Recoll user manual

COLLABORATORS

	TITLE : Recoll user manual	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jean-Francois Dockes	September 28, 2013	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Abstract

This document introduces full text search notions and describes the installation and use of the Recoll application. It currently describes Recoll 1.19.

Chapter 1

Introduction

1.1 Giving it a try

If you do not like reading manuals (who does?) and would like to give Recoll a try, just install the application and start the **recoll** graphical user interface (GUI), which will ask to index your home directory by default, allowing you to search immediately after indexing completes.

Do not do this if your home directory contains a huge number of documents and you do not want to wait or are very short on disk space. In this case, you may first want to customize the configuration to restrict the indexed area.

Also be aware that you may need to install the appropriate supporting applications for document types that need them (for example antiword for Microsoft Word files).

1.2 Full text search

Recoll is a full text search application. Full text search applications let you find your data by content rather than by external attributes (like a file name). More specifically, they will let you specify words (terms) that should or should not appear in the text you are looking for, and return a list of matching documents, ordered so that the most *relevant* documents will appear first.

You do not need to remember in what file or email message you stored a given piece of information. You just ask for related terms, and the tool will return a list of documents where these terms are prominent, in a similar way to Internet search engines.

A search application tries to determine which documents are most relevant to the search terms you provide. Computer algorithms for determining relevance can be very complex, and in general are inferior to the power of the human mind to rapidly determine relevance. The quality of relevance guessing is probably the most important aspect when evaluating a search application.

In many cases, you are looking for all the forms of a word, not for a specific form or spelling. These different forms may include plurals, different tenses for a verb, or terms derived from the same root or *stem* (example: floor, floors, floored, flooring...). Search applications usually expand queries to all such related terms (words that reduce to the same stem) and also provide a way to disable this expansion if you are actually searching for a specific form.

Stemming, by itself, does not accommodate for misspellings or phonetic searches. Recoll supports these features through a specific tool (the `term explorer`) which will let you explore the set of index terms along different modes.

1.3 Recoll overview

Recoll uses the Xapian information retrieval library as its storage and retrieval engine. Xapian is a very mature package using a sophisticated probabilistic ranking model. Recoll provides the mechanisms and interface to get data into and out of the system.

In practice, Xapian works by remembering where terms appear in your document files. The acquisition process is called indexing.

The resulting index can be big (roughly the size of the original document set), but it is not a document archive. Recoll can only display documents that still exist at the place from which they were indexed. (Actually, there is a way to reconstruct a document from the information in the index, but the result is not nice, as all formatting, punctuation and capitalization are lost).

Recoll stores all internal data in Unicode UTF-8 format, and it can index files with different character sets, encodings, and languages into the same index. It has input filters for many document types.

Stemming is the process by which Recoll reduces words to their radicals so that searching does not depend, for example, on a word being singular or plural (floor, floors), or on a verb tense (flooring, floored). Because the mechanisms used for stemming depend on the specific grammatical rules for each language, there is a separate Xapian stemmer module for most common languages where stemming makes sense.

Recoll stores the unstemmed versions of terms in the main index and uses auxiliary databases for term expansion (one for each stemming language), which means that you can switch stemming languages between searches, or add a language without needing a full reindex.

Storing documents written in different languages in the same index is possible, and commonly done. In this situation, you can specify several stemming languages for the index.

Recoll currently makes no attempt at automatic language recognition, which means that the stemmer will sometimes be applied to terms from other languages with potentially strange results. In practise, even if this introduces possibilities of confusion, this approach has been proven quite useful, and it is much less cumbersome than separating your documents according to what language they are written in.

Before version 1.18, Recoll stripped most accents and diacritics from terms, and converted them to lower case before either storing them in the index or searching for them. As a consequence, it was impossible to search for a particular capitalization of a term (`US` / `us`), or to discriminate two terms based on diacritics (`sake` / `saké`, `mate` / `maté`).

As of version 1.18, Recoll can optionally store the raw terms, without accent stripping or case conversion. In this configuration, it is still possible (and most common) for a query to be insensitive to case and/or diacritics. Appropriate term expansions are performed before actually accessing the main index. This is described in more detail in the section about index case and diacritics sensitivity.

