-1 characters), the string will be sent to Swish-e in UTF-8 encoding. This will results in some words indexed incorrectly. Setting `Parser-`

`WarningLevel` to 1 or more will display warnings when UTF-8 to 8859-1 conversion fails.

7.1.2.18 Can I add/remove files from an index?

Try building swish-e with the `--enable-incremental` option.

The rest of this FAQ applies to the default swish-e format.

Swish-e currently has no way to add or remove items from its index. But, Swish-e indexes so quickly that it's often possible to reindex the entire document set when a file needs to be added, modified or removed. If you are spidering a remote site then consider caching documents locally compressed.

Incremental additions can be handled in a couple of ways, depending on your situation. It's probably easiest to create one main index every night (or every week), and then create an index of just the new files between main indexing jobs and use the `-f` option to pass both indexes to Swish-e while searching.

You can merge the indexes into one index (instead of using -f), but it's not clear that this has any advantage over searching multiple indexes.

How does one create the incremental index?

One method is by using the `-N` switch to pass a file path to Swish-e when indexing. It will only index files that have a last modification date `newer` than the file supplied with the `-N` switch.

This option has the disadvantage that Swish-e must process every file in every directory as if they were going to be indexed (the test for `-N` is done last right before indexing of the file contents begin and after all other tests on the file have been completed) -- all that just to find a few new files.

Also, if you use the Swish-e index file as the file passed to `-N` there may be files that were added after indexing was started, but before the index file was written. This could result in a file not being added to the index.

Another option is to maintain a parallel directory tree that contains symlinks pointing to the main files. When a new file is added (or changed) to the main directory tree you create a symlink to the real file in the parallel directory tree. Then just index the symlink directory to generate the incremental index.

This option has the disadvantage that you need to have a central program that creates the new files that can also create the symlinks. But, indexing is quite fast since Swish-e only has to look at the files that need to be indexed. When you run full indexing you simply unlink (delete) all the symlinks.

Both of these methods have issues where files could end up in both indexes, or files being left out of an index. Use of file locks while indexing, and hash lookups during searches can help prevent these problems.

7.1.2.19 I run out of memory trying to index my files.

It’s true that indexing can take up a lot of memory! Swish-e is extremely fast at indexing, but that comes at the cost of memory.

The best answer is install more memory.

Another option is use the `-e` switch. This will require less memory, but indexing will take longer as not all data will be stored in memory while indexing. How much less memory and how much more time depends on the documents you are indexing, and the hardware that you are using.

Here’s an example of indexing all .html files in /usr/doc on Linux. This first example is *without* `-e` and used about 84M of memory:

```
    270279 unique words indexed.
    23841 files indexed.  177640166 total bytes.
    Elapsed time: 00:04:45 CPU time: 00:03:19
```

This is *with* `-e`, and used about 26M or memory:

```
    270279 unique words indexed.
    23841 files indexed.  177640166 total bytes.
    Elapsed time: 00:06:43 CPU time: 00:04:12
```

You can also build a number of smaller indexes and then merge together with `-M`. Using `-e` while merging will save memory.

Finally, if you do build a number of smaller indexes, you can specify more than one index when searching by using the `-f` switch. Sorting large results sets by a property will be slower when specifying multiple index files while searching.

7.1.2.20 "too many open files" when indexing with -e option

Some platforms report "too many open files" when using the -e economy option. The -e feature uses many temporary files (something like 377) plus the index files and this may exceed your system’s limits.

Depending on your platform you may need to set "ulimit" or "unlimit".

For example, under Linux bash shell:

```
  $ ulimit -n 1024
```

Or under an old Sparc

```
  % unlimit openfiles
```

7.1.2.21 My system admin says Swish-e uses too much of the CPU!

That's a good thing! That expensive CPU is supposed to be busy.

Indexing takes a lot of work -- to make indexing fast much of the work is done in memory which reduces the amount of time Swish-e is waiting on I/O. But, there's two things you can try:

The `-e` option will run Swish-e in economy mode, which uses the disk to store data while indexing. This makes Swish-e run somewhat slower, but also uses less memory. Since it is writing to disk more often it will be spending more time waiting on I/O and less time in CPU. Maybe.

The other thing is to simply lower the priority of the job using the `nice(1)` command:

```
    /bin/nice -15 swish-e -c search.conf
```

If concerned about searching time, make sure you are using the -b and -m switches to only return a page at a time. If you know that your result sets will be large, and that you wish to return results one page at a time, and that often times many pages of the same query will be requested, you may be smart to request all the documents on the first request, and then cache the results to a temporary file. The perl module File::Cache makes this very simple to accomplish.

7.1.3 Spidering

7.1.3.1 How can I index documents on a web server?

If possible, use the file system method `-S fs` of indexing to index documents in you web area of the file system. This avoids the overhead of spidering a web server and is much faster. (`-S fs` is the default method if `-S` is not specified).

If this is impossible (the web server is not local, or documents are dynamically generated), Swish-e provides two methods of spidering. First, it includes the http method of indexing `-S http`. A number of special configuration directives are available that control spidering (see Directives for the HTTP Access Method Only). A perl helper script (swishspider) is included in the *src* directory to assist with spidering web servers. There are example configurations for spidering in the *conf* directory.

As of Swish-e 2.2, there's a general purpose "prog" document source where a program can feed documents to it for indexing. A number of example programs can be found in the `prog-bin` directory, including a program to spider web servers. The provided spider.pl program is full-featured and is easily customized.

The advantage of the "prog" document source feature over the "http" method is that the program is only executed one time, where the swishspider.pl program used in the "http" method is executed once for every document read from the web server. The forking of Swish-e and compiling of the perl script can be quite expensive, time-wise.

The other advantage of the `spider.pl` program is that it's simple and efficient to add filtering (such as for PDF or MS Word docs) right into the spider.pl's configuration, and it includes features such as MD5 checks to prevent duplicate indexing, options to avoid spidering some files, or index but avoid spidering. And since it's a perl program there's no limit on the features you can add.

7.1.3.2 Why does swish report "./swishspider: not found"?

Does the file *swishspider* exist where the error message displays? If not, either set the configuration option SpiderDirectory to point to the directory where the *swishspider* program is found, or place the *swishspider* program in the current directory when running swish-e.

If you are running Windows, make sure "perl" is in your path. Try typing *perl* from a command prompt.

If you not running windows, make sure that the shebang line (the first line of the swishspider program that starts with #!) points to the correct location of perl. Typically this will be */usr/bin/perl* or */usr/local/bin/perl*. Also, make sure that you have execute and read permissions on *swishspider*.

The *swishspider* perl script is only used with the -S http method of indexing.

7.1.3.3 I'm using the spider.pl program to spider my web site, but some large files are not indexed.

The `spider.pl` program has a default limit of 5MB file size. This can be changed with the `max_size` parameter setting. See `perldoc spider.pl` for more information.

7.1.3.4 I still don't think all my web pages are being indexed.

The *spider.pl* program has a number of debugging switches and can be quite verbose in telling you what's happening, and why. See `perldoc spider.pl` for instructions.

7.1.3.5 Swish is not spidering Javascript links!

Swish cannot follow links generated by Javascript, as they are generated by the browser and are not part of the document.

7.1.3.6 How do I spider other websites and combine it with my own (filesystem) index?

You can either merge `-M` two indexes into a single index, or use `-f` to specify more than one index while searching.

You will have better results with the `-f` method.

7.1.4 Searching

7.1.4.1 How do I limit searches to just parts of the index?

If you can identify "parts" of your index by the path name you have two options.

The first options is by indexing the document path. Add this to your configuration:

```
MetaNames swishdocpath
```

Now you can search for words or phrases in the path name:

```
swish-e -w 'foo AND swishdocpath=(sales)'
```

So that will only find documents with the word "foo" and where the file's path contains "sales". That might not works as well as you like, though, as both of these paths will match:

```
/web/sales/products/index.html
/web/accounting/private/sales_we_messed_up.html
```

This can be solved by searching with a phrase (assuming "/" is not a WordCharacter):

```
swish-e -w 'foo AND swishdocpath=("/web/sales/")'
swish-e -w 'foo AND swishdocpath=("web sales")'  (same thing)
```

The second option is a bit more powerful. With the ExtractPath directive you can use a regular expression to extract out a sub-set of the path and save it as a separate meta name:

```
MetaNames department
ExtractPath department regex !^/web/([^/]+).+$!$1/
```

Which says match a path that starts with "/web/" and extract out everything after that up to, but not including the next "/" and save it in variable $1, and then match everything from the "/" onward. Then replace the entire matches string with $1. And that gets indexed as meta name "department".

Now you can search like:

```
swish-e -w 'foo AND department=sales'
```

and be sure that you will only match the documents in the /www/sales/* path. Note that you can map completely different areas of your file system to the same metaname:

```
    # flag the marketing specific pages
    ExtractPath department regex !^/web/(marketing|sales)/.+$!marketing/
    ExtractPath department regex !^/internal/marketing/.+$!marketing/
```

```
    # flag the technical departments pages
    ExtractPath department regex !^/web/(tech|bugs)/.+$!tech/
```

Finally, if you have something more complicated, use `-S prog` and write a perl program or use a filter to set a meta tag when processing each file.

7.1.4.2 How is ranking calculated?

The `swishrank` property value is calculated based on which Ranking Scheme (or algorithm) you have selected. In this discussion, any time the word **fancy** is used, you should consult the actual code for more details. It is open source, after all.

Things you can do to affect ranking:

MetaRankBias

You may configure your index to bias certain metaname values more or less than others. See the MetaRankBias configuration option in the SWISH-CONFIG manpage.

IgnoreTotalWordCountWhenRanking

Set to 1 (default) or 0 in your config file. See the SWISH-CONFIG manpage. **NOTE:** You must set this to 0 to use the IDF Ranking Scheme.

structure

Each term’s position in each HTML document is given a structure value based on the context in which the word appears. The structure value is used to artificially inflate the frequency of each term in that particular document. These structural values are defined in *config.h*:

```
#define RANK_TITLE             7
#define RANK_HEADER            5
#define RANK_META              3
#define RANK_COMMENTS          1
#define RANK_EMPHASIZED        0
```

For example, if the word `foo` appears in the title of a document, the Scheme will treat that document as if `foo` appeared 7 additional times.

All Schemes share the following characteristics:

AND searches

The rank value is averaged for all AND'd terms. Terms within a set of parentheses () are averaged as a single term (this is an acknowledged weakness and is on the TODO list).

OR searches

The rank value is summed and then doubled for each pair of OR'd terms. This results in higher ranks for documents that have multiple OR'd terms.

scaled rank

After a document's raw rank score is calculated, a final rank score is calculated using a fancy `log()` function. All the documents are then scaled against a base score of 1000. The top-ranked document will therefore always have a `swishrank` value of 1000.

Here is a brief overview of how the different Schemes work. The number in parentheses after the name is the value to invoke that scheme with `swish-e -R` or `RankScheme()`.

Default (0)

The default ranking scheme considers the number of times a term appears in a document (frequency), the MetaRankBias and the structure value. The rank might be summarized as:

```
DocRank = Sum of ( structure + metabias )
```

Consider this output with the DEBUG_RANK variable set at compile time:

```
Ranking Scheme: 0
Word entry 0 at position 6 has struct 7
Word entry 1 at position 64 has struct 41
Word entry 2 at position 71 has struct 9
Word entry 3 at position 132 has struct 9
Word entry 4 at position 154 has struct 9
Word entry 5 at position 423 has struct 73
Word entry 6 at position 541 has struct 73
Word entry 7 at position 662 has struct 73
File num: 1104.  Raw Rank: 21.  Frequency: 8 scaled rank: 30445
 Structure tally:
 struct 0x7 = count of 1 ( HEAD TITLE FILE ) x rank map of 8 = 8
```

```
 struct 0x9 = count of 3 ( BODY FILE ) x rank map of 1 = 3
```

```
 struct 0x29 = count of 1 ( HEADING BODY FILE ) x rank map of 6 = 6
```

```
 struct 0x49 = count of 3 ( EM BODY FILE ) x rank map of 1 = 3
```

Every word instance starts with a base score of 1. Then for each instance of your word, a running sum is taken of the structural value of that word position plus any bias you've configured. In the example above, the raw rank is `1 + 8 + 3 + 6 + 3 = 21`.

Consider this line:

```
struct 0x7 = count of 1 ( HEAD TITLE FILE ) x rank map of 8 = 8
```

That means there was one instance of our word in the title of the file. It's context was in the <head> tagset, inside the <title>. The <title> is the most specific structure, so it gets the RANK_TITLE score: 7. The base rank of 1 plus the structure score of 7 equals 8. If there had been two instances of this word in the title, then the score would have been `8 + 8 = 16`.

IDF (1)

IDF is short for Inverse Document Frequency. That's fancy ranking lingo for taking into account the total frequency of a term across the entire index, in addition to the term's frequency in a single document. IDF ranking also uses the relative density of a word in a document to judge its relevancy. Words that appear more often in a doc make that doc's rank higher, and longer docs are not weighted higher than shorter docs.

The IDF Scheme might be summarized as:

```
DocRank = Sum of ( density * idf * ( structure + metabias ) )
```

Consider this output from DEBUG_RANK:

```
Ranking Scheme: 1
File num: 1104  Word Score: 1  Frequency: 8  Total files: 1451
Total word freq: 108   IDF: 2564
Total words: 1145877   Indexed words in this doc: 562
Average words: 789   Density: 1120    Word Weight: 28716
Word entry 0 at position 6 has struct 7
Word entry 1 at position 64 has struct 41
Word entry 2 at position 71 has struct 9
Word entry 3 at position 132 has struct 9
Word entry 4 at position 154 has struct 9
Word entry 5 at position 423 has struct 73
Word entry 6 at position 541 has struct 73
Word entry 7 at position 662 has struct 73
Rank after IDF weighting: 574321
scaled rank: 132609
 Structure tally:
 struct 0x7 = count of  1 ( HEAD TITLE FILE ) x rank map of 8 = 8
```

```
 struct 0x9 = count of  3 ( BODY FILE ) x rank map of 1 = 3
```

```
    struct 0x29 = count of  1 ( HEADING BODY FILE ) x rank map of 6 = 6

    struct 0x49 = count of  3 ( EM BODY FILE ) x rank map of 1 = 3
```

It is similar to the default Scheme, but notice how the total number of files in the index and the total word frequency (as opposed to the document frequency) are both part of the equation.

Ranking is a complicated subject. SWISH-E allows for more Ranking Schemes to be developed and experimented with, using the -R option (from the swish-e command) and the RankScheme (see the API documentation). Experiment and share your findings via the discussion list.

7.1.4.3 How can I limit searches to the title, body, or comment?

Use the `-t` switch.

7.1.4.4 I can't limit searches to title/body/comment.

Or, *I can't search with meta names, all the names are indexed as "plain".*

Check in the config.h file if #define INDEXTAGS is set to 1. If it is, change it to 0, recompile, and index again. When INDEXTAGS is 1, ALL the tags are indexed as plain text, that is you index "title", "h1", and so on, AND they loose their indexing meaning. If INDEXTAGS is set to 0, you will still index meta tags and comments, unless you have indicated otherwise in the user config file with the IndexComments directive.

Also, check for the UndefinedMetaTags setting in your configuration file.

7.1.4.5 I've tried running the included CGI script and I get a "Internal Server Error"

Debugging CGI scripts are beyond the scope of this document. Internal Server Error basically means "check the web server's log for an error message", as it can mean a bad shebang (#!) line, a missing perl module, FTP transfer error, or simply an error in the program. The CGI script *swish.cgi* in the *example* directory contains some debugging suggestions. Type `perldoc swish.cgi` for information.

There are also many, many CGI FAQs available on the Internet. A quick web search should offer help. As a last resort you might ask your webadmin for help...

7.1.4.6 When I try to view the swish.cgi page I see the contents of the Perl program.

Your web server is not configured to run the program as a CGI script. This problem is described in `perldoc swish.cgi`.

7.1.4.7 How do I make Swish-e highlight words in search results?

Short answer:

Use the supplied swish.cgi or search.cgi scripts located in the *example* directory.

Long answer:

Swish-e can't because it doesn't have access to the source documents when returning results, of course. But a front-end program of your creation can highlight terms. Your program can open up the source documents and then use regular expressions to replace search terms with highlighted or bolded words.

But, that will fail with all but the most simple source documents. For HTML documents, for example, you must parse the document into words and tags (and comments). A word you wish to highlight may span multiple HTML tags, or be a word in a URL and you wish to highlight the entire link text.

Perl modules such as HTML::Parser and XML::Parser make word extraction possible. Next, you need to consider that Swish-e uses settings such as WordCharacters, BeginCharacters, EndCharacters, IgnoreFirstChar, and IgnoreLast, char to define a "word". That is, you can't consider that a string of characters with white space on each side is a word.

Then things like TranslateCharacters, and HTML Entities may transform a source word into something else, as far as Swish-e is concerned. Finally, searches can be limited by metanames, so you may need to limit your highlighting to only parts of the source document. Throw phrase searches and stopwords into the equation and you can see that it's not a trivial problem to solve.

All hope is not lost, thought, as Swish-e does provide some help. Using the `-H` option it will return in the headers the current index (or indexes) settings for WordCharacters (and others) required to parse your source documents as it parses them during indexing, and will return a "Parsed Words:" header that will show how it parsed the query internally. If you use fuzzy indexing (word stemming, soundex, or metaphone) then you will also need to stem each word in your document before comparing with the "Parsed Words:" returned by Swish-e.

The Swish-e stemming code is available either by using the Swish-e Perl module (SWISH::API) or the C library (included with the swish-e distribution), or by using the SWISH::Stemmer module available on CPAN. Also on CPAN is the module Text::DoubleMetaphone. Using SWISH::API probably provides the best stemming support.

7.1.4.8 Do filters effect the performance during search?

No. Filters (FileFilter or via "prog" method) are only used for building the search index database. During search requests there will be no filter calls.

7.1.5 I have read the FAQ but I still have questions about using Swish-e.

The Swish-e discussion list is the place to go. http://swish-e.org/. Please do not email developers directly. The list is the best place to ask questions.

Before you post please read *QUESTIONS AND TROUBLESHOOTING* located in the INSTALL page. You should also search the Swish-e discussion list archive which can be found on the swish-e web site.

In short, be sure to include in the following when asking for help.

- **The swish-e version (./swish-e -V)**
- **What you are indexing (and perhaps a sample), and the number of files**
- **Your Swish-e configuration file**
- **Any error messages that Swish-e is reporting**

7.2 Document Info

$Id: SWISH-FAQ.pod,v 1.36 2004/10/04 22:49:35 whmoseley Exp $

.

8 SWISH-BUGS - List of bugs known in Swish-e

8.1 DESCRIPTION

This file contains a list of bugs reported or known in Swish-e. If you find a bug listed here you do not need to report it as a bug. But feel free to bug the developers about it on the Swish-e discussion list.

8.2 Bugs in Swish-e version 2.4

- **Stopwords not removed from query with Soundex**

  In dev version 2.5.2 noticed that stopwords are not removed from the query when using Soundex. The plan is to rewrite the parser soon... (July 2004)

- **Wild card searching can be very slow**

  Wild card searching needs to be optimized.