Recoll has many parameters which define exactly what to index, and how to classify and decode the source documents. These are kept in configuration files. A default configuration is copied into a standard location (usually something like `/usr/[local/]share/recoll/examples`) during installation. The default values set by the configuration files in this directory may be overridden by values that you set inside your personal configuration, found by default in the `.recoll` sub-directory of your home directory. The default configuration will index your home directory with default parameters and should be sufficient for giving Recoll a try, but you may want to adjust it later, which can be done either by editing the text files or by using configuration menus in the **recoll** GUI. Some other parameters affecting only the **recoll** GUI are stored in the standard location defined by Qt.

The indexing process is started automatically the first time you execute the **recoll** GUI. Indexing can also be performed by executing the **recollindex** command.

Searches are usually performed inside the **recoll** GUI, which has many options to help you find what you are looking for. However, there are other ways to perform Recoll searches: mostly a command line interface, a Python programming interface, a KDE KIO slave module, and a Ubuntu Unity Lens module.

Chapter 2

Indexing

2.1 Introduction

Indexing is the process by which the set of documents is analyzed and the data entered into the database. Recoll indexing is normally incremental: documents will only be processed if they have been modified. On the first execution, all documents will need processing. A full index build can be forced later by specifying an option to the indexing command (**recollindex** `-z` or `-Z`).

The following sections give an overview of different aspects of the indexing processes and configuration, with links to detailed sections.

2.1.1 Indexing modes

Recoll indexing can be performed along two different modes:

- **Periodic (or batch) indexing:** indexing takes place at discrete times, by executing the **recollindex** command. The typical usage is to have a nightly indexing run programmed into your **cron** file.
- **Real time indexing:** indexing takes place as soon as a file is created or changed. **recollindex** runs as a daemon and uses a file system alteration monitor such as inotify, Fam or Gamin to detect file changes.

The choice between the two methods is mostly a matter of preference, and they can be combined by setting up multiple indexes (ie: use periodic indexing on a big documentation directory, and real time indexing on a small home directory). Monitoring a big file system tree can consume significant system resources.

The choice of method and the parameters used can be configured from the **recoll** GUI: Preferences → Indexing schedule

2.1.2 Configurations, multiple indexes

The parameters describing what is to be indexed and local preferences are defined in text files contained in a configuration directory.

All parameters have defaults, defined in system-wide files.

Without further configuration, Recoll will index all appropriate files from your home directory, with a reasonable set of defaults.

A default personal configuration directory (`$HOME/.recoll/`) is created when a Recoll program is first executed. It is possible to create other configuration directories, and use them by setting the `RECOLL_CONFDIR` environment variable, or giving the `-c` option to any of the Recoll commands.

In some cases, it may be interesting to index different areas of the file system to separate databases. You can do this by using multiple configuration directories, each indexing a file system area to a specific database. Typically, this would be done to separate personal and shared indexes, or to take advantage of the organization of your data to improve search precision.

The generated indexes can be queried concurrently in a transparent manner.

For index generation, multiple configurations are totally independant from each other. When multiple indexes need to be used for a single search, some parameters should be consistent among the configurations.

2.1.3 Document types

Recoll knows about quite a few different document types. The parameters for document types recognition and processing are set in configuration files.

Most file types, like HTML or word processing files, only hold one document. Some file types, like email folders or zip archives, can hold many individually indexed documents, which may themselves be compound ones. Such hierarchies can go quite deep, and Recoll can process, for example, a LibreOffice document stored as an attachment to an email message inside an email folder archived in a zip file...

Recoll indexing processes plain text, HTML, OpenDocument (Open/LibreOffice), email formats, and a few others internally.

Other file types (ie: postscript, pdf, ms-word, rtf ...) need external applications for preprocessing. The list is in the installation section. After every indexing operation, Recoll updates a list of commands that would be needed for indexing existing files types. This list can be displayed by selecting the menu option File → Show Missing Helpers in the **recoll** GUI. It is stored in the `missing` text file inside the configuration directory.

By default, Recoll will try to index any file type that it has a way to read. This is sometimes not desirable, and there are ways to either exclude some types, or on the contrary to define a positive list of types to be indexed. In the latter case, any type not in the list will be ignored.