  Here's a three letter search:

  ```
      $ swish-e -w 'tra*' -m1
      # Number of hits: 99952
      # Search time: 5.424 seconds
  ```

  Two letters:

  ```
      $ swish-e -w 'tr*' -m1
      # Number of hits: 100000
      # Search time: 10.563 seconds
  ```

  Single letter search:

  ```
      $ swish-e -w 't*' -m1
      # Number of hits: 100000
      # Search time: 510.939 seconds
  ```

  and used about 280MB or RAM.

  This is a potential for a DoS attack. If you have a large index you may wish to filter out single character wild cards.

- **Character Encodings**

  The XML parser (Expat) returns UTF-8 data to swish-e. Therefore, the XML parser should only be used for parsing US-ASCII encoded text.

  The XML2 & HTML2 parsers (Libxml2) converts characters from UTF-8 to 8859-1 encodings before indexing and writing properties. Indexing non-8859-1 data may result in invalid character mappings.

These issues will be resolved soon.

- Phrase search failes with DoubleMetaphone

  DoubleMetaphone searching can produce two search words for a single query word. The words are expanded to (word1 OR word2), but that fails in a phrase query: "some phrase (word1 or word2) here"

  swish-e query parser is due for a rewrite, and this could be resolved then.

  ```
      Reported: August 20, 2002 - moseley
  ```

- Merging

  merge.c does not check for matching stopwords or buzzwords in each index.

  History:

  ```
      Reported: September 3, 2002 - moseley
  ```

- ResultSortOrder

  ResultSort order is not used (and is not documented). The problem is that the data passed to `Compare_Properties()` does not have access to the ResultSortOrder table.

History:

```
    Reported: September 3, 2002 - moseley
```

8.3 Document Info

$Id: SWISH-BUGS.pod,v 1.9 2004/10/04 22:49:34 whmoseley Exp $

.

9 Proposed changes for Swish-e 3.0

9.1 Overview

This pages is intended to give users of Swish-e an idea of the changes to come, to foster discussion of the direction of Swish-e, and a place where developers can map out new ideas.

None of this is written in stone. Any of the developers can write their ideas in this document, but that doesn’t mean it will actually happen ;).

9.2 UTF-8 support

Supporting Unicode basically requires a full re-write of all the code.

9.3 drop expat-based parsers, require libxml2

This might simplify the code somewhat as well.

9.4 Support Incremental Indexing

The Swish-e index structure currently makes it difficult to do incremental indexing, range limiting, and presents limits to indexing due to memory requirements. A database may solve some of these issues, at possibly a cost of performance.

Swish-e has been linked with Berkeley DB. Although much slower in indexing, this may allow incremental indexing. Currently, the idea is to offer both database backends.

UPDATE: Mon Nov 8 15:07:59 CST 2004 (karman@cray.com)

This feature is in the 2.5 branch already. What kind of requirements do we have to label it ’stable’?

9.5 Split code into Search and Indexing code

There may be a small benefit from creating a smaller search-only program. CGI scripts may be faster, and the code would be smaller for those that want to embed Swish-e in to other applications.

Currently, linking libswish-e into a program adds about 720K. Not real significant, but it could be if a number of processes are running with Swish-e. Another option is to build libswish-e as a shared library.

UPDATE: Mon Nov 8 15:09:12 CST 2004 (karman@cray.com)

This seems done in the 2.4 release. Is that true?

9.6 Switch to Content-Types

Moseley: Dec 28, 2000

I’m wondering if it might be smart to switch from the current "Document Types" to Content-Types. Currently, Swish-e know how to parse three types of documents TXT, HTML, and XML. There’s currently two new configuration directives DefaultContents and IndexContents that map file extensions to one of the three types. This doesn’t really work when spidering since it’s the content-type that describes the document and not the file extension.

It’s an issue that can wait, but I’m concerned about backward compatiblity before people start using the IndexContents and DefaultContents config directives and then we change to content-type in the future. There’s probably not that many people using those, but it might be work noting in the documentation that it will change, if we agree.

The main reason to use content-type instead is for http processing where you can’t depend on the file extension to determine the document type, so with http we have to use content-type to determine how to deal with the file. This is somewhat moot, as mapping can now be done with -S prog.

I’d propose that Swish-e uses a mime.types file to map from extension to content-type. You could add or override mappings in the config file:

```
    AddType text/plain .doc .log
```

```
    DefaultType text/html  # like DefaultContents currently
```

The file source "plug-in" (whatever that ends up being) would return a content-type, but if not returned then Swish-e would map the type from the file name using the mime.types file or any AddType directives.

Again, internally Swish-e only knows about text/[TXT|HTML|XML], so there should be a way to map other types, otherwise Swish-e might ignore the file. We could continue to use the three type names or switch completely to content-types.

For example, if we continued to use [TXT|HTML|XML]

```
     MapType TXT  text/directory text/logfile
     MapType HTML text/html
```

Or maybe just extend the current directives

```
     IndexContents HTML .htm .html text/html
```

Where the content-type would have precedence over the file extensions.

This would tell Swish-e that those types are handled by those internal handlers.

Then as I've mentioned before, you might specify filters as such

```
    FilterDocument application/msword /path/to/word-to-text
```

And word-to-text would convert to text and return one of the three content-types that Swish-e knows how to parse, or a different content type if were to chain filters.

9.7 Enhanced the PropertyNames directive

Moseley: Updated Jan 13, 2001

If the PropertyNames directive was enhanced to be able to limit the number of characters stored, optionally extract text from HTML, and was able to define what type of docs (text, XML, HTML) it applied to, then the existing PropertyNames feature would work like the new StoreDescription feature but be useful for more than just one use.

I'm not clear how to enhance the syntax of Properties and/or Metanames, but here's some ideas. Rainer suggested that an xml-type of format might be best and commonly understood. That's a good idea. Below are some older ideas that I had. But you will get the idea...

The metaname structure could have flags for properties:

```
     1 - limiting to a length
     2 - stripping HTML
     3 - encoding HTML entities on output
```

Oct 9, 2001 - The code is now in Swish-e to limit a string property to a length. The stripping of HTML is an issue for discussion. And encoding entities on output should be a result_outpu.c issue.

UPDATE: Mon Nov 8 15:13:26 CST 2004 (karman@cray.com)

Is this fully supported in 2.4?

9.8 Apache/XML style configuration

This would be to allow some directives to be set per directory, or perl file extenstion (or content-type).

9.9 Document Info

$Id: SWISH-3.0.pod,v 1.7 2004/11/08 21:19:35 karman Exp $

.

10 SWISH-LIBRARY - Interface to the Swish-e C library

10.1 What is the Swish-e C library?

The C library in an interface to the Swish-e search code. It provides a way to embed Swish-e into your applications. This API is based on Swish-e version 2.3.

Note: This is a NEW API as of Swish-e version 2.3. The C language interface has changed as has the perl interface to Swish-e. The new Perl interface is the SWISH::API module and is included with the Swish-e distribution. The old SWISHE perl module has been rewritten to work with the new API. The SWISHE perl module is no longer included with the Swish-e distribution, but can be downloaded from the Swish-e web site.

The advantage of the library is that the index files or files can be opened one time and many queries made on the open index. This saves the startup time required to fork and run the swish-e binary, and the expensive time of opening up the index file. Some benchmarks have shown a three fold increase in speed.

The downside is that your program now has more code and data in it (the index tables can use quite a bit of memory), and if a fatal error happens in swish it will bring down your program. These are things to think about, especially if embedding swish into a web server such as Apache where there are many processes serving requests.

The best way to learn about the library is to look at two files included with the Swish-e distribution that make use of the library.

src/libtest.c

This file gives a basic overview of linking a C program with the Swish-e library. Not all available functions are used in that example, but it should give you a good overview of building a C program with swish-e.

To build and run libtest chdir to the src directory and run the commands:

```
$ make libtest
$ ./libtest [optional name of index file]
```

You will be prompted for the search words. The default index used is *index.swish-e*. This can be overridden by placing a list of index files in a quote-protected string.

```
$ ./libtest ’index1 index2 index3’
```

perl/API.xs

The *API.xs* file is a Perl "xsub" interface to the C library and is part of the SWISH::API Perl module. This is an object-oriented interface to the Swish-e library and demonstrates how the various search "objects" are created by C calls and how they are destroyed when no longer needed.

10.2 Installing the Swish-e library

The Swish-e library is installed when you run "make install" when building Swish-e. No extra installation steps are required.

The library consists of a header file "swish-e.h" and a library "libswish-e.*" that can either be a static or shared library depending on your platform.

10.3 Library Overview

When you first attach to an index file (or index files) you are returned a "swish handle". From the handle you create one or more "search objects" which holds the parameters to query the index, such as the query string, sort order, search phrase delimiter, limit parameters and HTML structure bits. The "object" is really just a pointer to a C structure, but it’s helpful to think of it as an object that data and functionality associated with it.

The search object is used to query the index. A query returns a "results object". The results object holds the number of hits, the parsed query per index, and the result set. The results object keeps track of the current position in the result set. You may "seek" to a specific record within the result set (useful for displaying a page of results).

Finally, a result object represents a single result from the result list. A result object provides access to the result’s properties (such as file name, rank, etc.).

In addition to results, there are functions available to access the header values stored in the index file, functions to check and report errors, and a few utility functions.

10.4 Available Functions

Below is the list of available function included in the Swish-e C language API.

These functions (and typedefs) are defined in the *swish-e.h* header file. The common objects (e.g. structures) used are:

```
    SW_HANDLE  - swish handle that associates with an index file
    SW_SEARCH  - search "object" that holds search parameters
    SW_RESULTS - results "object" that holds a result set
    SW_RESULT  - a single result used for accessing the result’s properties
    SW_FUZZYWORD - used for fuzzy (stemming) word conversion
```

10.4.1 Searching

SW_HANDLE SwishInit(char *IndexFiles);

This functions opens and reads the header info of the index files included in IndexFiles string. The string should contain a space-separated list of index files.

```
SW_HANDLE myhandle;
myhandle = SwishInit("file1.idx");
```

Typically you will open a handle at the beginning of your program and use it to make multiple queries on an index.

This function will always return a swish handle. You must check for errors, and on error free the memory used by the handle, or abort.

Here’s an example of aborting:

```
SW_HANDLE swish_handle;
swish_handle = SwishInit("file1.idx file2.idx");
if ( SwishError( swish_handle ) )
    SwishAbortLastError( swish_handle );
```

And here’s an example of catching the error:

```
SW_HANDLE swish_handle;
swish_handle = SwishInit("file1.idx file2.idx");
if ( SwishError( swish_handle ) )
{
    printf("Failed to connect to swish. %s\n", SwishErrorString( swish_handle ) );
    SwishClose( swish_handle );  /* free the memory used */
    return 0;
}
```

You may have more than one handle active at a time.

Swish-e will not tell you if the index file changes on disk (such as after reindexing). In a persistent environment (e.g. mod_perl) the calling program should check to see if the index file has changed on disk. A common way to do this is to store the inode number before opening the index `file(s),` and then stat the file name every so often and reopen the index files if the inode number changes.

void SwishClose(SW_HANDLE handle);

This function closes and frees the memory of a Swish handle. Every swish handle should be freed when done searching the index. Failing to close the handle will result in a memory leak.

SW_SEARCH New_Search_Object(SW_HANDLE handle, const char *query);

Returns a new search "object". The search object holds the parameters used for searching an index. A single search object can be used to query the index multiple times. The available settings listed below are "sticky" in that they remain set on the search object until change.

void SwishSetStructure(SW_SEARCH srch, int structure);

Sets the "structure" flag in the search object. The structure flag is used to limit searches to parts of HTML files (such as to the title or headers). The default is to not limit. This provides the functionality of the -H command line switch.

void SwishPhraseDelimiter(SW_SEARCH srch, char delimiter);

Sets the phrase delimiter character. The default is double-quotes.

void SwishSetSort(SW_SEARCH srch, char *sort);

Sets the sort order of the results. This is the same as the -s switch used with the swish-e binary.

void SwishSetQuery(SW_SEARCH srch, char *query);

Sets the query string in the search object. This typically is not needed since it can be set when creating the search object or when executing a query.

void SwishSetSearchLimit(SW_SEARCH srch, char *propertyname, char *low, char *hi);

Sets the limit parameters for a search. Provides the same functionality as the -L command line switch. You may specify a range of property values that search results must be within. You may call `SwishSetSearchLimit()` only one time for each property (but can set limits on more than one property at a time).

Unlike the other settings on the search object, once you run a query on the search object you must call `SwishResetSearchLimit()` to change or clear the limit parameters.

void SwishResetSearchLimit(SW_SEARCH srch);

Resets the limits set on a search object set by `SwishSetSearchLimit().`

void Free_Search_Object(SW_SEARCH srch);

Frees the search object. This must be called when done with the search object. Generally, you can reuse a search object for multiple queries so typically you would call this right before calling `SwishClose().`

You may free the search object before freeing and generated results objects.

SW_RESULTS SwishExecute(SW_SEARCH search, const char *query);

Searches the index or indexes based on the parameters in the search object. Returns a results object. See below for functions to access the data stored in the results object.

You should always check for errors after calling `SwishExecute().`

SW_RESULTS SwishQuery(SW_HANDLE, const char *words);

This is a short-cut function that bypasses the creation of a search object (actually, bypasses the need to create and free a search object). This only allows passing in a query string; other search parameters cannot be set. The results are sorted by rank.

You should always check for errors after calling `SwishQuery().`

10.4.2 Reading Results

int SwishHits(SW_RESULTS results);

Returns the number of results in the results object.

SWISH_HEADER_VALUE SwishParsedWords(SW_RESULTS, const char *index_name);

Returns the tokenized query. Words are split by WordCharacters and stopwords are removed. The parsed words are useful for highlighting search terms in your program.

The "index_name" is the name of the index supplied in the `SwishInit()` function call.

Returns a SWISH_HEADER_VALUE union of type SWISH_LIST which is a char **. See src/libtest.c for an example of accessing the strings in this list, but in general you may cast this to a (char **).

SWISH_HEADER_VALUE SwishRemovedStopwords(SW_RESULTS, const char *index_name);

Returns a list of stopwords removed from the input query.

Returns a SWISH_HEADER_VALUE union of type SWISH_LIST which is a char **. See src/libtest.c for an example of accessing the strings in this list, but in general you may cast this to a (char **).

int SwishSeekResult(SW_RESULTS, int position);

Sets the current seek position in the list of results, with position zero being the first record (unlike -b where one is the first result).

Returns the position or a negative number on error.

SW_RESULT SwishNextResult(SW_RESULTS);

Returns the next result, or NULL if not more results are available.

The result object returned does not need to be freed after use (unlike the swish handle, search object, and results object).

const char *SwishResultPropertyStr(SW_RESULT, char *propertyname);

This function is mostly useful for testing as it returns odd results on errors.

Aborts if called with a NULL SW_RESULT object

Returns a string value of the specified property.

Returns the empty string "" if the current result does not have the specified property assigned.

Returns the string "(null)" on invalid property name (i.e. property name is not defined in the index) and sets an error (see below) indicating the invalid property name.

The string returned does not need to be freed, but is only valid for the current result. If you wish to save the string you must copy it locally.

Dates are formatted using the hard-coded format string: "%Y-%m-%d %H:%M:%S" in localtime.

unsigned long SwishResultPropertyULong(SW_RESULT r, char *propertyname);

Returns a numeric property as an unsigned long. Numeric properties are used for both PropertyNamesNumeric and PropertyNamesDate type of properties. Dates are returned as a unix timestamp as reported by the system when the index was created.

Swish-e will abort if called with a NULL SW_RESULT object. Without the SW_RESULT object swish-e cannot set any error codes.

On error returns UMAX_LONG. This is commonly defined in limits.h. Check `SwishError()` (see below) for the type of error.

If `SwishError()` returns false (zero) then it simply means that this result does not have any data for the specified property.

If `SwishError()` returns true (non-zero) then either the propertyname specified is invalid, or the property requested is not a numeric (or date) property (e.g. it's a string property).

See below on how to fetch the specific error message when `SwishError()` is true.

PropValue *getResultPropValue (SW_RESULT r, char *propertyname, int ID);

This is a low-level function to fetch a property regardless of type. This is likely the best function for accessing properties.

Swish-e will abort if called with a NULL SW_RESULT object. Propertyname is the name of the property. ID is the id number of the property, if known. ID is not normally used in the API, but it's purpose is to avoid looking up the property ID for every result displayed.

The return PropValue is a structure that contains a flag to indicate the type, and a union that holds the property value. They flags and structure are defined in swish-e.h.

The property must be copied locally and the returned "PropValue" value must be freed by calling `freeResultPropValue()` to avoid a memory leak.

On error returns NULL. Check `SwishError()` (see below) for the type of error.

If returns NULL but `SwishError()` returns false (zero) then it simply means that this result does not have any data for the specified property.

If `SwishError()` returns true (non-zero) then the property name specified is invalid (i.e. not defined for the index).

See below on how to fetch the specific error message when `SwishError()` is true.

See perl/API.xs for an example on using this function.

void freeResultPropValue(void)

Frees the "PropValue" returned after calling `getResultPropValue().`

void Free_Results_Object(SW_RESULTS results);

Frees the results object (frees the result set). This must be called when done reading the results and before calling `SwishClose().`

10.4.3 Accessing the Index Header Values

Each index file has associated header values that describe the index. These functions provide access to this data. The header data is returned as a union SWISH_HEADER_VALUE, and a pointer to a SWISH_HEADER_TYPE is passed in and the returned value indicates the type of data that is returned. See src/libtest.c and perl/API.xs for examples.

const char **SwishHeaderNames(SW_HANDLE);

Returns the list of possible header names. This list is the same for all index files of a given version of Swish-e. It provides a way to gain access to all headers without having to list them in your program.

const char **SwishIndexNames(SW_HANDLE);

Returns a list of index files opened. This is just the list of index files specified in the `SwishInit()` call. You need the name of the index file to access a specific index’s header values.

SWISH_HEADER_VALUE SwishHeaderValue(SW_HANDLE, const char *index_name, const char *cur_header, SWISH_HEADER_TYPE *type);

Fetches the header value for the given index file, and the header name. The call sets the "type" passed in to the type of value returned.

See src/libtest.c and perl/API.xs for examples.

SWISH_HEADER_VALUE SwishResultIndexValue(SW_RESULT, const char *name, SWISH_HEADER_TYPE *type);

This is like `SwishHeaderValue()` above, but instead of supplying an index file name and a swish handle, supply a result object and the header value is fetched from the result's related index file.

10.4.4 Accessing Property Meta Data

In addition to the pre-defined standard properties, you have the option of adding additional "meta" properties to be indexed and/or added to the list of properties returned with each result. Consult the sections on the MetaNames and PropteryNames directives in the CONFIGURATION FILE for an explanation of how to do this.

These functions provide access to the meta data stored in an index. You can use them to determine what meta/property information is available for an index including all the pre-defined standard properties. See libtest.c for an example.

SWISH_META_LIST SwishMetaList(SW_HANDLE, const char *index_name);

Returns the list of meta entries for the given index file as a null-terminated array of SW_META objects. Use the functions below to extract specific fields from the SW_META structure. Meta's are distinct from properties.

SWISH_META_LIST SwishPropertyList(SW_HANDLE, const char *index_name);

This function is the same as `SwishMetaList()` but it returns an array of properties as opposed to meta objects. Property attributes can be extracted in the same was as meta objects using the functions below.

SWISH_META_LIST SwishResultMetaList(SW_RESULT);

This is like `SwishMetaList()` above but determines the index to use from a result object.

SWISH_META_LIST SwishResultPropertyList(SW_RESULT);

This is like `SwishPropertyList()` above but like `SwishResultMetaList()` uses a result object instead of an index name.

const char *SwishMetaName(SW_META);

Given a SW_META object returned by one of the above, this function will return the meta/property's name. You can use this name to access a property's value for a given as described above.

int SwishMetaType(SW_META);

Get the data type for the given meta/property. Known types are listed in swish-e.h

SwishMetaID(SW_META);

Get the internal ID number for the given meta/property. These id's are unique per index file but are not unique per results.

10.4.5 Checking for Errors

You should check for errors after all calls. The last error is stored in the swish handle object, and is only valid until the next operation (which resets the error flags).

Currently, some errors are flagged as "critical" errors. In these cases you should destroy (by calling the `SwishClose()` function) the current swish handle. If you have other objects in scope (e.g. a search object or results object) destroy those first.

The types of errors that are critical can be seen in src/error.c. Currently the list includes:

```
    Could not open index file
    Unknown index file format
    Index file(s) is empty
    Index file error
    Invalid swish handle
    Invalid results object
```

int SwishError(SW_HANDLE);

This returns true if an error condition exists. It returns the error number, which is a integer less than zero on error. This should be checked before calling any of the other error functions below.

const char *SwishErrorString(SW_HANDLE);

This returns a general text description of the current error.

const char *SwishLastErrorMsg(SW_HANDLE);

In some cases this will return a string with specifics about the current error. For example, `SwishErrorString()` may return "Unknown metaname", but `SwishLastErrorMsg()` will return a string with the name of the unknown metaname.

int SwishCriticalError(SW_HANDLE);

Returns true if the current error condition is a critical error. On critical errors you should free up any current objects and call `SwishClose()` as swish may be in an unstable state.

void SwishAbortLastError(SW_HANDLE);

This is a convenience function that will format and print the last error message, and then abort the program.

void set_error_handle(FILE *where);

Sets where errors and warnings are printed (when printed by swish). For historical reasons, when swish-e first starts up errors and warnings are sent to stdout.

void SwishErrorsToStderr(void);

A convenience method to send errors to stderr instead of stdout.

10.4.6 Utility Functions

const char *SwishWordsByLetter(SWISH * sw, char *indexname, char c);

Returns all the words in the index "indexname" that begin with the letter passed in. Returns NULL if the name of the index file is invalid.

This fuction may change in the future since only 8-bit chars can currently be used.

char * SwsishStemWord(SW_HANDLE sw, char *in_word);

Deprecated

This can be used to convert a word to its stem. It uses only the original Porter Stemmer.

SW_FUZZYWORD SwishFuzzyWord(SW_RESULT r, char *word);

Stems "word" based on the fuzzy mode selected during indexing.

The fuzzy mode used during indexing is stored in the index file. Since each result is linked to a given index file this method allows stemming a word based on it’s index file.

One possible use for this is to highlight search terms in a document summary, which would be based on a given result.

The methods below can be used to access the data returned. The SW_FUZZYWORD object must be freed when done to avoid a memory leak.

const char **SwishFuzzyWordList(SW_FUZZYWORD fw);

Returns a null terminated list of strings returned by the stemmer. In most cases this will be a single string.

Here's an example:

```
        SW_FYZZYWORD fuzzy_word = SwishFuzzyWord( result );
        const char **word_list = SwishFuzzyWordList( fuzzy_word );
        while ( *word_list )
        {
            printf("%s\n", *word_list );
            word_list++;
        }
        SwishFuzzyWordFree( fuzzy_word );
```

If the stemmer does not convert the string (for example attempting to stem numeric data) the word_list will contain the original word. To tell if the stemmer actually stemmed the word check the return value with `SwishFuzzyWordError().`

int SwishFuzzyWordError(SW_FUZZYWORD fw);

This returns zero if the stemming operation was sucessfull, otherwise it returns a value indicating the reason the word was not stemmed. The return values are defined in the swish-e src/stemmer.h file.

Not all stemmers set this value correctly. But since `SwishFuzzyWordList()` will return a valid string regardless of the return value, you can often just ignore this setting. That's what I do.

int SwishFuzzyWordCount(SW_FUZZYWORD fw);

Returns the count of string in the word list available by calling `SwishFuzzyWordList().`

This is normally just one, but in the case of DoubleMetaphone it can be one or two (i.e. DoubleMetaphone can return one or two strings).

const char *SwishFuzzyMode(SW_RESULT r);

Returns the name of the stemmer used for the given result (which is related to an index).

void SwishFuzzyWordFree(SW_FUZZYWORD fw);

Frees the memory used by the SW_FUZZYWORD.

10.5 Bug-Reports

Please report bug reports to the Swish-e discussion group. Feel also free to improve or enhance this feature.

10.6 Author

Original interface: Aug 2000 Jose Ruiz jmruiz@boe.es

Updated: Aug 22, 2002 - Bill Moseley

Interface redesigned for Swish-e version 2.3 Oct 17, 2002 - Bill Moseley

10.7 Document Info

$Id: SWISH-LIBRARY.pod,v 1.12 2004/05/01 01:24:17 whmoseley Exp $

.

11 SWISH::API - Perl interface to the Swish-e C Library

11.1 SYNOPSIS

```
use SWISH::API;

my $swish = SWISH::API->new( 'index.swish-e' );

$swish->AbortLastError
    if $swish->Error;

# A short-cut way to search

my $results = $swish->Query( "foo OR bar" );

# Or more typically
my $search = $swish->New_Search_Object;

# then in a loop
my $results = $search->Execute( $query );

# always check for errors (but aborting is not always necessary)

$swish->AbortLastError
    if $swish->Error;

# Display a list of results

my $hits = $results->Hits;
if ( !$hits ) {
    print "No Results\n";
    return;  /* for example *.
}

print "Found ", $results->Hits, " hits\n";

# Seek to a given page - should check for errors
$results->SeekResult( ($page-1) * $page_size );
```

```
    while ( my $result = $results->NextResult ) {
        printf("Path: %s\n  Rank: %lu\n  Size: %lu\n  Title: %s\n  Index: %s\n  Modified: %s\n  Record #: %lu\n  File   #: %lu\n\n",
            $result->Property( "swishdocpath" ),
            $result->Property( "swishrank" ),
            $result->Property( "swishdocsize" ),
            $result->Property( "swishtitle" ),
            $result->Property( "swishdbfile" ),
            $result->ResultPropertyStr( "swishlastmodified" ),
            $result->Property( "swishreccount" ),
            $result->Property( "swishfilenum" )
        );
    }
```

```
    # display properties and metanames
```

```
    for my $index_name ( $swish->IndexNames ) {
        my @metas = $swish->MetaList( $index_name );
        my @props = $swish->PropertyList( $index_name );
```

```
        for my $m ( @metas ) {
            my $name = $m->Name;
            my $id = $m->ID;
            my $type = $m->Type;
        }
        # (repeat above for @props)
    }
```

11.2 DESCRIPTION

This module provides a Perl interface to the Swish-e search engine. This module allows embedding the swish-e search code into your application avoiding the need to fork to run the swish-e binary and to keep an index file open when running multiple queries. This results in increased search performance.

11.3 DEPENDENCIES

You must have installed Swish-e version 2.4 before building this module. Download from:

```
    http://swish-e.org
```

11.4 OVERVIEW

This module includes a number of classes.

Searching consists of connecting to a swish-e index (or indexes), and then running queries against the open index. Connecting to the index creates a swish object blessed into the SWISH::API class.

A SWISH::API::Search object is created from the SWISH::API object. The SWISH::API::Search object can have associated parameters (e.g. result sort order).

The SWISH::API::Search object is used to query the associated index file or files. A query on a search object returns a results object of the class SWISH::API::Results. Then individual results of the SWISH::API::Result class can be fetched by calling a method of the results object.

Finally, a result's properties can be accessed by calling methods on the result object.

11.5 METHODS

11.5.1 SWISH::API - Swish Handle Object

To begin using Swish you must first create a Swish Handle object. This object makes the connection to one or more index files and is used to create objects used for searching the associated index files.

$swish = SWISH::API->new($index_files);

This method returns a swish handle object blessed into the SWISH::API class. `$index_files` is a space separated list of index files to open. This always returns an object, even on errors. Caller must check for errors (see below).

@indexes = $swish->IndexNames;

Returns a list of index names associated with the swish handle. These were the indexes specified as a parameter on the SWISH::API->new call. This can be used in calls below that require specifying the index file name.

@header_names = $swish->HeaderNames;

Returns a list of possible header names. These can be used to lookup header values. See `Swish-HeaderValue` method below.

@values = $swish->HeaderValue($index_file, $header_name);

A swish-e index has data associated with it stored in the index header. This method provides access to that data.

Returns the header value for the header and index file specified. Most headers are a single item, but some headers (e.g. "Stopwords") return a list.

The list of possible header names can be obtained from the SwishHeaderNames method.

$swish->RankScheme(0|1);

Similar to the -R option with the swish-e command line tool. The default ranking scheme is 0. Set it to 1 to experiment with other ranking features. See the SWISH-CONFIG documentation for more on ranking schemes.

11.5.1.1 Error Handling

All errors are stored in and accessed via the SWISH::API object (the Swish Handle). That is, even an error that occurs when calling a method on a result (SWISH::API::Result) object will store the error in the parent SWISH:API object.

Check for errors after every method call. Some errors are critical errors and will require destruction of the SWISH::API object. Critical errors will typically only happen when attaching to the database and are errors such as an invalid index file name, permissions errors, or passing invalid objects to calls.

Typically, if you receive an error when attaching to an index file or files you should assume that the error is critical and let the swish object fall out of scope (and destroyed). Otherwise, if an error is detected you should check if it is a critical error. If the error is not critical you may continue using the objects that have been created (for example, an invalid meta name will generate a non-critical error, so you may continue searching using the same search object).

Error state is cleared upon a new query.

Again, all error methods need to be called on the parent swish object

$swish->Error

Returns true if an error occurred on the last operation. On errors the value returned is the internal Swish-e error number (which is less than zero).

$swish->CriticalError

Returns true if the last error was a critical error

$swish->AbortLastError

Aborts the running program and prints an error message to STDERR.

$str = $swish->ErrorString

Returns the string description of the current error (based on the value returned by $swish->Error). This is a generic error string.

$msg = $swish->LastErrorMsg

Returns a string with specific information about the last error, if any. For example, if a query of:

```
        badmeta=foo
```

and "badmeta" is an invalid metaname $swish->ErrorString might return "Unknown metaname", but $swish->LastErrorMsg might return "badmeta".

11.5.1.2 Generating Search and Result Objects

$search = $swish->New_Search_Object($query);

This creates a new search object blessed into the SWISH::API::Search class. The optional `$query` parameter is a query string to store in the search object.

See the section on `SWISH::API::Search` for methods available on the returned object.

The advantage of this method is that a search object can be used for multiple queries:

```
    $search = $swish->New_Search_Objet;
    while ( $query = next_query() ) {
        $results = $search->Execute( $query );
        ...
    }
```

$results = $swish->Query($query);

This is a short-cut which avoids the step of creating a separate search object. It returns a results object blessed into the SWISH::API::Results class described below.

This method basically is the equivalent of

```
    $results = $swish->New_Search_Object->Execute( $query );
```

11.5.2 SWISH::API::Search - Search Objects

A search object holds the parameters used to generate a list of results. These methods are used to adjust these parameters and to create the list of results for the current set of search parameters.

$search->SetQuery($query);

This will set (or replace) the query string associated with a search object. This method is typically not used as the query can be set when executing the actual query or when creating a search object.

$search->SetStructure($structure_bits);

This method may change in the future.

A "structure" is a bit-mapped flag used to limit search results to specific parts of an HTML document, such as the title or in H tags. The possible bits are:

```
    IN_FILE         = 1      This is the default
    IN_TITLE        = 2      In <title> tag
    IN_HEAD         = 4      In <head> tag
    IN_BODY         = 8      In <body>
    IN_COMMENTS     = 16     In html comments
    IN_HEADER       = 32     In <h*>
    IN_EMPHASIZED   = 64     In <em>, <b>, <strong>, <i>
    IN_META         = 128    In a meta tag (e.g. not swishdefault)
```

So if you wish to limit your searches to words in heading tags (e.g. <H1>) or in the <title> tag use:

```
    $search->SetStructure( IN_HEAD | IN_TITLE );
```

$search->PhraseDelimiter($char);

Sets the character used as the phrase delimiter in searches. The default is double-quotes (").

$search->SetSearchLimit($property, $low, $high);

Sets a range from `$low` to `$high` inclusive that the give `$property` must be in to be selected as a result. Call multiple times to set more than one limit on different properties. Limits are ANDed, that is, a result must be within the range of all limits specified to be included in a list of results.

For example to limit searches to documents modified in the last 48 hours:

```
    my $start = time - 48 * 60 * 60;
    $search->SetSearchLimit( ’swishlastmodified’, $start, time() );
```

An error will be set if the property has already been specified or if `$high` < $low.

Other errors may not be reported until running the query, such as the property name is invalid or if `$low` or `$high` are not numeric and the property specified is a numeric property.

Once a query is run you cannot change the limit settings without calling the ResetSearchLimit method first.

$search->ResetSearchLimit;

Clears the limit parameters for the given object. This must be called if the limit parameters need to be changed.

$search->SetSort($sort_string);

Sets the sort order of search results. The string is a space separated list of valid document properties. Each property may contain a qualifier that sets the direction of the sort.

For example, to sort the results by path name in ascending order and by rank in descending order:

```
    $search->SetSort( 'swishdocpath asc swishrank desc' );
```

The "asc" and "desc" qualifiers are optional, and if omitted ascending is assumed.

Currently, errors (e.g invalid property name) are not detected on this call, but rather when executing a query. This may change in the future.

11.5.3 SWISH::API::Results - Generating and accessing results

Searching generates a results object blessed into the SWISH::API::Results class.

$results = $search->Execute($query);

Executes a query based on the parameters in the search object. `$query` is an optional query string to use for the search ($query replaces the set query string in the search object).

A typical use would be to create a search object once and then call this method for each query using the same search object changing only the passed in $query.

The caller should check for errors after making this all.

11.5.4 Results Methods

A query creates a results object that contains information about the query (e.g. number of hits) and access to the individual results.

$hits = $results->Hits;

Returns the number of results for the query. If zero and no errors were reported after calling $search->Execute then the query returned zero results.

@parsed_words = $results->ParsedWords($index_name);

Returns an array of tokenized words and operators with stopwords removed. This is the array of tokens used by swish for the query.

`$index_name` must match one of the index files specified on the creation of the swish object (via the SWISH::API->new call).

The parsed words are useful for highlighting search terms in associated documents.

@removed_stopwords = $results->RemovedStopwords($index_name) ;

Returns an array of stopwords removed from a query, if any, for the index specified.

`$index_name` must match one of the index files specified on the creation of the swish object (via the SWISH::API->new call).

$results->SeekResult($position);

Seeks to the position specified in the result list. Zero is the first position and $results->Hits-1 is the last position. Seeking past the end of results sets a non-critical error condition.

Useful for seeking to a specific "page" of results.

$result = $results->NextResult;

Fetches the next result from the list of results. Returns undef if no more results are available. `$result` is an object blessed into the SWISH::API::Result class.

11.5.5 SWISH::API::Result - Result Methods

The follow methods provide access to data related to an individual result.

$prop = $result->Property($prop_name);

Fetches the property specified for the current result. An invalid property name will cause an exception (which can be caught by wrapping the call in an eval block).

Can return undefined.

Date properties are returned as a timestamp. Use something like Date::Format to format the strings (or just call scalar `localtime( $prop )`).

$prop = $result->ResultPropertyStr($prop_name);

Fetches and formats the property. Unlike above, invalid property names return the string "(null)" -- this will likely change to match the above (i.e. throw an exception).

Undefined values are returned at the null string ("").

$value = $result->ResultIndexValue($header_name);

Returns the header value specified. This is similar to $swish->HeaderValue(), but the index file is not specified (it is determined by the result).

11.5.6 Utility Methods

@metas = $swish->MetaList($index_name);

Swish-e has "MetaNames" which allow searching by fields in the index. This method returns information about the Metanames.

Pass in the name of an open index file name and returns a list of SWISH::API::MetaName objects. Three methods are currently defined on these objects:

```
    $meta->Name;
    $meta->ID;
    $meta->Type;
```

Name returns the name of the meta as defined in the MetaNames config option when the index was created.

The ID is the internal ID number used to represent the meta name.

Type is the type of metaname. Currently only one type exists and its value is zero.

@props = $swish->PropertyList($index_name);

Swish-e can store content or "properties" in the index and return this data when running a query. A document's path, URL, title, size, date or summary are examples of properites. Each property is accessed via its PropertyName. This method returns information about the PropertNames stored in the index.

Pass in the name of an open index file name and returns a list of SWISH::API::MetaName objects. Three methods are currently defined on these objects:

```
    $prop->Name;
    $prop->ID;
    $prop->Type;
```

Name returns the name of the meta as defined in the MetaNames config option when the index was created.

The ID is the internal ID number used to represent the meta name.

Type is the type of metaname. Currently only one type exists and its value is zero.

@propes = $result->PropertyList;

@meta = $result->MetaList;

These also return a list of Property or Metaname description objects, but are accessed via a result record. Since the result comes from a specific index file there's no need to specify the index file name.

$stemmed_word = $swish->StemWord($word);

Deprecated

Returns the stemmed version of the passed in word.

Deprecated because only stems using the original Porter Stemmer and uses a shared memory location in the SW_HANDLE object to store the stemmed word. See below for other stemming options.

$fuzzy_word = $swish->Fuzzy($indexname, $word);

Like StemWord used to work, only it uses whatever stemmer is named in $indexname. Returns the same kind of fuzzy_word object as the `FuzzyWord()` method.

$mode_string = $result->FuzzyMode;

Returns the string (e.g. "Stemming_en", "Soundex", "None") indicating the stemming method used while indexing the given document.

$fuzzy_word = $result->FuzzyWord($word);

Converts `$word` using the same fuzzy mode used to index the $result. Returns a SWISH::API::FuzzyWord object. Methods on the object are used to access the converted words and other data as shown below.

$count = $fuzzy_word->WordCount;

Returns the number of output words. Normally this is the value one, but may be more depending on the stemmer used. DoubleMetaphone can return two strings for a single input string.

$status = $fuzzy_word->WordError;

Returns any error code that the stemmer might set. Normally, this return value is zero, indicating that the stemming/fuzzy operation succedded. The values returned are defined in the swish-e source file /src/stemmer.h.

@words = $fuzzy_word->WordList;

Returns the converted words from the stemming/fuzzy operation. Normally, the array will contain a single element, although may contain more (i.e. if DoubleMetaphone is used and the input word returns two strings).

In the event that a word does not stem (e.g. trying to stem a number), this method will return the original input word specified when $result->FuzzyWord(`$word`) was called.

@parsed_words = $swish->SwishWords($string, $index_file);

* Not implemented *

Splits up the input string into tokens of swish words and operators.

11.6 NOTES

Perl's garbage collection makes it easy to write code for searching with Swish-e, but care must be taken not to keep objects around too long which can use up memory.