Excluding types can be done by adding name patterns to the `skippedNames` list, which can be done from the GUI Index configuration menu. It is also possible to exclude a mime type independantly of the file name by associating it with the `rclnull` filter. This can be done by editing the mimeconf configuration file.

In order to define a positive list, You need to edit the main configuration file (recoll.conf) and set the `indexedmimetypes` configuration variable. Example:

```
indexedmimetypes = text/html application/pdf
```

It is possible to redefine this parameter for subdirectories. Example:

```
[/path/to/my/dir]
indexedmimetypes = application/pdf
```

(When using sections like this, don't forget that they remain in effect until the end of the file or another section indicator). There is no GUI way to edit the parameter, because this option runs contrary to Recoll main goal which is to help you find information, independantly of how it may be stored.

2.1.4 Recovery

In the rare case where the index becomes corrupted (which can signal itself by weird search results or crashes), the index files need to be erased before restarting a clean indexing pass. Just delete the `xapiandb` directory (see next section), or, alternatively, start the next **recollindex** with the `-z` option, which will reset the database before indexing.

2.2 Index storage

The default location for the index data is the `xapiandb` subdirectory of the Recoll configuration directory, typically `$HOME/.recoll/xapiandb/`. This can be changed via two different methods (with different purposes):

- You can specify a different configuration directory by setting the `RECOLL_CONFDIR` environment variable, or using the `-c` option to the Recoll commands. This method would typically be used to index different areas of the file system to different indexes. For example, if you were to issue the following commands:

```
export RECOLL_CONFDIR=~/.indexes-email
recoll
```

Then Recoll would use configuration files stored in `~/.indexes-email/` and, (unless specified otherwise in `recoll.conf`) would look for the index in `~/.indexes-email/xapiandb/`.

Using multiple configuration directories and configuration options allows you to tailor multiple configurations and indexes to handle whatever subset of the available data you wish to make searchable.

- For a given configuration directory, you can specify a non-default storage location for the index by setting the `dbdir` parameter in the configuration file (see the configuration section). This method would mainly be of use if you wanted to keep the configuration directory in its default location, but desired another location for the index, typically out of disk occupation concerns.

The size of the index is determined by the size of the set of documents, but the ratio can vary a lot. For a typical mixed set of documents, the index size will often be close to the data set size. In specific cases (a set of compressed mbox files for example), the index can become much bigger than the documents. It may also be much smaller if the documents contain a lot of images or other non-indexed data (an extreme example being a set of mp3 files where only the tags would be indexed).

Of course, images, sound and video do not increase the index size, which means that nowadays (2012), typically, even a big index will be negligible against the total amount of data on the computer.

The index data directory (`xapiandb`) only contains data that can be completely rebuilt by an index run (as long as the original documents exist), and it can always be destroyed safely.

2.2.1 Xapian index formats

Xapian versions usually support several formats for index storage. A given major Xapian version will have a current format, used to create new indexes, and will also support the format from the previous major version.

Xapian will not convert automatically an existing index from the older format to the newer one. If you want to upgrade to the new format, or if a very old index needs to be converted because its format is not supported any more, you will have to explicitly delete the old index, then run a normal indexing process.

Using the `-z` option to **recollindex** is not sufficient to change the format, you will have to delete all files inside the index directory (typically `~/.recoll/xapiandb`) before starting the indexing.

2.2.2 Security aspects

The Recoll index does not hold copies of the indexed documents. But it does hold enough data to allow for an almost complete reconstruction. If confidential data is indexed, access to the database directory should be restricted.

Recoll (since version 1.4) will create the configuration directory with a mode of 0700 (access by owner only). As the index data directory is by default a sub-directory of the configuration directory, this should result in appropriate protection.

If you use another setup, you should think of the kind of protection you need for your index, set the directory and files access modes appropriately, and also maybe adjust the `umask` used during index updates.

2.3 Index configuration

Variables set inside the Recoll configuration files control which areas of the file system are indexed, and how files are processed. These variables can be set either by editing the text files or by using the dialogs in the recoll GUI.

The first time you start **recoll**, you will be asked whether or not you would like it to build the index. If you want to adjust the configuration before indexing, just click Cancel at this point, which will get you into the configuration interface. If you exit at this point, `recoll` will have created a `~/.recoll` directory containing empty configuration files, which you can edit by hand.