Here's an example of a potential problem. Say you have a very large number of documents indexed and you want to find the first hit for a number of popular keywords (error checking omitted in this bad example):

```
 sub first_hit {
   my $query = shift;
   my $handle = SWISH::API->new( 'index.swish-e');
   my $results = $handle->Query( $query );
   my $first_hit = $results->NextResult;
   return $first_hit;
 }
```

```
 my @first_hit_list;
 for ( @keywords )
     push @first_hit_list, $first_hit($_);
 }
```

The `first_hit()` subroutine is returning a SWISH::Result object. That makes it easy to access properties:

```
# print file names
for my $result ( @first_hit_list ) {
   print $result->Property('swishdocpath'),"\n";
}
```

But as long as a SWISH::API::Result object is around, so is the entire list of results generated by the $handle->Query() call, and the index file is still open (because a SWISH::API::Result depends on a SWISH::API::Results object, which depends on a SWISH::API object).

In this case it would be better to return from `first_hit()` just the properties you need:

```
   ...
   my $first_hit = $results->NextResult;
   return $first_hit->Property('swishdocpath');
}
```

Then when `first_hit()` sub ends the result list will be freed, and the index file closed, thanks to Perl's reference count tracking.

Note: the other problem with the above code is that the same index file is opened for each call to the function. Don't do that, instead open the index file once.

11.7 COPYRIGHT

11.8 AUTHOR

Bill Moseley moseley@hank.org. 2002/2003/2004

11.9 SUPPORT

Please contact the Swish-e discussion email list for support with this module or with Swish-e. Please do not contact the developers directly.

12 swish.cgi -- Example Perl script for searching with the SWISH-E search engine.

12.1 DESCRIPTION

`swish.cgi` is a CGI script for searching with the SWISH-E search engine version 2.1-dev and above. It returns results a page at a time, with matching words from the source document highlighted, showing a few words of content on either side of the highlighted word.

The script is highly configurable. Features include searching multiple (or selectable) indexes, limiting searches to a subset of documents, sorting by a number of different properties, and limiting results to a date range.

On unix type systems the swish.cgi script is installed in the directory $prefix/lib/swish-e, which is typically /usr/local/lib/swish-e. This can be overridden by the configure options --prefix or --libexecdir.

The standard configuration (i.e. not using a config file) should work with most swish index files. Customization of the parameters will be needed if you are indexing special meta data and want to search and/or display the meta data. The configuration can be modified by editing this script directly, or by using a configuration file (.swishcgi.conf by default). The script's configuration file is described below.

You are strongly encouraged to get the default configuration working before making changes. Most problems using this script are the result of configuration modifications.

The script is modular in design. Both the highlighting code and output generation is handled by modules, which are included in the *example/modules* distribution directory and installed in the $libexecdir/perl directory. This allows for easy customization of the output without changing the main CGI script.

Included with the Swish-e distribution is a module to generate standard HTML output. There's also modules and template examples to use with the popular Perl templating systems HTML::Template and Template-Toolkit. This is very useful if your site already uses one of these templating systems The HTML::Template and Template-Toolkit packages are not distributed with Swish-e. They are available from the CPAN (http://search.cpan.org).

This scipt can also run basically unmodified as a mod_perl handler, providing much better performance than running as a CGI script. Usage under mod_perl is described below.

Please read the rest of the documentation. There's a `DEBUGGING` section, and a `FAQ` section.

This script should work on Windows, but security may be an issue.

12.2 REQUIREMENTS

A reasonably current version of Perl. 5.00503 or above is recommended (anything older will not be supported).

The Date::Calc module is required to use the date range feature of the script. The Date::Calc module is also available from CPAN.

12.3 INSTALLATION

Here’s an example installation session under Linux. It should be similar for other operating systems.

For the sake of simplicity in this installation example all files are placed in web server space, including files such as swish-e index and configuration files that would normally not be made available via the web server. Access to these files should be limited once the script is running. Either move the files to other locations (and adjust the script’s configuration) or use features of the web server to limit access (such as with *.htaccess*).

Please get a simple installation working before modifying the configuration file. Most problems reported for using this script have been due to improper configuration.

The script’s default settings are setup for initial testing. By default the settings expect to find most files and the swish-e binary in the same directory as the script.

For *security* reasons, once you have tested the script you will want to change settings to limit access to some of these files by the web server (either by moving them out of web space, or using access control such as *.htaccess*). An example of using *.htaccess* on Apache is given below.

It’s expected that swish-e has already been unpacked and the swish-e binary has be compiled from source and "make install" has been run. If swish-e was installed from a vendor package (such as from a RPM or Debian package) see that pakage’s documentation for where files are installed.

Example Installation:

1. **Symlink or copy the swish.cgi.**

   Symlink (or copy if your platform or webserver does not allow symlinks) the swish.cgi script from the installation directory to a local directory. Typically, this would be the cgi-bin directory or a location where CGI script are located. In this example a new directory is created and the script is symlinked.

   ```
       ~$ mkdir swishdir
       ~$ cd swishdir
       ~/swishdir$ ln -s /usr/local/lib/swish-e/swish.cgi
   ```

   The installation directory is set at configure time with the --prefix or --libexecdir options, but by default is in /usr/local/lib/swish-e.

2. **Create an index**

   Use an editor and create a simple configuration file for indexing your files. In this example the Apache documentation is indexed. Last we run a simple query to test that the index works correctly.

```
~/swishdir$ cat swish.conf
IndexDir /usr/local/apache/htdocs
IndexOnly .html .htm
DefaultContents HTML*
StoreDescription HTML* <body> 200000
MetaNames swishdocpath swishtitle
ReplaceRules remove /usr/local/apache/
```

If you do not have the Apache docs installed then pick another directory to index such as /usr/share/doc.

Create the index.

```
~/swishdir$ swish-e -c swish.conf
Indexing Data Source: "File-System"
Indexing "/usr/local/apache/htdocs"
Removing very common words...
no words removed.
Writing main index...
Sorting words ...
Sorting 7005 words alphabetically
Writing header ...
Writing index entries ...
  Writing word text: Complete
  Writing word hash: Complete
  Writing word data: Complete
7005 unique words indexed.
5 properties sorted.
124 files indexed.  1485844 total bytes.  171704 total words.
Elapsed time: 00:00:02 CPU time: 00:00:02
Indexing done!
```

Now, verify that the index can be searched:

```
~/swishdir$ swish-e -w install -m 1
# SWISH format: 2.1-dev-25
# Search words: install
# Number of hits: 14
# Search time: 0.001 seconds
# Run time: 0.040 seconds
1000 htdocs/manual/dso.html "Apache 1.3 Dynamic Shared Object (DSO) support" 17341
.
```

Let’s see what files we have in our directory now:

```
~/swishdir$ ls -1
index.swish-e
index.swish-e.prop
swish.cgi
swish.conf
```

3. **Test the CGI script**

   This is a simple step, but often overlooked. You should test from the command line instead of jumping ahead and testing with the web server. See the `DEBUGGING` section below for more information.

   ```
   ~/swishdir$ ./swish.cgi | head
   Content-Type: text/html; charset=ISO-8859-1
   ```

   ```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
   <html>
       <head>
          <title>
             Search our site
          </title>
       </head>
       <body>
   ```

   The above shows that the script can be run directly, and generates a correct HTTP header and HTML.

   If you run the above and see something like this:

   ```
   ~/swishdir >./swish.cgi
   bash: ./swish.cgi: No such file or directory
   ```

   then you probably need to edit the script to point to the correct location of your perl program. Here's one way to find out where perl is located (again, on unix):

   ```
   ~/swishdir$ which perl
   /usr/local/bin/perl
   ```

   ```
   ~/swishdir$ /usr/local/bin/perl -v
   This is perl, v5.6.0 built for i586-linux
   ...
   ```

   Good! We are using a reasonably current version of perl.

   Now that we know perl is at */usr/local/bin/perl* we can adjust the "shebang" line in the perl script (e.g. the first line of the script):

   ```
   ~/swishdir$ pico swish.cgi
   (edit the #! line)
   ~/swishdir$ head -1 swish.cgi
   #!/usr/local/bin/perl -w
   ```

4. **Test with the web server**

   How you do this is completely dependent on your web server, and you may need to talk to your web server admin to get this working. Often files with the .cgi extension are automatically set up to run as CGI scripts, but not always. In other words, this step is really up to you to figure out!

   This example shows creating a *symlink* from the web server space to the directory used above. This will only work if the web server is configured to follow symbolic links (the default for Apache).

   This operation requires root access:

   ```
   ~/swishdir$ su -c "ln -s $HOME/swishdir /usr/local/apache/htdocs/swishdir"
   Password: *********
   ```

   If your account is on an ISP and your web directory is *~/public_html* the you might just move the entire directory:

   ```
   mv ~/swishdir ~/public_html
   ```

   Now, let’s make a real HTTP request:

   ```
   ~/swishdir$ GET http://localhost/swishdir/swish.cgi | head -3
   #!/usr/local/bin/perl -w
   package SwishSearch;
   use strict;
   ```

   Oh, darn. It looks like Apache is not running the script and instead returning it as a static page. Apache needs to be told that swish.cgi is a CGI script.

   .htaccess comes to the rescue:

   ```
   ~/swishdir$ cat .htaccess
   ```

   ```
   # Deny everything by default
   Deny From All
   ```

   ```
   # But allow just CGI script
   <files swish.cgi>
       Options ExecCGI
       Allow From All
       SetHandler cgi-script
   </files>
   ```

   That "Deny From All" prevents access to all files (such as config and index files), and only access is allowed to the *swish.cgi* script.

Let's try the request one more time:

```
~/swishdir >GET http://localhost/swishdir/swish.cgi | head
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
       <title>
          Search our site
       </title>
    </head>
    <body>
        <h2>
        <a href="http://swish-e.org">
```

That looks better! Now use your web browser to test.

Now, you may note that the links are not valid on the search results page. The swish config file contained the line:

```
 ReplaceRules remove /usr/local/apache/
```

To make those links works (and assuming your web server will follow symbolic links):

```
~/swishtest$ ln -s /usr/local/apache/htdocs
```

BTW - "GET" used above is a program included with Perl's LWP library. If you do no have this you might try something like:

```
wget -O - http://localhost/swishdir/swish.cgi | head
```

and if nothing else, you can always telnet to the web server and make a basic request.

```
~/swishtest$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /swishtest/swish.cgi http/1.0
```

```
HTTP/1.1 200 OK
Date: Wed, 13 Feb 2002 20:14:31 GMT
Server: Apache/1.3.20 (Unix) mod_perl/1.25_01
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
       <title>
          Search our site
       </title>
    </head>
    <body>
```

This may seem like a lot of work compared to using a browser, but browsers are a poor tool for basic CGI debugging.

If you have problems check the `DEBUGGING` section below.

12.4 CONFIGURATION

If you want to change the location of the swish-e binary or the index file, use multiple indexes, add additional metanames and properties, change the default highlighting behavior, etc., you will need to adjust the script's configuration settings.

Again, please get a test setup working with the default parameters before making changes to any configuration settings. Better to debug one thing at a time...

In general, you will need to adjust the script's settings to match the index file you are searching. For example, if you are indexing a hypermail list archive you may want to make the script use metanames/properties of Subject, Author, and, Email address. Or you may wish to provide a way to limit searches to subsets of documents (e.g. parts of your directory tree).

To make things somewhat "simple", the configuration parameters are included near the top of the swish.cgi program. That is the only place that the individual parameters are defined and explained, so you will need to open up the swish.cgi script in an editor to view the options. Further questions about individual settings should be referred to the swish-e discussion list.

The parameters are all part of a perl `hash` structure, and the comments at the top of the program should get you going. The perl hash structure may seem a bit confusing, but it makes it easy to create nested and complex parameters. Syntax is important, so cut-n-paste should be your best defense if you are not a perl programmer.

By the way, Perl has a number of quote operators. For example, to quote a string you might write:

```
title => 'Search My Site',
```

Some options take more than one parameter, where each parameter must be quoted. For example:

```
metanames => [ 'swishdefault', 'swishtitle',  'swishdocpath' ],
```

which assigns an array ([...]) of three strings to the "metanames" variable. Lists of quoted strings are so common in perl that there’s a special operator called "qw" (quote word) to save typing all those quotes:

```
metanames => [ qw/ swishdefault swishtitle swishdocpath / ],
```

or to use the parenthesis as the quote character (you can pick any):

```
metanames => [ qw( swishdefault swishtitle swishdocpath ) ],
```

There are two options for changing the configuration settings from their default values: One way is to edit the script directly, or the other was is to use a separate configuration file. In either case, the configuration settings are a basic perl hash reference.

Using a configuration file is described below, but contains the same hash structure.

There are many configuration settings, and some of them are commented out either by using a "#" symbol, or by simply renaming the configuration directive (e.g. by adding an "x" to the parameter name).

A very basic configuration setup might look like:

```
return {
    title           => ’Search the Swish-e list’,   # Title of your choice.
    swish_binary    => ’swish-e’,                   # Location of swish-e binary
    swish_index     => ’index.swish-e’,             # Location of your index file
};
```

Or if searching more than one index:

```
return {
    title           => ’Search the Swish-e list’,
    swish_binary    => ’swish-e’,
    swish_index     => [’index.swish-e’, ’index2’],
};
```

Both of these examples return a reference to a perl hash (`return {...}`). In the second example, the multiple index files are set as an array reference.

Note that in the example above the swish-e binary file is relative to the current directory. If running under mod_perl you will need to use absolute paths.

The script can also use the SWISH::API perl module (included with the swish-e distribution in the *perl* directory) to access the swish-e index. The `use_library` option is used to enable the use of the SWISH::API module:

```
    return {
        title           => 'Search the Swish-e list',
        swish_index     => ['index.swish-e', 'index2'],
        use_library     => 1, # enable use of the SWISH::API module
    };
```

The module must be available via the `@INC` array, like all Perl modules.

Using the SWISH::API module avoids the need to fork and execute a the swish-e program. Under mod_perl you will may see a significant performance improvement when using the SWISH::API module. Under normal CGI usage you will probably not see any speed improvements.

Using A Configuration File

As mentioned above, configuration settings can be either set in the *swish.cgi* script, or set in a separate configuration file. Settings in a configuration file will override the settings in the script.

By default, the *swish.cgi* script will attempt to read settings from the file *.swishcgi.conf*. For example, you might only wish to change the title used in the script. Simply create a file called *.swishcgi.conf* in the same directory as the CGI script:

```
    > cat .swishcgi.conf
    # Example swish.cgi configuration script.
    return {
       title => 'Search Our Mailing List Archive',
    };
```

The settings you use will depend on the index you create with swish:

```
    return {
         title           => 'Search the Apache documentation',
         swish_binary    => 'swish-e',
         swish_index     => 'index.swish-e',
         metanames       => [qw/swishdefault swishdocpath swishtitle/],
         display_props   => [qw/swishtitle swishlastmodified swishdocsize swishdocpath/],
         title_property  => 'swishdocpath',
         prepend_path    => 'http://myhost/apachedocs',
```

```
        name_labels => {
            swishdefault        => 'Search All',
            swishtitle          => 'Title',
            swishrank           => 'Rank',
            swishlastmodified   => 'Last Modified Date',
            swishdocpath        => 'Document Path',
            swishdocsize        => 'Document Size',
        },
```

```
    };
```

The above configuration defines metanames to use on the form. Searches can be limited to these metanames.

"display_props" tells the script to display the property "swishlastmodified" (the last modified date of the file), the document size, and path with the search results.

The parameter "name_labels" is a hash (reference) that is used to give friendly names to the metanames.

Here’s another example. Say you want to search either (or both) the Apache 1.3 documentation and the Apache 2.0 documentation indexed seperately.

```
     return {
        title       => ’Search the Apache Documentation’,
        date_ranges => 0,
        swish_index => [ qw/ index.apache index.apache2 / ],
        select_indexes  => {
             method  => ’checkbox_group’,
             labels  => [ ’1.3.23 docs’, ’2.0 docs’ ],  # Must match up one-to-one to swish_index
             description => ’Select: ’,
         },
```

```
   };
```

Now you can select either or both sets of documentation while searching.

All the possible settings are included in the default configuration located near the top of the *swish.cgi* script. Open the *swish.cgi* script with an editor to look at the various settings. Contact the Swish-e Discussion list for help in configuring the script.

12.5 DEBUGGING

Most problems with using this script have been a result of improper configuration. Please get the script working with default settings before adjusting the configuration settings.

The key to debugging CGI scripts is to run them from the command line, not with a browser.

First, make sure the program compiles correctly:

```
$ perl -c swish.cgi
swish.cgi syntax OK
```

Next, simply try running the program:

```
$ ./swish.cgi | head
Content-Type: text/html; charset=ISO-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
       <title>
          Search our site
       </title>
    </head>
    <body>
```

Under Windows you will need to run the script as:

```
C:\wwwroot\swishtest> perl swish.cgi
```

Now, you know that the program compiles and will run from the command line. Next, try accessing the script from a web browser.

If you see the contents of the CGI script instead of its output then your web server is not configured to run the script. With Apache look at settings like ScriptAlias, SetHandler, and Options.

If an error is reported (such as Internal Server Error or Forbidden) you need to locate your web server's error_log file and carefully read what the problem is. Contact your web administrator for help locating the web server's error log.

If you don't have access to the web server's error_log file, you can modify the script to report errors to the browser screen. Open the script and search for "CGI::Carp". (Author's suggestion is to debug from the command line -- adding the browser and web server into the equation only complicates debugging.)

The script does offer some basic debugging options that allow debugging from the command line. The debugging options are enabled by setting an environment variable "SWISH_DEBUG". How that is set depends on your operating system and the shell you are using. These examples are using the "bash" shell syntax.

Note: You can also use the "debug_options" configuration setting, but the recommended method is to set the environment variable.

You can list the available debugging options like this:

```
$ SWISH_DEBUG=help ./swish.cgi >outfile
Unknown debug option 'help'.  Must be one of:
       basic: Basic debugging
     command: Show command used to run swish
     headers: Show headers returned from swish
      output: Show output from swish
     summary: Show summary of results
        dump: Show all data available to templates
```

Debugging options may be combined:

```
$ SWISH_DEBUG=command,headers,summary ./swish.cgi >outfile
```

You will be asked for an input query and the max number of results to return. You can use the defaults in most cases. It's a good idea to redirect output to a file. Any error messages are sent to stderr, so those will still be displayed (unless you redirect stderr, too).

Here are some examples:

```
~/swishtest$ SWISH_DEBUG=basic ./swish.cgi >outfile
Debug level set to: 1
Enter a query [all]:
Using 'not asdfghjklzxcv' to match all records
Enter max results to display [1]:
```

```
------ Can't use DateRanges feature ------------
```

```
Script will run, but you can't use the date range feature
Can't locate Date/Calc.pm in @INC (@INC contains: modules /usr/local/lib/perl5/5.6.0/i586-linux /usr/local/lib/perl5/5.6.0 /usr/local/lib/perl5/site_perl/5.6.0/i586-linux /usr/local/lib/perl5/site_perl/5.6.0 /usr/local/lib/perl5/site_perl/5.005/i586-linux /usr/local/lib/perl5/site_perl/5.005 /usr/local/lib/perl5/site_perl .) at modules/DateRanges.pm line 107, <STDIN> line 2.
BEGIN failed--compilation aborted at modules/DateRanges.pm line 107, <STDIN> line 2.
Compilation failed in require at ./swish.cgi line 971, <STDIN> line 2.
```

```
--------------
Can't exec "./swish-e": No such file or directory at ./swish.cgi line 1245, <STDIN> line 2.
Child process Failed to exec './swish-e' Error: No such file or directory at ./swish.cgi line 1246, <STDIN> line 2.
Failed to find any results
```

The above indicates two problems. First problem is that the Date::Calc module is not installed. The Date::Calc module is needed to use the date limiting feature of the script.

The second problem is a bit more serious. It's saying that the script can't find the swish-e binary file. In this example it's specified as being in the current directory. Either correct the path to the swish-e binary, or make a local copy or symlink to the swish-e binary.

```
~/swishtest$ cat .swishcgi.conf
    return {
       title       => 'Search the Apache Documentation',
       swish_binary => '/usr/local/bin/swish-e',
       date_ranges => 0,
    };
```

Now, let's try again:

```
~/swishtest$ SWISH_DEBUG=basic ./swish.cgi >outfile
Debug level set to: 1
```

```
---------- Read config parameters from ’.swishcgi.conf’ ------
$VAR1 = {
          ’date_ranges’ => 0,
          ’title’ => ’Search the Apache Documentation’
        };
-------------------------
Enter a query [all]:
Using ’not asdfghjklzxcv’ to match all records
Enter max results to display [1]:
Found 1 results
```

```
Can’t locate SWISH::TemplateDefault.pm in @INC (@INC contains: modules /usr/local/lib/perl5/5.6.0/i586-linux /usr/local/lib/perl5/5.6.0 /usr/local/lib/perl5/site_perl/5.6.0/i586-linux /usr/local/lib/perl5/site_perl/5.6.0 /usr/local/lib/perl5/site_perl/5.005/i586-linux /usr/local/lib/perl5/site_perl/5.005 /usr/local/lib/perl5/site_perl .) at ./swish.cgi line 608.
```

This means that the swish.cgi script could not locate a required module. To correct this locate where the SWISH::Template module is installed and add a "use lib" line to your configuration file (or to the swish.cgi script):

```
~/swishtest$ cat .swishcgi.conf
use lib ’/home/bill/local/lib/perl’;
```

```
return {
   title       => ’Search the Apache Documentation’,
   date_ranges => 0,
};
```

```
~/swishtest$ SWISH_DEBUG=basic ./swish.cgi >outfile
Debug level set to: 1
```

```
---------- Read config parameters from ’.swishcgi.conf’ ------
$VAR1 = {
          ’date_ranges’ => 0,
          ’title’ => ’Search the Apache Documentation’
        };
-------------------------
Enter a query [all]:
Using ’not asdfghjklzxcv’ to match all records
Enter max results to display [1]:
Found 1 results
```

That is much better!

The "use lib" statement tells Perl where to look for modules by adding the path supplied to an array called @INC.

Note that most modules are in the SWISH namespace. For example, the default output module is called SWISH::TemplateDefault. When Perl is looking for that module it is looking for the file *SWISH/TemplateDefault.pm*. If the "use lib" statement is set as:

```
use lib '/home/bill/local/lib/perl';
```

then Perl will look (among other places) for the file

```
/home/bill/local/lib/perl/SWISH/TemplateDefault.pm
```

when attempting to load the SWISH::TemplateDefault module. Relative paths may also be used.

```
use lib 'modules';
```

will cause Perl to look for the file:

```
./modules/SWISH/TemplateDefault.pm
```

relative to where the swish.cgi script is running. (This is not true when running under mod_perl).

Here's another common problem. Everything checks out, but when you run the script you see the message:

```
Swish returned unknown output
```

Ok, let's find out what output it is returning:

```
~/swishtest$ SWISH_DEBUG=headers,output ./swish.cgi >outfile
Debug level set to: 13
```

```
---------- Read config parameters from '.swishcgi.conf' ------
$VAR1 = {
          'swish_binary' => '/usr/local/bin/swish-e',
          'date_ranges' => 0,
          'title' => 'Search the Apache Documentation'
        };
-------------------------
Enter a query [all]:
Using 'not asdfghjklzxcv' to match all records
Enter max results to display [1]:
  usage: swish [-i dir file ... ] [-S system] [-c file] [-f file] [-l] [-v (num)]
  ...
version: 2.0
   docs: http://sunsite.berkeley.edu/SWISH-E/
```

```
*** 9872 Failed to run swish: 'Swish returned unknown output' ***
Failed to find any results
```

Oh, looks like /usr/local/bin/swish-e is version 2.0 of swish. We need 2.1-dev and above!

12.6 Frequently Asked Questions

Here's some common questions and answers.

12.6.1 How do I change the way the output looks?

The script uses a module to generate output. By default it uses the SWISH::TemplateDefault.pm module. The module used is selected in the swish.cgi configuration file. Modules are located in the example/modules/SWISH directory in the distribution, but are installed in the $prefix/lib/swish-e/perl/SWISH/ directory.

To make simple changes you can edit the installed SWISH::TemplatDefault module directly, otherwise make a copy of the module and modify its package name. For example, change directories to the location of the installed module and copy the module to a new name:

```
    $ cp TemplateDefault.pm MyTemplateDefault.pm
```

Then at the top of the module adjust the "package" line to:

```
    package SWISH::MyTemplateDefault;
```

To use this modules you need to adjust the configuration settings (either at the top of *swish.cgi* or in a configuration file:

```
        template => {
            package     => 'SWISH::MyTemplateDefault',
        },
```

The module does not need to be in the SWISH namespace, and can be stored in any location as long as the module can be found via the `@INC` array (i.e. modify the "use lib" statement in swish.cgi if needed).

12.6.2 How do I use a templating system with swish.cgi?

In addition to the TemplateDefault.pm module, the swish-e distribution includes two other Perl modules for generating output using the templating systems HTML::Template and Template-Toolkit.

Templating systems use template files to generate the HTML, and make maintaining the look of a large (or small) site much easier. HTML::Template and Template-Toolkit are separate packages and can be downloaded from the CPAN. See http://search.cpan.org.

Two basic templates are provided as examples for generating output using these templating systems. The example templates are located in the *example* directory. The module *SWISH::TemplateHTMLTemplate* uses the file *swish.tmpl* to generate its output, while the module *SWISH::TemplateToolkit* uses the *swish.tt* file. (Note: swish.tt was renamed from search.tt Jun 03, 2004.)

To use either of these modules you will need to adjust the "template" configuration setting. Examples for both templating systems are provided in the configuration settings near the top of the *swish.cgi* program.

Use of these modules is an advanced usage of *swish.cgi* and are provided as examples only.

All of the output generation modules are passed a hash with the results from the search, plus other data use to create the output page. You can see this hash by using the debugging option "dump" or by using the included SWISH::TemplateDumper module:

```
~/swishtest >cat .swishcgi.conf
    return {
       title       => 'Search the Apache Documentation',
       template => {
            package     => 'SWISH::TemplateDumper',
        },
    };
```

And run a query. For example:

```
http://localhost/swishtest/swish.cgi?query=install
```

12.6.3 Why are there three different highlighting modules?

Three are three highlighting modules included with the swish-e distribution. Each is a trade-off of speed vs. accuracy:

```
SWISH::DefaultHighlight - reasonably fast, but does not highlight phrases
SWISH::PhraseHighlight  - reasonably slow, but is reasonably accurate
SWISH::SimpleHighlight  - fast, some phrases, but least accurate
```

Eh, the default is actually "PhraseHighlight". Oh well.

All of the highlighting modules slow down the script. Optimizations to these modules are welcome!

12.6.4 My ISP doesn't provide access to the web server logs

There are a number of options. One way it to use the CGI::Carp module. Search in the swish.cgi script for:

```
use Carp;
# Or use this instead -- PLEASE see perldoc CGI::Carp for details
# use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
```

And change it to look like:

```
#use Carp;
# Or use this instead -- PLEASE see perldoc CGI::Carp for details
use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
```

This should be only for debugging purposes, as if used in production you may end up sending quite ugly and confusing messages to your browsers.

12.6.5 Why does the output show (NULL)?

Swish-e displays (NULL) when attempting to display a property that does not exist in the index.

The most common reason for this message is that you did not use StoreDescription in your config file while indexing.

```
StoreDescription HTML* <body> 200000
```

That tells swish to store the first 200,000 characters of text extracted from the body of each document parsed by the HTML parser. The text is stored as property "swishdescription".

The index must be recreated after changing the swish-e configuration.

Running:

```
~/swishtest > ./swish-e -T index_metanames
```

will display the properties defined in your index file.

This can happen with other properties, too. For example, this will happen when you are asking for a property to display that is not defined in swish.

```
~/swishtest > ./swish-e -w install -m 1 -p foo
# SWISH format: 2.1-dev-25
# Search words: install
err: Unknown Display property name "foo"
.
```

```
~/swishtest > ./swish-e -w install -m 1 -x ’Property foo=<foo>\n’
# SWISH format: 2.1-dev-25
# Search words: install
# Number of hits: 14
# Search time: 0.000 seconds
# Run time: 0.038 seconds
Property foo=(NULL)
.
```

To check that a property exists in your index you can run:

```
~/swishtest > ./swish-e -w not dkdk -T index_metanames | grep foo
        foo : id=10 type=70  META_PROP:STRING(case:ignore) *presorted*
```

Ok, in this case we see that "foo" is really defined as a property. Now let’s make sure *swish.cgi* is asking for "foo" (sorry for the long lines):

```
~/swishtest > SWISH_DEBUG=command ./swish.cgi > /dev/null
Debug level set to: 3
Enter a query [all]:
Using ’not asdfghjklzxcv’ to match all records
Enter max results to display [1]:
---- Running swish with the following command and parameters ----
./swish-e  \
-w  \
’swishdefault=(not asdfghjklzxcv)’  \
-b  \
1  \
-m  \
1  \
-f  \
index.swish-e  \
-s  \
swishrank  \
desc  \
swishlastmodified  \
desc  \
-x  \
’<swishreccount>\t<swishtitle>\t<swishdescription>\t<swishlastmodified>\t<swishdocsize>\t<swishdocpath>\t<fos>\t<swishrank>\t<swishdocpath>\n’  \
-H  \
9
```

If you look carefully you will see that the -x parameter has "fos" instead of "foo", so there’s our problem.

12.6.6 How do I use the SWISH::API perl module with swish.cgi?

Use the `use_library` configuration directive:

```
use_library => 1,
```

This will only provide improved performance when running under mod_perl or other persistent environments.

12.6.7 Why does the "Run time" differ when using the SWISH::API module

When using the SWISH::API module the run (and search) times are calculated within the script. When using the swish-e binary the swish-e program reports the times. The "Run time" may include the time required to load and compile the SWISH::API module.

12.7 MOD_PERL

This script can be run under mod_perl (see http://perl.apache.org). This will improve the response time of the script compared to running under CGI by loading the swish.cgi script into the Apache web server.

You must have a mod_perl enabled Apache server to run this script under mod_perl.

Configuration is simple. In your httpd.conf or your startup.pl file you need to load the script. For example, in httpd.conf you can use a perl section:

```
<perl>
    use lib '/usr/local/apache/cgi-bin';  # location of the swish.cgi file
    use lib '/home/yourname/swish-e/example/modules';  # modules required by swish.cgi
    require "swish.cgi";
</perl>
```

Again, note that the paths used will depend on where you installed the script and the modules. When running under mod_perl the swish.cgi script becomes a perl module, and therefore the script does not need to be installed in the cgi-bin directory. (But, you can actually use the same script as both a CGI script and a mod_perl module at the same time, read from the same location.)

The above loads the script into mod_perl. Then to configure the script to run add this to your httpd.conf configuration file:

```
<location /search>
    PerlSetVar Swish_Conf_File /home/yourname/swish-e/myconfig.pl
    allow from all
    SetHandler perl-script
    PerlHandler SwishSearch
</location>
```

Note that you use the "Swish_Conf_File" setting in httpd.conf to tell the script which config file to use. This means you can use the same script (and loaded modules) for different search sites (running on the same Apache server). You can just specify differnt config files for each Location and they can search different indexes and have a completely different look for each site, but all share the same code.

Note that the config files are cached in the swish.cgi script. Changes to the config file will require restarting the Apache server before they will be reloaded into the swish.cgi script. This avoids calling `stat()` for every request.

Unlike CGI, mod_perl does not change the current directory to the location of the script, so your settings for the swish binary and the path to your index files must be absolute paths (or relative to the server root).

Using the SWISH::API module with mod_perl will provide the most performance improvements. Use of the SWISH::API module can be enabled by the configuration setting `use_library`:

```
use_library     => 1,
```

Without highlighting code enabled, using the SWISH::API module resulted in about 20 requests per second, where running the swish-e binary slowed the script down to about 8 requests per second.

Note that the highlighting code is slow. For the best search performance turn off highlighting. In your config file you can add:

```
    highlighting    => 0,  # disable highlighting
```

and the script will show the first 500 chars of the description (or whatever you set for "max_chars"). Without highlight one test was processing about 20 request per second. With The "PhraseHighlight" module that dropped to a little better than two requests per second, "DefaultHighlight" was about 2.3 request per second, and "SimpleHighlight" was about 6 request per second.

Experiement with different highlighting options when testing performance.

Please post to the swish-e discussion list if you have any questions about running this script under mod_perl.

Here’s some general request/second on an Athlon XP 1800+ with 1/2GB RAM, Linux 2.4.20.

```
                               Highlighting Mode
```

```
                   None     Phrase    Default     Simple
   Using SWISH::API   45        1.5        2          12
   ----------------------------------------------------------------------------
   Using swish-e      12        1.3       1.8         7.5
     binary
```

As you can see the highlighting code is a limiting factor.

12.8 SpeedyCGI

SpeedyCGI (also called PersistentPerl) is another way to run Perl scripts persistently. SpeedyCGI is good if you do not have mod_perl available or do not have root access. SpeedyCGI works on Unix systems by loading the script into a "back end" process and keeping it in memory between requests. New requests are passed to the back end processes which avoids the startup time required by a Perl CGI script.

Install SpeedyCGI from http://daemoninc.com/ (your OS may provide a packaged version of SpeedyCGI) and then change the first line of swish.cgi. For example, if the speedy binary is installed in /usr/bin/speedy, use the line:

```
    #! /usr/bin/speedy -w -- -t60
```

The -w option is passed to Perl, and all options following the double-dash are SpeedyCGI options.

Note that when using SpeedyCGI configuration data is cached in memory. If you change the swish.cgi configuration file (.swishcgi.conf) then touch the main swish.cgi script to force reloading of configuration data.

12.9 Spidering

There are two ways to spider with swish-e. One uses the "http" input method that uses code that's part of swish. The other way is to use the new "prog" method along with a perl helper program called `spider.pl`.

Here's an example of a configuration file for spidering with the "http" input method. You can see that the configuration is not much different than the file system input method. (But, don't use the http input method -- use the -S prog method shown below.)

```
        # Define what to index
        IndexDir http://www.myserver.name/index.html
        IndexOnly .html .htm
```

```
        IndexContents HTML* .html .htm
        DefaultContents HTML*
        StoreDescription HTML* <body> 200000
        MetaNames swishdocpath swishtitle
```

```
        # Define http method specific settings -- see swish-e documentation
        SpiderDirectory ../swish-e/src/
        Delay 0
```

You index with the command:

```
        swish-e -S http -c spider.conf
```

Note that this does take longer. For example, spidering the Apache documentation on a local web server with this method took over a minute, where indexing with the file system took less than two seconds. Using the "prog" method can speed this up.

Here's an example configuration file for using the "prog" input method:

```
        # Define the location of the spider helper program
        IndexDir ../swish-e/prog-bin/spider.pl
```

```
        # Tell the spider what to index.
        SwishProgParameters default http://www.myserver.name/index.html
```

```
        IndexContents HTML* .html .htm
        DefaultContents HTML*
        StoreDescription HTML* <body> 200000
        MetaNames swishdocpath swishtitle
```

Then to index you use the command:

```
swish-e -c prog.conf -S prog -v 0
```

Spidering with this method took nine seconds.

12.10 Stemmed Indexes

Many people enable a feature of swish called word stemming to provide "fuzzy" search options to their users. The stemming code does not actually find the "stem" of word, rather removes and/or replaces common endings on words. Stemming is far from perfect, and many words do not stem as you might expect. Plus, currently only English is supported. But, it can be a helpful tool for searching your site. You may wish to create both a stemmed and non-stemmed index, and provide a checkbox for selecting the index file.

To enable a stemmed index you simply add to your configuration file:

```
UseStemming yes
```

If you want to use a stemmed index with this program and continue to highlight search terms you will need to install a perl module that will stem words. This section explains how to do this.

The perl module is included with the swish-e distribution. It can be found in the examples directory (where you found this file) and called something like:

```
SWISH-Stemmer-0.05.tar.gz
```

The module should also be available on CPAN (http://search.cpan.org/).

Here’s an example session for installing the module. (There will be quite a bit of output when running make.)

```
% gzip -dc SWISH-Stemmer-0.05.tar.gz |tar xof -
% cd SWISH-Stemmer-0.05
% perl Makefile.PL
or
% perl Makefile.PL PREFIX=$HOME/perl_lib
% make
% make test
```

```
(perhaps su root at this point if you did not use a PREFIX)
% make install
% cd ..
```

Use the **PREFIX** if you do not have root access or you want to install the modules in a local library. If you do use a PREFIX setting, add a `use lib` statement to the top of this swish.cgi program.

For example:

```
use lib qw(
    /home/bmoseley/perl_lib/lib/site_perl/5.6.0
    /home/bmoseley/perl_lib/lib/site_perl/5.6.0/i386-linux/
);
```

Once the stemmer module is installed, and you are using a stemmed index, the `swish.cgi` script will automatically detect this and use the stemmer module.

12.11 DISCLAIMER

Please use this CGI script at your own risk.

This script has been tested and used without problem, but you should still be aware that any code running on your server represents a risk. If you have any concerns please carefully review the code.

See http://www.w3.org/Security/Faq/www-security-faq.html

Security on Windows questionable.

12.12 SUPPORT

The SWISH-E discussion list is the place to ask for any help regarding SWISH-E or this example script. See http://swish-e.org.

Before posting please review:

```
http://swish-e.org/2.2/docs/INSTALL.html#When_posting_please_provide_the_
```

Please do not contact the author or any of the swish-e developers directly.

12.13 LICENSE

swish.cgi $Revision: 1.20 $ Copyright (C) 2001 Bill Moseley search@hank.org Example CGI program for searching with SWISH-E

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

12.14 AUTHOR

Bill Moseley

13 search.cgi -- Example Perl program for searching with Swish-e and SWISH::API

13.1 DESCRIPTION

This is a very simple program that shows how to use the SWISH::API module in a CGI script or mod_perl handler using Template-Toolkit to generate output. This program is intended for programmers that want to create a custom search script.

Unlike *swish.cgi* this script does not have many features, and provides no external configuration (with the execption of a few config options under mod_perl). So don't ask why it doesn't do something. The point is that this script is used as a starting point that YOU customize.

13.2 REQUIREMENTS

You must have swish-e and the SWISH::API module installed. See the README and INSTALL documents in the swish-e distribution. As of this writing SWISH::API is part of the swish-e distribution, but in the future may be provided as a separate package (provided on the CPAN). In either case SWISH::API is a separate installation procedure from installing swish-e. The Storable module is also required if using mod_perl.

This program does require that some modules are installed from CPAN. You will need Template-Toolkit and HTML::FillInForm (which depends on HTML::Parser). How those are installed depends on your computer's packaging system.

You will need a web server, obviously. The discussion below assumes Apache is used. If you are using MS IIS take note that IIS works differently in a number of ways.