The configuration is documented inside the installation chapter of this document, or in the recoll.conf(5) man page, but the most current information will most likely be the comments inside the sample file. The most immediately useful variable you may interested in is probably topdirs, which determines what subtrees get indexed.

The applications needed to index file types other than text, HTML or email (ie: pdf, postscript, ms-word...) are described in the external packages section.

As of Recoll 1.18 there are two incompatible types of Recoll indexes, depending on the treatment of character case and diacritics. The next section describes the two types in more detail.

2.3.1 Multiple indexes

Multiple Recoll indexes can be created by using several configuration directories which are usually set to index different areas of the file system. A specific index can be selected for updating or searching, using the `RECOLL_CONFDIR` environment variable or the `-c` option to **recoll** and **recollindex**.

A typical usage scenario for the multiple index feature would be for a system administrator to set up a central index for shared data, that you choose to search or not in addition to your personal data. Of course, there are other possibilities. There are many cases where you know the subset of files that should be searched, and where narrowing the search can improve the results. You can achieve approximately the same effect with the directory filter in advanced search, but multiple indexes will have much better performance and may be worth the trouble.

A **recollindex** program instance can only update one specific index.

The main index (defined by `RECOLL_CONFDIR` or `-c`) is always active. If this is undesirable, you can set up your base configuration to index an empty directory.

The different search interfaces (GUI, command line, ...) have different methods to define the set of indexes to be used, see the appropriate section.

If a set of multiple indexes are to be used together for searches, some configuration parameters must be consistent among the set. These are parameters which need to be the same when indexing and searching. As the parameters come from the main configuration when searching, they need to be compatible with what was set when creating the other indexes (which came from their respective configuration directories).

Most importantly, all indexes to be queried concurrently must have the same option concerning character case and diacritics stripping, but there are other constraints. Most of the relevant parameters are described in the linked section.

2.3.2 Index case and diacritics sensitivity

As of Recoll version 1.18 you have a choice of building an index with terms stripped of character case and diacritics, or one with raw terms. For a source term of `Résumé`, the former will store `resume`, the latter `Résumé`.

Each type of index allows performing searches insensitive to case and diacritics: with a raw index, the user entry will be expanded to match all case and diacritics variations present in the index. With a stripped index, the search term will be stripped before searching.

A raw index allows for another possibility which a stripped index cannot offer: using case and diacritics to discriminate between terms, returning different results when searching for `US` and `us` or `resume` and `résumé`. Read the section about search case and diacritics sensitivity for more details.

The type of index to be created is controlled by the `indexStripChars` configuration variable which can only be changed by editing the configuration file. Any change implies an index reset (not automated by Recoll), and all indexes in a search must be set in the same way (again, not checked by Recoll).

If the `indexStripChars` is not set, Recoll 1.18 creates a stripped index by default, for compatibility with previous versions.

As a cost for added capability, a raw index will be slightly bigger than a stripped one (around 10%). Also, searches will be more complex, so probably slightly slower, and the feature is still young, so that a certain amount of weirdness cannot be excluded.

One of the most adverse consequence of using a raw index is that some phrase and proximity searches may become impossible: because each term needs to be expanded, and all combinations searched for, the multiplicative expansion may become unmanageable.

2.3.3 The index configuration GUI

Most parameters for a given index configuration can be set from a **recoll** GUI running on this configuration (either as default, or by setting `RECOLL_CONFDIR` or the `-c` option.)

The interface is started from the Preferences → Index Configuration menu entry. It is divided in four tabs, Global parameters, Local parameters, Web history (which is explained in the next section) and Search parameters.

The Global parameters tab allows setting global variables, like the lists of top directories, skipped paths, or stemming languages.

The Local parameters tab allows setting variables that can be redefined for subdirectories. This second tab has an initially empty list of customisation directories, to which you can add. The variables are then set for the currently selected directory (or at the top level if the empty line is selected).

The Search parameters section defines parameters which are used at query time, but are global to an index and affect all search tools, not only the GUI.

The meaning for most entries in the interface is self-evident and documented by a `ToolTip` popup on the text label. For more detail, you will need to refer to the configuration section of this guide