13.3 OVERVIEW

The *search.cgi* script and related templates are installed when swish-e is installed. *search.cgi* is installed in $prefix/lib/swish-e/ and templates are installed in $prefix/share/swish-e/templates/. `$prefix` is /usr/local by default but can be changed when running the swish-e *configure* script. Upon installation *search.cgi* is updated with correct paths to your perl binary and

When running as a CGI script *search.cgi* is copied or symlinked to the location of your CGI scripts (or any directory that allows CGI scripts). By default, the *search.cgi* script looks for the index *index.swish-e* in the current directory (that's what the web server considers the current directory). On Apache running mod-cgi that's the same place as the script. On IIS it's not. If your index is elsewhere you will need to modify the script.

The script works by parsing the query, calling SWISH::API to run the actual search, then calls Template-Toolkit to generate the ouput.

The script calls the *search.tt* template. This template generates the query form and the search results. The *search.tt* template uses a Template-Toolkit "WRAPPER" function to wrap the search form and results in your site's design. This design is in the *page_layout* template. The idea is if you use Template-Toolkit to manage your entire site then your entire site would be formatted by the same *page_layout* template. The *page_layout* template calls two other templates *common_header* and *common_footer* to generate a common header and footer for the site. Those are just demonstrating Template-Toolkit’s features.

The *page_layout* page only defines the basic structure of the site. The true design of the site is managed by style sheets. *style.css* defines the basic layout and *markup.css* sets fonts and colors.

Note: these style sheets are included directly in the output of the CGI script. In production the style sheets would be stored as separate style sheet files and imported by the browser instead of directly included in the search results page.

See the section MOD_PERL below for more on templates.

Highlighting of search terms is provided by the SWISH::PhraseHighlight module. That is a very slow module, so you may wish to disable it if you expect a lot of traffic.

13.4 INSTALLATION EXAMPLE

Enough talking, sometimes it’s nice to see a complete example. Below swish-e is installed in the default location (/usr/local). The "$" is a normal user prompt, where "#" is a root prompt. Use ./configure --prefix to install in another location (e.g. if you do not have root access).

Download and install swish-e

```
$ wget -q http://swish-e.org/Download/latest.tar.gz
$ tar zxf latest.tar.gz
$ cd swish-e-2.x.x
$ (./configure && make) >/dev/null
$ make check
$ su
# make install
# exit
```

Install SWISH::API

```
$ cd perl
$ perl Makefile.PL && make && make test
$ su
# make install
$ exit
```

Install requried Perl modules. You can install via RPMs, Debs or directly from the CPAN or by using the CPAN shell.

```
# su
# perl -MCPAN -e ’install Template’
# perl -MCPAN -e ’install HTML::FillInForm’
# exit
```

Now setup the script in someplace that allows CGI scripts.

```
$ cd $HOME/apache
$ ln -s /usr/local/lib/swish-e/search.cgi .
$ cat .htaccess
deny from all
<files search.cgi>
    allow from all
    SetHandler cgi-script
    Options +ExecCGI
</files>
```

Create an index

```
$ cat swish.config
IndexOnly .htm .html
DefaultContents HTML*
StoreDescription HTML* <body>
metanames swishtitle swishdocpath
```

```
$ swish-e -c swish.config -i /usr/share/doc/apache-doc/manual
```

Test the index and the CGI script:

```
$ swish-e -w apache -m1 | grep hits
# Number of hits: 152
```

```
$ lynx -dump http://localhost/apache/search.cgi?query=apache | grep hits
    Showing page 1 (1 - 10 of 152 hits) [3]Next
          'hits' => 152,
```

Now, the above isn't very helpful because the Apache documentation indexed is not in the web space. You would likely index content available on your web site.

13.5 Using with SpeedyCGI

Perl CGI script must be compiled for each request. SpeedyCGI is a tool to speed up scripts by running them persistently. To run *search.cgi* with SpeedyCGI install the program (you can Google, right?) and then change the first line of *search.cgi* to run the *speedy* program.

For example:

```
#!/usr/bin/speedy -w
```

13.6 Using with MOD_PERL

This script can be run directly as a mod_perl handler, and the same code can be used to run multiple sites by using separate Location directives and passing in a "site id." The script caches in memory different configurations based on this site id.

Below is a complete httpd.conf file. It requires an Apache httpd that has mod_perl compiled in statically. It runs mod_perl on a high port (port 5000) listening to all interfaces.

For testing I put this config file in a directory along with *search.cgi*, but that’s just done to make the example simple (i.e. so I don’t have to show any absolute paths). Normally the httpd.conf and the swish.cgi "module" would be in separate locations.

```
# httpd.conf -- test file for search.cgi as mod_perl handler

<ifModule mod_so.c>
    LoadModule mime_module /usr/lib/apache/1.3/mod_mime.so
</IfModule>

ErrorLog swish_error_log
PidFile swish_httpd.pid

Listen *:5000

<perl>
    push @PerlSetVar, [
        index  => Apache->server_root_relative( ’index.swish-e’),
    ];
    $DocumentRoot =  Apache->server_root_relative;
    require "search.cgi";
</perl>

NameVirtualHost *:5000
<VirtualHost *:5000>

    ServerName localhost

    <Location /search>
        SetHandler  perl-script
        PerlHandler SwishAPISearch
    </Location>
```

```
        <Location /othersite>
            SetHandler perl-script
            PerlHandler SwishAPISearch
            # Define this site
            PerlSetVar  site_id othersite
            PerlSetVar  title "Some other Site"
        </Location>
```

```
    </VirtualHost>
```

The server is started using this command:

```
    $ /usr/sbin/apache-perl -d $(pwd) -f $(pwd)/httpd.conf
```

which says to use the current directory as the ServerRoot. (See comments below.) Stop the server like:

```
    $ kill ‘cat swish_httpd.pid‘
```

Then access either:

```
    http://localhost:5000/search
    http://localhost:5000/othersite
```

A few Notes:

I like test configurations to not care where things are located. Thus, the above httpd.conf does a few tricks in the "Perl Section" shown.

First, mod_perl, unlike CGI, doesn’t set the working directory. So, the index file name must be absolute. This is accomplished by a PerlSetVar entry building the index file name from the ServerRoot.

Second, the DocumentRoot is set to the same as the ServerRoot. The DocumentRoot needs to be set so search.cgi can figure out the path to the script (for creating next and previous links).

Third, the script is loaded by a `require` statement. This works only because the current directory "." is in Perl’s `@INC` path at Apache start up time and *search.cgi* is also in the current directory. Normally, set PERL5LIB on server startup or use a "use lib" line in your startup.pl file to point to the location of search.cgi.

The "PerlSetVar" lines pass config information into the script. Note that they can be set globally or specific to a given Location.

The following config options are currently available:

site_id

The site_id options allow caching of configurations on a per-site basis. It’s overkill in this example, but normally you might have expensive configuration processes that you might want to do only once. But, since there is caching by this id it’s a good id to set a site_id if using more than one Location directive.

index

This specifies the index file to use. The index file needs to be absolute as discussed above. Example:

```
    PerlSetVar index /usr/share/swish/site.index
```

title

This options sets the title that’s passed into the template.

template

Sets the file name of the template use to generate the form. This might be useful if you want an "advanced" form, for example.

template_path

This can be used to update the path where templates are searched. Useful if you wish to override templates.

page_size

This allow changing the default number of results shown per page.

13.7 SUPPORT

Not much support is provided. But what support is provided is ONLY provided via the Swish-e discussion list.

```
    http://swish-e.org/
```

13.8 AUTHOR

Bill Moseley

13.9 LICENSE

13.10 SEE ALSO

SWISH::API, Template, HTML::FillInForm

14 spider.pl - Example Perl program to spider web servers

14.1 SYNOPSIS

```
spider.pl [<spider config file>] [<URL> ...]
```

```
# Spider using some common defaults and capture the output
# into a file
```

```
./spider.pl default http://myserver.com/ > output.txt
```

```
# or using a config file
```

```
spider.config:
@servers = (
    {
        base_url    => 'http://myserver.com/',
        email       => 'me@myself.com',
        # other spider settings described below
    },
);
```

```
./spider.pl spider.config > output.txt
```

```
# or using the default config file SwishSpiderConfig.pl
./spider.pl > output.txt
```

```
# using with swish-e
```

```
./spider.pl spider.config | swish-e -c swish.config -S prog -i stdin
```

```
# or in two steps
./spider.pl spider.config > output.txt
swish-e -c swish.config -S prog -i stdin < output.txt
```

```
# or with compression
./spider.pl spider.config | gzip > output.gz
gzip -dc output.gz | swish-e -c swish.config -S prog -i stdin
```

```
# or having swish-e call the spider directly using the
# spider config file SwishSpiderConfig.pl:
swish-e -c swish.config -S prog -i spider.pl
```

```
# or the above but passing passing a parameter to the spider:
echo "SwishProgParameters  spider.config" >> swish.config
echo "IndexDir spider.pl" >> swish.config
swish-e -c swish.config -S prog
```

```
Note: When running on some versions of Windows (e.g. Win ME and Win 98 SE)
you may need to tell Perl to run the spider directly:
```

```
    perl spider.pl | swish-e -S prog -c swish.conf -i stdin
```

```
This pipes the output of the spider directly into swish.
```

14.2 DESCRIPTION

spider.pl is a program for fetching documnts from a web server, and outputs the documents to STDOUT in a special format designed to be read by Swish-e.

The spider can index non-text documents such as PDF and MS Word by use of filter (helper) programs. These programs are not part of the Swish-e distribution and must be installed separately. See the section on filtering below.

A configuration file is noramlly used to control what documents are fetched from the web `server(s).` The configuration file and its options are described below. The is also a "default" config suitable for spidering.

The spider is designed to spider web pages and fetch documents from one host at a time -- offsite links are not followed. But, you can configure the spider to spider multiple sites in a single run.

spider.pl is distributed with Swish-e and is installed in the swish-e library directory at installation time. This directory (libexedir) can be seen by running the command:

```
swish-e -h
```

Typically on unix-type systems the spider is installed at:

```
/usr/local/lib/swish-e/spider.pl
```

This spider stores all links in memory while processing and does not do parallel requests.

14.2.1 Running the spider

The output from *spider.pl* can be captured to a temporary file which is then fed into swish-e:

```
 ./spider.pl > docs.txt
 swish-e -c config -S prog -i stdin < docs.txt
```

or the output can be passed to swish-e via a pipe:

```
./spider.pl | swish-e -c config -S prog -i stdin
```

or the swish-e can run the spider directly:

```
swish-e -c config -S prog -i spider.pl
```

One advantage of having Swish-e run *spider.pl* is that Swish-e knows where to locate the program (based on libexecdir compiled into swish-e).

When running the spider *without* any parameters it looks for a configuration file called *SwishSpiderConfig.pl* in the current directory. The spider will abort with an error if this file is not found.

A configuration file can be specified as the first parameter to the spider:

```
 ./spider.pl spider.config > output.txt
```

If running the spider via Swish-e (i.e. Swish-e runs the spider) then use the Swish-e config option SwishProgParameters to specify the config file:

In swish.config:

```
 # Use spider.pl as the external program:
 IndexDir spider.pl
 # And pass the name of the spider config file to the spider:
 SwishProgParameters spider.config
```

And then run Swish-e like this:

```
 swish-e -c swish.config -S prog
```

Finally, by using the special word "default" on the command line the spider will use a default configuration that is useful for indexing most sites. It’s a good way to get started with the spider:

```
    ./spider.pl default http://my_server.com/index.html > output.txt
```

There’s no "best" way to run the spider. I like to capture to a file and then feed that into Swish-e.

The spider does require Perl’s LWP library and a few other reasonably common modules. Most well maintained systems should have these modules installed. See REQUIREMENTS below for more information. It’s a good idea to check that you are running a current version of these modules.

Note: the "prog" document source in Swish-e bypasses many Swish-e configuration settings. For example, you cannot use the IndexOnly directive with the "prog" document source. This is by design to limit the overhead when using an external program for providing documents to swish; after all, with "prog", if you don’t want to index a file, then don’t give it to swish to index in the first place.

So, for spidering, if you do not wish to index images, for example, you will need to either filter by the URL or by the content-type returned from the web server. See CALLBACK FUNCTIONS below for more information.

14.2.2 Robots Exclusion Rules and being nice

By default, this script will not spider files blocked by *robots.txt*. In addition, The script will check for <meta name="robots"..> tags, which allows finer control over what files are indexed and/or spidered. See http://www.robotstxt.org/wc/exclusion.html for details.

This spider provides an extension to the <meta> tag exclusion, by adding a **NOCONTENTS** attribute. This attribute turns on the `no_contents` setting, which asks swish-e to only index the document’s title (or file name if not title is found).

For example:

```
      <META NAME="ROBOTS" CONTENT="NOCONTENTS, NOFOLLOW">
```

says to just index the document’s title, but don’t index its contents, and don’t follow any links within the document. Granted, it’s unlikely that this feature will ever be used...

If you are indexing your own site, and know what you are doing, you can disable robot exclusion by the ignore_robots_file configuration parameter, described below. This disables both *robots.txt* and the meta tag parsing. You may disable just the meta tag parsing by using `ignore_robots_headers`.

This script only spiders one file at a time, so load on the web server is not that great. And with libwww-perl-5.53_91 HTTP/1.1 keep alive requests can reduce the load on the server even more (and potentially reduce spidering time considerably).

Still, discuss spidering with a site’s administrator before beginning. Use the delay_sec to adjust how fast the spider fetches documents. Consider running a second web server with a limited number of children if you really want to fine tune the resources used by spidering.

14.2.3 Duplicate Documents

The spider program keeps track of URLs visited, so a document is only indexed one time.

The Digest::MD5 module can be used to create a "fingerprint" of every page indexed and this fingerprint is used in a hash to find duplicate pages. For example, MD5 will prevent indexing these as two different documents:

```
    http://localhost/path/to/some/index.html
    http://localhost/path/to/some/
```

But note that this may have side effects you don’t want. If you want this file indexed under this URL:

```
    http://localhost/important.html
```

But the spider happens to find the exact content in this file first:

```
    http://localhost/developement/test/todo/maybeimportant.html
```

Then only that URL will be indexed.

14.2.4 Broken relative links

Sometimes web page authors use too many `/../` segments in relative URLs which reference documents above the document root. Some web servers such as Apache will return a 400 Bad Request when requesting a document above the root. Other web servers such as Micorsoft IIS/5.0 will try and "correct" these errors. This correction will lead to loops when spidering.

The spider can fix these above-root links by placing the following in your spider config:

```
    remove_leading_dots => 1,
```

It is not on by default so that the spider can report the broken links (as 400 errors on sane webservers).

14.2.5 Compression

If The Perl module Compress::Zlib is installed the spider will send the

```
   Accept-Encoding: gzip
```

header and uncompress the document if the server returns the header

```
Content-Encoding: gzip
```

MD5 checksomes are done on the compressed data.

MD5 may slow down indexing a tiny bit, so test with and without if speed is an issue (which it probably isn't since you are spidering in the first place). This feature will also use more memory.

14.3 REQUIREMENTS

Perl 5 (hopefully at least 5.00503) or later.

You must have the LWP Bundle on your computer. Load the LWP::Bundle via the CPAN.pm shell, or download libwww-perl-x.xx from CPAN (or via ActiveState's ppm utility). Also required is the the HTML-Parser-x.xx bundle of modules also from CPAN (and from ActiveState for Windows).

```
 http://search.cpan.org/search?dist=libwww-perl
 http://search.cpan.org/search?dist=HTML-Parser
```

You will also need Digest::MD5 if you wish to use the MD5 feature. HTML::Tagset is also required. Other modules may be required (for example, the pod2xml.pm module has its own requirementes -- see perldoc pod2xml for info).

The spider.pl script, like everyone else, expects perl to live in /usr/local/bin. If this is not the case then either add a symlink at /usr/local/bin/perl to point to where perl is installed or modify the shebang (#!) line at the top of the spider.pl program.

Note that the libwww-perl package does not support SSL (Secure Sockets Layer) (https) by default. See *README.SSL* included in the libwww-perl package for information on installing SSL support.

14.4 CONFIGURATION FILE

The spider configuration file is a read by the script as Perl code. This makes the configuration a bit more complex than simple text config files, but allows the spider to be configured programmatically.

For example, the config file can contain logic for testing URLs against regular expressions or even against a database lookup while running.

The configuration file sets an array called `@servers`. This array can contain one or more hash structures of parameters. Each hash structure is a configuration for a single server.

Here's an example:

```
my %main_site = (
    base_url   => 'http://example.com',
    same_hosts => 'www.example.com',
    email      => 'admin@example.com',
);
```

```
my %news_site = (
    base_url   => 'http://news.example.com',
    email      => 'admin@example.com',
);
```

```
@servers = ( \%main_site, \%news_site );
1;
```

The above defines two Perl hashes (%main_site and %news_site) and then places a *reference* (the backslash before the name of the hash) to each of those hashes in the `@servers` array. The "1;" at the end is required at the end of the file (Perl must see a true value at the end of the file).

The `config file path` is the first parameter passed to the spider script.

```
./spider.pl F<config>
```

If you do not specify a config file then the spider will look for the file *SwishSpiderConfig.pl* in the current directory.

The Swish-e distribution includes a *SwishSpiderConfig.pl* file with a few example configurations. This example file is installed in the *prog-bin/* documentation directory (on unix often this is /usr/local/share/swish-e/prog-bin).

When the special config file name "default" is used:

```
SwishProgParameters default http://www.mysite/index.html [<URL>] [...]
```

Then a default set of parameters are used with the spider. This is a good way to start using the spider before attempting to create a configuration file.

The default settings skip any urls that look like images (well, .gif .jpeg .png), and attempts to filter PDF and MS Word documents IF you have the required filter programs installed (which are not part of the Swish-e distribution). The spider will follow "a" and "frame" type of links only.

Note that if you do use a spider configuration file that the default configuration will NOT be used (unless you set the "use_default_config" option in your config file).

14.5 CONFIGURATION OPTIONS

This describes the required and optional keys in the server configuration hash, in random order...

base_url

This required setting is the starting URL for spidering.

This sets the first URL the spider will fetch. It does NOT limit spidering to URLs at or below the level of the directory specified in this setting. For that feature you need to use the test_url callback function.

Typically, you will just list one URL for the base_url. You may specify more than one URL as a reference to a list and each will be spidered:

```
    base_url => [qw! http://swish-e.org/ http://othersite.org/other/index.html !],
```

but each site will use the same config opions. If you want to index two separate sites you will likely rather add an additional configuration to the `@servers` array.

You may specify a username and password:

```
    base_url => 'http://user:pass@swish-e.org/index.html',
```

If a URL is protected by Basic Authentication you will be prompted for a username and password. The parameter max_wait_time controls how long to wait for user entry before skipping the current URL. See also credentials below.

same_hosts

This optional key sets equivalent **authority** `name(s)` for the site you are spidering. For example, if your site is `www.mysite.edu` but also can be reached by `mysite.edu` (with or without `www`) and also `web.mysite.edu` then:

Example:

```
    $serverA{base_url} = 'http://www.mysite.edu/index.html';
    $serverA{same_hosts} = ['mysite.edu', 'web.mysite.edu'];
```

Now, if a link is found while spidering of:

```
    http://web.mysite.edu/path/to/file.html
```

it will be considered on the same site, and will actually spidered and indexed as:

```
http://www.mysite.edu/path/to/file.html
```

Note: This should probably be called **same_host_port** because it compares the URI `host:port` against the list of host names in same_hosts. So, if you specify a port name in you will want to specify the port name in the the list of hosts in same_hosts:

```
my %serverA = (
    base_url    => 'http://sunsite.berkeley.edu:4444/',
    same_hosts  => [ qw/www.sunsite.berkeley.edu:4444/ ],
    email       => 'my@email.address',
);
```

email

This required key sets the email address for the spider. Set this to your email address.

agent

This optional key sets the name of the spider.

link_tags

This optional tag is a reference to an array of tags. Only links found in these tags will be extracted. The default is to only extract links from >a< tags.

For example, to extract tags from `a` tags and from `frame` tags:

```
my %serverA = (
    base_url    => 'http://sunsite.berkeley.edu:4444/',
    same_hosts  => [ qw/www.sunsite.berkeley.edu:4444/ ],
    email       => 'my@email.address',
    link_tags   => [qw/ a frame /],
);
```

use_default_config

This option is new for Swish-e 2.4.3.

The spider has a hard-coded default configuration that's available when the spider is run with the configuration file listed as "default":

```
./spider.pl default <url>
```

This default configuration skips urls that match the regular expression:

```
/\.(?:gif|jpeg|png)$/i
```

and the spider will attempt to use the SWISH::Filter module for filtering non-text documents. (You still need to install programs to do the actual filtering, though).

Here’s the basic config for the "default" mode:

```
@servers = (
{
    email               => ’swish@user.failed.to.set.email.invalid’,
    link_tags           => [qw/ a frame /],
    keep_alive          => 1,
    test_url            => sub {  $_[0]->path !~ /\.(?:gif|jpeg|png)$/i },
    test_response       => $response_sub,
    use_head_requests   => 1,  # Due to the response sub
    filter_content      => $filter_sub,
} );
```

The filter_content callback will be used if SWISH::Filter was loaded and ready to use. This doesn’t mean that filtering will work automatically -- you will likely need to install aditional programs for filtering (like Xpdf or Catdoc).

The test_response callback will be set to test if a given content type can be filtered by SWISH::Filter (if SWISH::Filter was loaded), otherwise, it will check for content-type of text/* -- any text type of document.

Normally, if you specify your own config file:

```
./spider.pl my_own_spider.config
```

then you must setup those features available in the default setting in your own config file. But, if you wish to build upon the "default" config file then set this option.

For example, to use the default config but specify your own email address:

```
@servers = (
    {
        email               => my@email.address,
        use_default_config  => 1,
        delay_sec           => 0,
    },
);
1;
```

What this does is "merge" your config file with the default config file.

delay_sec

This optional key sets the delay in seconds to wait between requests. See the LWP::RobotUA man page for more information. The default is 5 seconds. Set to zero for no delay.

When using the keep_alive feature (recommended) the delay will be used only where the previous request returned a "Connection: closed" header.

delay_min (deprecated)

Set the delay to wait between requests in minutes. If both delay_sec and delay_min are defined, delay_sec will be used.

max_wait_time

This setting is the number of seconds to wait for data to be returned from the request. Data is returned in chunks to the spider, and the timer is reset each time a new chunk is reported. Therefore, documents (requests) that take longer than this setting should not be aborted as long as some data is received every max_wait_time seconds. The default it 30 seconds.

NOTE: This option has no effect on Windows.

max_time

This optional key will set the max minutes to spider. Spidering for this host will stop after max_time minutes, and move on to the next server, if any. The default is to not limit by time.

max_files

This optional key sets the max number of files to spider before aborting. The default is to not limit by number of files. This is the number of requests made to the remote server, not the total number of files to index (see max_indexed). This count is displayted at the end of indexing as `Unique URLs`.

This feature can (and perhaps should) be use when spidering a web site where dynamic content may generate unique URLs to prevent run-away spidering.

max_indexed

This optional key sets the max number of files that will be indexed. The default is to not limit. This is the number of files sent to swish for indexing (and is reported by `Total Docs` when spidering ends).

max_size

This optional key sets the max size of a file read from the web server. This **defaults** to 5,000,000 bytes. If the size is exceeded the resource is skipped and a message is written to STDERR if the DEBUG_SKIPPED debug flag is set.

Set max_size to zero for unlimited size. If the server returns a Content-Length header then that will be used. Otherwise, the document will be checked for size limitation as it arrives. That's a good reason to have your server send Content-Length headers.

See also use_head_requests below.

keep_alive

This optional parameter will enable keep alive requests. This can dramatically speed up spidering and reduce the load on server being spidered. The default is to not use keep alives, although enabling it will probably be the right thing to do.

To get the most out of keep alives, you may want to set up your web server to allow a lot of requests per single connection (i.e MaxKeepAliveRequests on Apache). Apache's default is 100, which should be good.

When a connection is not closed the spider does not wait the "delay_sec" time when making the next request. In other words, there is no delay in requesting documents while the connection is open.

Note: try to filter as many documents as possible **before** making the request to the server. In other words, use test_url to look for files ending in `.html` instead of using test_response to look for a content type of `text/html` if possible. Do note that aborting a request from test_response will break the current keep alive connection.

Note: you must have at least libwww-perl-5.53_90 installed to use this feature.

use_head_requests

This option is new as of swish-e 2.4.3 and can effect the speed of spidering and the load of the web server.

To understand this you will likely need to read about the CALLBACK FUNCTIONS below -- specifically about the test_response callback function. This option is also only used when keep_alive is also enabled (although it could be debated that it's useful without keep alives).

This option tells the spider to use http HEAD requests before each request.

Normally, the spider simply does a GET request and after receiving the first chunk of data back from the web server calls the test_response callback function (if one is defined in your config file). The test_response callback function is a good place to test the content-type header returned from the server and reject types that you do not want to index.

Now, *if* you are using the keep_alive feature then rejecting a document will often (always?) break the keep alive connection.

So, what the use_head_requests option does is issue a HEAD request for every document, checks for a Content-Length header (to check if the document is larger than max_size, and then calls your test_response callback function. If your callback function returns true then a GET request is used to fetch the document.

The idea is that by using HEAD requests instead of GET request a false return from your test_response callback function (i.e. rejecting the document) will not break the keep alive connection.

Now, don’t get too excited about this. Before using this think about the ratio of rejected documents to accepted documents. If you reject no documents then using this feature will double the number of requests to the web server -- which will also double the number of connections to the web server. But, if you reject a large percentage of documents then this feature will help maximize the number of keep alive requests to the server (i.e. reduce the number of separate connections needed).

There’s also another problem with using HEAD requests. Some broken servers may not respond correctly to HEAD requests (some issues a 500 error), but respond fine to a normal GET request. This is something to watch out for.

Finally, if you do not have a test_response callback AND max_size is set to zero then setting use_head_requests will have no effect.

And, with all other factors involved you might find this option has no effect at all.

skip

This optional key can be used to skip the current server. It’s only purpose is to make it easy to disable a specific server hash in a configuration file.

debug

Set this item to a comma-separated list of debugging options.

Options are currently:

```
    errors, failed, headers, info, links, redirect, skipped, url
```

Here are basically the levels:

```
    errors      =>   general program errors (not used at this time)
    url         =>   print out every URL processes
    headers     =>   prints the response headers
    failed      =>   failed to return a 200
    skipped     =>   didn’t index for some reason
    info        =>   a little more verbose
    links       =>   prints links as they are extracted
    redirect    =>   prints out redirected URLs
```

Debugging can be also be set by an environment variable SPIDER_DEBUG when running *spider.pl*. You can specify any of the above debugging options, separated by a comma.

For example with Bourne type shell:

```
SPIDER_DEBUG=url,links spider.pl [....]
```

Before Swish-e 2.4.3 you had to use the internal debugging constants or'ed together like so:

```
debug => DEBUG_URL | DEBUG_FAILED | DEBUG_SKIPPED,
```

You can still do this, but the string version is easier. In fact, if you want to turn on debugging dynamically (for example in a `test_url()` callback function) then you currently *must* use the DEBUG_* constants. The string is converted to a number only at the start of spiderig -- after that the debug parameter is converted to a number.

quiet

If this is true then normal, non-error messages will be supressed. Quiet mode can also be set by setting the environment variable SPIDER_QUIET to any true value.

```
SPIDER_QUIET=1
```

max_depth

The max_depth parameter can be used to limit how deeply to recurse a web site. The depth is just a count of levels of web pages decended, and not related to the number of path elements in a URL.

A max_depth of zero says to only spider the page listed as the base_url. A max_depth of one will spider the base_url page, plus all links on that page, and no more. The default is to spider all pages.

ignore_robots_file

If this is set to true then the robots.txt file will not be checked when spidering this server. Don't use this option unless you know what you are doing.

use_cookies

If this is set then a "cookie jar" will be maintained while spidering. Some (poorly written ;) sites require cookies to be enabled on clients.

This requires the HTTP::Cookies module.

use_md5

If this setting is true, then a MD5 digest "fingerprint" will be made from the content of every spidered document. This digest number will be used as a hash key to prevent indexing the same content more than once. This is helpful if different URLs generate the same content.

Obvious example is these two documents will only be indexed one time:

```
        http://localhost/path/to/index.html
        http://localhost/path/to/
```

This option requires the Digest::MD5 module. Spidering with this option might be a tiny bit slower.

validate_links

Just a hack. If you set this true the spider will do HEAD requests all links (e.g. off-site links), just to make sure that all your links work.

credentials

You may specify a username and password to be used automatically when spidering:

```
        credentials => 'username:password',
```

A username and password supplied in a URL will override this setting. This username and password will be used for every request.

See also the get_password callback function below. get_password, if defined, will be called when a page requires authorization.

credential_timeout

Sets the number of seconds to wait for user input when prompted for a username or password. The default is 30 seconds.

Set this to zero to wait forever. Probably not a good idea.

Set to undef to disable asking for a password.

```
        credential_timeout => undef,
```

remove_leading_dots

Removes leading dots from URLs that might reference documents above the document root. The default is to not remove the dots.

14.6 CALLBACK FUNCTIONS

Callback functions can be defined in your parameter hash. These optional settings are *callback* subroutines that are called while processing URLs.

A little perl discussion is in order:

In perl, a scalar variable can contain a reference to a subroutine. The config example above shows that the configuration parameters are stored in a perl *hash*.

```
    my %serverA = (
        base_url    => 'http://sunsite.berkeley.edu:4444/',
        same_hosts  => [ qw/www.sunsite.berkeley.edu:4444/ ],
        email       => 'my@email.address',
        link_tags   => [qw/ a frame /],
    );
```

There's two ways to add a reference to a subroutine to this hash:

sub foo { return 1; }

```
    my %serverA = (
        base_url    => 'http://sunsite.berkeley.edu:4444/',
        same_hosts  => [ qw/www.sunsite.berkeley.edu:4444/ ],
        email       => 'my@email.address',
        link_tags   => [qw/ a frame /],
        test_url    => \&foo,  # a reference to a named subroutine
    );
```

Or the subroutine can be coded right in place:

```
    my %serverA = (
        base_url    => 'http://sunsite.berkeley.edu:4444/',
        same_hosts  => [ qw/www.sunsite.berkeley.edu:4444/ ],
        email       => 'my@email.address',
        link_tags   => [qw/ a frame /],
        test_url    => sub { reutrn 1; },
    );
```

The above example is not very useful as it just creates a user callback function that always returns a true value (the number 1). But, it's just an example.

The function calls are wrapped in an eval, so calling die (or doing something that dies) will just cause that URL to be skipped. If you really want to stop processing you need to set $server->{abort} in your subroutine (or send a kill -HUP to the spider).

The first two parameters passed are a URI object (to have access to the current URL), and a reference to the current server hash. The `server` hash is just a global hash for holding data, and useful for setting flags as described below.

Other parameters may be also passed in depending the the callback function, as described below. In perl parameters are passed in an array called "@_". The first element (first parameter) of that array is $_[0], and the second is $_[1], and so on. Depending on how complicated your function is you may wish to shift your parameters off of the `@_` list to make working with them easier. See the examples below.

To make use of these routines you need to understand when they are called, and what changes you can make in your routines. Each routine deals with a given step, and returning false from your routine will stop processing for the current URL.

test_url

test_url allows you to skip processing of urls based on the url before the request to the server is made. This function is called for the base_url links (links you define in the spider configuration file) and for every link extracted from a fetched web page.

This function is a good place to skip links that you are not interested in following. For example, if you know there's no point in requesting images then you can exclude them like:

```
    test_url => sub {
        my $uri = shift;
        return 0 if $uri->path =~ /\.(gif|jpeg|png)$/;
        return 1;
    },
```

Or to write it another way:

```
    test_url => sub { $_[0]->path !~ /\.(gif|jpeg|png)$/ },
```

Another feature would be if you were using a web server where path names are NOT case sensitive (e.g. Windows). You can normalize all links in this situation using something like

```
    test_url => sub {
        my $uri = shift;
        return 0 if $uri->path =~ /\.(gif|jpeg|png)$/;
```

```
        $uri->path( lc $uri->path ); # make all path names lowercase
        return 1;
    },
```

The important thing about test_url (compared to the other callback functions) is that it is called while *extracting* links, not while actually fetching that page from the web server. Returning false from test_url simple says to not add the URL to the list of links to spider.

You may set a flag in the server hash (second parameter) to tell the spider to abort processing.

```
    test_url => sub {
        my $server = $_[1];
        $server->{abort}++ if $_[0]->path =~ /foo\.html/;
        return 1;
    },
```

You cannot use the server flags:

```
no_contents
no_index
no_spider
```

This is discussed below.

test_response

This function allows you to filter based on the response from the remote server (such as by content-type).

Web servers use a Content-Type: header to define the type of data returned from the server. On a web server you could have a .jpeg file be a web page -- file extensions may not always indicate the type of the file.

If you enable use_head_requests then this function is called after the spider makes a HEAD request. Otherwise, this function is called while the web pages is being fetched from the remote server, typically after just enought data has been returned to read the response from the web server.

The test_response callback function is called with the following parameters:

```
( $uri, $server, $response, $content_chunk )
```

The `$response` variable is a HTTP::Response object and provies methods of examining the server's response. The `$content_chunk` is the first chunk of data returned from the server (if not a HEAD request).

When not using use_head_requests the spider requests a document in "chunks" of 4096 bytes. 4096 is only a suggestion of how many bytes to return in each chunk. The test_response routine is called when the first chunk is received only. This allows ignoring (aborting) reading of a very large file, for example, without having to read the entire file. Although not much use, a reference to this chunk is passed as the forth parameter.

If you are spidering a site with many different types of content that you do not wish to index (and cannot use a test_url callback to determine what docs to skip) then you will see better performance using both the use_head_requests and keep_alive features. (Aborting a GET request kills the keep-alive session.)

For example, to only index true HTML (text/html) pages:

```
test_response => sub {
    my $content_type = $_[2]->content_type;
    return $content_type =~ m!text/html!;
},
```

You can also set flags in the server hash (the second parameter) to control indexing:

```
    no_contents -- index only the title (or file name), and not the contents
    no_index    -- do not index this file, but continue to spider if HTML
    no_spider   -- index, but do not spider this file for links to follow
    abort       -- stop spidering any more files
```

For example, to avoid index the contents of "private.html", yet still follow any links in that file:

```
    test_response => sub {
        my $server = $_[1];
        $server->{no_index}++ if $_[0]->path =~ /private\.html$/;
        return 1;
    },
```

Note: Do not modify the URI object in this call back function.

filter_content

This callback function is called right before sending the content to swish. Like the other callback function, returning false will cause the URL to be skipped. Setting the `abort` server flag and returning false will abort spidering.

You can also set the `no_contents` flag.

This callback function is passed four parameters. The URI object, server hash, the HTTP::Response object, and a reference to the content.

You can modify the content as needed. For example you might not like upper case:

```
    filter_content => sub {
        my $content_ref = $_[3];
```

```
        $$content_ref = lc $$content_ref;
        return 1;
    },
```

I more reasonable example would be converting PDF or MS Word documents for parsing by swish. Examples of this are provided in the *prog-bin* directory of the swish-e distribution.

You may also modify the URI object to change the path name passed to swish for indexing.

```
    filter_content => sub {
        my $uri = $_[0];
        $uri->host('www.other.host') ;
        return 1;
    },
```

Swish-e's ReplaceRules feature can also be used for modifying the path name indexed.

Note: Swish-e now includes a method of filtering based on the SWISH::Filter Perl modules. See the SwishSpiderConfig.pl file for an example how to use SWISH::Filter in a filter_content callback function.

If you use the "default" configuration (i.e. pass "default" as the first parameter to the spider) then SWISH::Filter is used automatically. This only adds code for calling the programs to filter your content -- you still need to install applications that do the hard work (like xpdf for pdf conversion and catdoc for MS Word conversion).

The a function included in the *spider.pl* for calling SWISH::Filter when using the "default" config can also be used in your config file. There's a function called `swish_filter()` that returns a list of two subroutines. So in your config you could do:

```
    my ($filter_sub, $response_sub ) = swish_filter();
```

```
    @server = ( {
        test_response   => $response_sub,
        filter_content  => $filter_sub,
        [...],
    } );
```

The `$response_sub` is not required, but is useful if using HEAD requests (use_head_requests): It tests the content type from the server to see if there's any filters that can handle the document. The `$filter_sub` does all the work of filtering a document.

Make sense? If not, then that's what the Swish-e list is for.

spider_done

This callback is called after processing a server (after each server listed in the `@servers` array if more than one).

This allows your config file to do any cleanup work after processing. For example, if you were keeping counts during, say, a `test_response()` callback function you could use the `spider_done()` callback to print the results.

output_function

If defined, this callback function is called instead of printing the content and header to STDOUT. This can be used if you want to store the output of the spider before indexing.

The output_function is called with the following parameters:

```
   ($server, $content, $uri, $response, $bytecount, $path);
```

Here is an example that simply shows two of the params passed:

```
    output_function => sub {
        my ($server, $content, $uri, $response, $bytecount, $path) = @_;
        print STDERR  "passed: uri $uri, bytecount $bytecount...\n";
        # no output to STDOUT for swish-e
    }
```

You can do almost the same thing with a filter_content callback.

get_password

This callback is called when a HTTP password is needed (i.e. after the server returns a 401 error). The function can test the URI and Realm and then return a username and password separated by a colon:

```
    get_password => sub {
        my ( $uri, $server, $response, $realm ) = @_;
        if ( $uri->path =~ m!^/path/to/protected! && $realm eq 'private' ) {
            return 'joe:secret931password';
        }
        return;  # sorry, I don't know the password.
    },
```

Use the credentials setting if you know the username and password and they will be the same for every request. That is, for a site-wide password.

Note that you can create your own counters to display in the summary list when spidering is finished by adding a value to the hash pointed to by `$server-`{counts}>.

```
    test_url => sub {
        my $server = $_[1];
        $server->{no_index}++ if $_[0]->path =~ /private\.html$/;
        $server->{counts}{'Private Files'}++;
        return 1;
    },
```

Each callback function **must** return true to continue processing the URL. Returning false will cause processing of *the current* URL to be skipped.

14.6.1 More on setting flags

Swish (not this spider) has a configuration directive NoContents that will instruct swish to index only the title (or file name), and not the contents. This is often used when indexing binary files such as image files, but can also be used with html files to index only the document titles.

As shown above, you can turn this feature on for specific documents by setting a flag in the server hash passed into the test_response or filter_content subroutines. For example, in your configuration file you might have the test_response callback set as:

```
    test_response => sub {
        my ( $uri, $server, $response ) = @_;
        # tell swish not to index the contents if this is of type image
        $server->{no_contents} = $response->content_type =~ m[^image/];
        return 1;  # ok to index and spider this document
    }
```

The entire contents of the resource is still read from the web server, and passed on to swish, but swish will also be passed a No-Contents header which tells swish to enable the NoContents feature for this document only.

Note: Swish will index the path name only when NoContents is set, unless the document's type (as set by the swish configuration settings IndexContents or DefaultContents) is HTML *and* a title is found in the html document.

Note: In most cases you probably would not want to send a large binary file to swish, just to be ignored. Therefore, it would be smart to use a filter_content callback routine to replace the contents with single character (you cannot use the empty string at this time).

A similar flag may be set to prevent indexing a document at all, but still allow spidering. In general, if you want completely skip spidering a file you return false from one of the callback routines (test_url, test_response, or filter_content). Returning false from any of those three callbacks will stop processing of that file, and the file will **not** be spidered.

But there may be some cases where you still want to spider (extract links) yet, not index the file. An example might be where you wish to index only PDF files, but you still need to spider all HTML files to find the links to the PDF files.

```
    $server{test_response} = sub {
        my ( $uri, $server, $response ) = @_;
        $server->{no_index} = $response->content_type ne 'application/pdf';
        return 1;  # ok to spider, but don't index
    }
```

So, the difference between `no_contents` and `no_index` is that `no_contents` will still index the file name, just not the contents. `no_index` will still spider the file (if it's `text/html`) but the file will not be processed by swish at all.

Note: If `no_index` is set in a test_response callback function then the document *will not be filtered*. That is, your filter_content callback function will not be called.

The `no_spider` flag can be set to avoid spiderering an HTML file. The file will still be indexed unless `no_index` is also set. But if you do not want to index and spider, then simply return false from one of the three callback funtions.

14.7 SIGNALS

Sending a SIGHUP to the running spider will cause it to stop spidering. This is a good way to abort spidering, but let swish index the documents retrieved so far.

14.8 CHANGES

List of some of the changes

14.8.1 Thu Sep 30 2004 - changes for Swish-e 2.4.3

Code reorganization and a few new featues. Updated docs a little tiny bit. Introduced a few spelling mistakes.

Config opiton: use_default_config

It used to be that you could run the spider like:

```
    spider.pl default <some url>
```

and the spider would use its own internal config. But if you used your own config file then the defaults were not used. This options allows you to merge your config with the default config. Makes making small changes to the default easy.

Config option: use_head_requests

Tells the spider to make a HEAD request before GET'ing the document from the web server. Useful if you use keep_alive and have a `test_response()` callback that rejects many documents (which breaks the connection).

Config option: spider_done

Callback to tell you (or tell your config as it may be) that the spider is done. Useful if you need to do some extra processing when done spidering -- like record counts to a file.

Config option: get_password

This callback is called when a document returns a 401 error needing a username and password. Useful if spidering a site proteced with multiple passwords.

Config option: output_function

If defined spider.pl calls this instead of sending ouptut to STDOUT.

Config option: debug

Now you can use the words instead of or’ing the DEBUG_* constants together.

14.9 TODO

Add a "get_document" callback that is called right before making the "GET" request. This would make it easier to use cached documents. You can do that now in a test_url callback or in a test_response when using HEAD request.

Save state of the spider on SIGHUP so spidering could be restored at a later date.

14.10 COPYRIGHT

14.11 SUPPORT

Send all questions to the The SWISH-E discussion list.

See http://sunsite.berkeley.edu/SWISH-E.

15 SWISH::Filter - Perl extension for filtering documents with Swish-e

15.1 SYNOPSIS

```
use SWISH::Filter;
```

```
# load available filters into memory
my $filter = SWISH::Filter->new;
```

```
# convert a document
```

```
my $doc = $filter->convert(
      document     => \$scalar_ref,  # path or ref to a doc
      content_type => $content_type, # content type if doc reference
      name         => $real_path,    # optional name for this file (useful for debugging)
      user_data    => $whatever,     # optional data to make available to filters
 );
```

```
return unless $doc;  # empty doc, zero size, or no filters installed
```

```
# Was the document converted by a filter?
my $was_filtered = $doc->was_filtered;
```

```
# Skip if the file is not text
return if $doc->is_binary;
```

```
# Print out the doc
my $doc_ref = $doc->fetch_doc;
print $$doc_ref;
```

```
# Fetch the final content type of the document
my $content_type = $doc->content_type;
```

```
# Fetch Swish-e parser type (TXT*, XML*, HTML*, or undefined)
my $doc_type = $doc->swish_parser_type;
```

15.2 DESCRIPTION

SWISH::Filter provides a unified way to convert documents into a type that Swish-e can index. Individual filters are installed as separate perl modules. For example, there might be a filter that converts from PDF format to HTML format.

Note that this is just a framework for filtering documents. Additional helper programs or Perl module may need to be installed to use SWISH::Filter to filter documents. For example, to filter PDF documents you must install the Xpdf package.

The filters are automatically loaded when `SWISH::Filters->new()` is called. Filters define a type and priority that determines the processing order of the filter. Filters are processed in this sort order until a filter accepts the document for filtering. The filter uses the document's content type to determine if the filter should handle the current document. The content-type is determined by the files suffix if not supplied by the calling program.

The individual filters are not designed to be used as separate modules. All access to the filters is through this SWISH::Filter module.

Normally, once a document is filtered processing stops. Filters can filter the document and then set a flag saying that filtering should continue (for example a filter that uncompresses a MS Word document before passing on to the filter that converts from MS Word to text). All this should be transparent to the end user. So, filters can be pipe-lined.

The idea of SWISH::Filter is that new filters can be created, and then downloaded and installed to provide new filtering capabilities. For example, if you needed to index MS Excel documents you might be able to download a filter from the Swish-e site and magically next time you run indexing MS Excel docs would be indexed.

The SWISH::Filter setup can be used with -S prog or -S http. It works best with the -S prog method because the filter modules only need to be loaded and compiled one time. The -S prog program *spider.pl* will automatically use SWISH::Filter when spidering with default settings (using "default" as the first parameter to spider.pl).

The -S http indexing method uses a Perl helper script called *swishspider*. *swishspider* has been updated to work with SWISH::Filter, but (unlike spider.pl) does not contain a "use lib" line to point to the location of SWISH::Filter. This means that by default *swishspider* will **not** use SWISH::Filter for filtering. The reason for this is because *swishspider* runs for every URL fetched, and loading the Filters for each document can be slow. The recommended way of spidering is using -S prog with spider.pl, but if -S http is desired the way to enable SWISH::Filter is to set PERL5LIB before running swish so that *swishspider* will be able to locate the SWISH::Filter module. Here's one way to set the PERL5LIB with the bash shell:

```
$ export PERL5LIB=‘swish-filter-test -path‘
```

15.3 METHODS

$filter = SWISH::Filter->new()

This creates a SWISH::Filter object. You may pass in options as a list or a hash reference.

15.3.1 SWISH::Filter->new Options

There is currently only one option that can be passed in to `new()`:

ignore_filters

Pass in a reference to a list of filter names to ignore. For example, if you have two filters installed "Pdf2HTML" and "Pdf2XML" and want to avoid using "Pdf2XML":

```
    my $filter = SWISH::Filter->new( ignore_filters => [’Pdf2XML’];
```

$doc_object = $filter->convert();

This method filters a document. Returns an object of the class SWISH::Filter::document or undefined if passed in an empty document, a filename that cannot be read off disk, or if no filters have been loaded.

SWISH::Filter::document methods listed below can be called on the object to, for example, check if the document was filtered and to fetch the document content (filtered or not).

You must pass in a hash (or hash reference) of parameters to the `convert()` method. The possible parameters are:

document

This can be either a path to a file, or a scalar reference to a document in memory. This is required.

content_type

The MIME type of the document. This is only required when passing in a scalar reference to a document. The content type string is what the filters use to match a document type.

When passing in a file name and content_type is not set, then the content type will be determined from the file’s extension by using the MIME::Types Perl module (available on CPAN).

name

Optional name to pass in to filters that will be used in error and warning messages.

user_data

Optional data structure that all filters may access. This can be fetched in a filter by:

```
    my $user_data = $doc_object->user_data;
```

And used in the filter as:

```
    if ( ref $user_data && $user_data->{pdf2html}{title} ) {
        ...
    }
```

It’s up to the filter author to use a unique first-level hash key for a given filter.

Example of using the `convert()` method:

```
    $doc_object = $filter->convert(
        document     => $doc_ref,
        content-type => 'application/pdf',
    );
```

$filter->mywarn()

Internal function used for writing warning messages to STDERR if $ENV{FILTER_DEBUG} is set. Set the environment variable FILTER_DEBUG before running to see extra messages while processing.

@filters = $filter->filter_list;

Returns a list of filter objects installed.

@filter = $filter->can_filter($content_type);

This is useful for testing to see if a mimetype might be handled by SWISH::Filter wihtout having to pass in a document. Helpful if doing HEAD requests.

Returns an array of filters that can handle this type of document

15.4 WRITING FILTERS

Filters are standard perl modules that are installed into the SWISH::Filters name space. Filters are not complicated -- see the existing filters for examples.

Each filter defines the content-types (or mimetypes) that it can handle. These are specified as a list of regular expressions to match against the document’s content-type. If one of the mimetypes of a filter match the incoming document’s content-type the filter is called. The filter can then either filter the content or return undefined indicating that it decided not to filter the document for some reason. If the document is converted the filter returns either a reference to a scalar of the content or a file name where the content is stored. The filter also must change the content-type of the document to reflect the new document.

Filters typically use external programs or modules to do that actual work of converting a document from one type to another. For example, programs in the Xpdf packages are used for converting PDF files. The filter can (and should) test for those programs in its `new()` method.

Filters also can define a type and priority. These attributes are used to set the order filters are tested for a content-type match. This allows you to have more than one filter that can work on the same content-type.

If a filter calls `die()` then the filter is removed from the chain and will not be called again *during the same run*. Calling die when running with -S http or -S fs has no effect since the program is run once per document.

Once a filter returns something other than undef no more filters will be called. If the filter calls $filter->set_continue then processing will continue as if the file was not filtered. For example, a filter can uncompress data and then set $filter->set_continue and let other filters process the document.

This is the list of methods the filter should or may define (as specificed):

new() * required *

This method returns either an object which provides access to the filter, or undefined if the filter is not to be used.

The `new()` method is a good place to check for required modules or helper programs. Returning undefined prevents the filter from being included in the filter chain.

The new method must return a blessed hash reference. The only required attribute is **mimetypes**. This attribute must contain a reference to an array of regular expressions used for matching the content-type of the document passed in.

Example:

```
    sub new {
        my ( $class ) = @_;
```

```
        # List of regular expressions
        my @mimetypes = (
            qr[application/(x-)?msword],
            qr[application/worddoc],
        );
```

```
        my %settings = (
            mimetypes   => \@mimetypes,
```

```
            # Optional settings
            priority    => 20,
            type        => 2,
        );
```

```
            return bless \%settings, $class;
        }
```

The attribute "mimetypes" returns an array reference to a list of regular expressions. Those patterns are matched against each document's content type.

filter() * required *

This is the function that does the work of converting a document from one content type to another. The function is passed the document object. See document object methods listed below for what methods may be called on a document.

The function can return undefined (or any false value) to indicate that the filter did not want to process the document. Other filters will then be tested for a content type match.

If the document is filtered then the filter must set the new document's content type (if it changed) and return either a file name where the document can be found or a reference to a scalar containing the document.

type()

Returns a number. Filters are sorted (for processing in a specific order) and this number is simply the primary key used in sorting. If not specified the filter's type used for sorting is 2.

This is an optional method. You can also set the type in your `new()` constructor as shown above.

priority()

Returns a number. Filters are sorted (for processing in a specific order) and this number is simply the secondary key used in sorting. If not specified the filter's priority is 50.

This is an optional method. You can also set the priority in your `new()` constructor as shown above.

Again, the point of the `type()` and `priority()` methods is to allow setting the sort order of the filters. Useful if you have two filters for filtering the same content-type, but prefer to use one over the other. Neither are required.

Here's a module to convert MS Word documents using the program "catdoc":

```
    package SWISH::Filters::Doc2txt;
    use vars qw/ $VERSION /;
```

```
    $VERSION = '0.02';
```

```
sub new {
    my ( $class ) = @_;

    my $self = bless {
        mimetypes   => [ qr!application/(x-)?msword! ],
        priority    => 50,
    }, $class;

    # check for helpers
    return $self->set_programs( 'catdoc' );

}

sub filter {
    my ( $self, $doc ) = @_;

    my $content = $self->run_catdoc( $doc->fetch_filename ) || return;

    # update the document's content type
    $filter->set_content_type( 'text/plain' );

    # return the document
    return \$content;
}
1;
```

The `new()` constructor creates a blessed hash which contains an array reference of mimetypes patterns that this filter accepts. The priority sets this filter to run after any other filters that might handle the same type of content. The *set_programs()* function says that we need to call a program called "catdoc". The function either returns `$self` or undefined if catdoc could not be found. The *set_programs()* function creates a new method for running catdoc.

The filter function runs catdoc passing in the name of the file (if the file is in memory a temporary file is created). That *run_catdoc()* function was created by the *set_programs()* call above.

15.5 SWISH::Filter::document Methods

These methods are available to Filter authors, and also provide access to the document after calling the `convert()` method to end-users of SWISH::Filter.

End users of SWISH::Filter will use a subset of these methods. Mostly:

```
$doc_object->fetch_doc      # and alias for fetch_document_reference()
$doc_object->was_filtered   # true the document was filtered
$doc_object->content_type   # document’s current content type (mime type)
$doc_object->swish_parser_type # returns a parser type to use with Swish-e -S prog method
$doc_object->is_binary      # returns $content_type !~ m[^text/];
```

These methods are also available to the individual filter modules. The filter’s "filter" function is also passed a SWISH::Filter::document object. Method calls may be made on this object to check the document’s current content type, or to fetch the document as either a file name or a reference to a scalar containing the document content.

15.5.1 Methods used by end-users and filter authors

$doc_ref = $doc_object->fetch_doc_reference;

Returns a scalar reference to the document. This can be used when the filter can operate on the document in memory (or if an external program expects the input to be from standard input).

If the file is currently on disk then it will be read into memory. If the file was stored in a temporary file on disk the file will be deleted once read into memory. The file will be read in binmode if $doc->is_binary is true.

Note that $doc_object->fetch_doc is an alias.

$was_filtered = $doc_object->was_filtered

Returns true if some filter processed the document

$content_type = $doc_object->content_type;

Fetches the current content type for the document.

Example:

```
return unless $filter->content_type =~ m!application/pdf!;
```

$type = $doc_object->swish_parser_type

Returns a parser type based on the content type

$doc_object->is_binary

Returns true if the document’s content-type does not match "text/".

15.5.2 Methods used by filter authors

$file_name = $doc_object->fetch_filename;

Returns a path to the document as stored on disk. This name can be passed to external programs (e.g. catdoc) that expect input as a file name.

If the document is currently in memory then a temporary file will be created. Do not expect the file name passed to be the real path of the document.

The file will be written in binmode if $doc->is_binary is true.

This method is not normally used by end-users of SWISH::Filter.

$doc_object->set_continue;

Processing will continue to the next filter if this is set to a true value. This should be set for filters that change encodings or uncompress documents.

$doc_object->set_content_type($type);

Sets the content type for a document.

$doc_object->name

Fetches the name of the current file. This is useful for printing out the name of the file in an error message. This is the name passed in to the SWISH::Filter->convert method. It is optional and thus may not always be set.

```
    my $name = $doc_object->name || 'Unknown name';
    warn "File '$name': failed to convert -- file may be corrupt\n";
```

$doc_object->user_data

Fetches the the user_data passed in to the filter. This can be any data or data structure passed into SWISH::Filter->new.

This is an easy way to pass special parameters into your filters.

Example:

```
    my $data = $doc_object->user_data;
    # see if a choice for the <title> was passed in
    if ( ref $data eq 'HASH' && $data->{pdf2html}{title_field}  {
       ...
       ...
    }
```

15.6 SWISH::Filters::_BASE

Each filter is a subclass of SWISH::Filters::_BASE. A number of methods are available by default (and some can be overridden). Others are useful when writing your `new()` constructor.

$self->type

This method fetches the type of the filter. The value returned sets the primary sort key for sorting the filters. You can override this in your filter, or just set it as an attribute in your object. The default is 2.

The idea of the "type" is to create groups of filters, if needed. For example, you might have a set of filters that are used for uncompressing some documents before passing on to another group for filtering.

$self->priority

This method fetches the priority of the filter. The value returned sets the secondary sort key for sorting the filters. You can override this in your filter, or just set it as an attribute in your object. The default method returns 50.

The priority is useful if you have multiple filters for the same content type that use different methods for filtering (say one uses wvWare and another uses catdoc for filtering MS Word files). You might give the wvWare filter a lower priority number so it runs before the catdoc filter if both wvWare AND catdoc happen to be installed at the same time.

@types = $self->mimetypes

Returns the list of mimetypes (as regular expressions) set for the filter.

$pattern = $self->can_filter_mimetype($content_type)

Returns true if passed in content type matches one of the filter’s mimetypes Returns the pattern that matched.

mywarn($message)

method for printing out message if debugging is available

$boolean = $self->set_programs(@program_list);

Returns true if all the programs listed in `@program_list` are found and can be executed as the current user. Creates a method for each program with the "run_" prefix. Returns false is ANY program cannot be found.

Actually, it returns $self, so you can make it the last statement in your constructor.

So in your constructor you might do:

```
    return $self->set_programs( qw/ pdftotext pdfinfo / );
```

Then in your `filter()` method:

```
    my $content = $self->run_pdfinfo( $doc->fetch_filename, [options] );
```

$path = $self->find_binary($prog);

Use in a filter's `new()` method to test for a necesary program located in $PATH. Returns the path to the program or undefined if not found or does not pass the -x file test.

$bool = $self->use_modules(@module_list);

Attempts to load each of the module listed and calls its `import()` method.

Use to test and load required modules within a filter without aborting.

```
    return unless $self->use_modules( qw/ Spreadsheet::ParseExcel  HTML::Entities / );
```

A warning message is displayed if the FILTER_DEBUG environment variable is true. Returns `$self` if no error.

$doc_ref = $self->run_program($program, @args);

Runs `$program` with @args. Must pass in @args.

Under Windows calls IPC::Open2, which may pass data through the shell. Double-quotes are escaped (backslashed) and each parameter is wrapped in double-quotes.

On other platforms a fork and exec is used to avoid passing any data through the shell. Returns a reference to a scalar containing the output from your program, or dies.

This method is intended to read output from a program that converts one format into text. The output is read back in text mode -- on systems like Windows this means \r\n (CRLF) will be convertet to \n.

15.7 TESTING

Filters can be tested with the *swish-filter-test* program. Run:

```
    swish-filter-test -man
```

for documentation.

15.8 SUPPORT

Please contact the Swish-e discussion list. http://swish-e.org

15.9 Bugs, todo items, and other notes

TBD

15.10 AUTHOR

Bill Moseley

15.11 COPYRIGHT

Table of Contents: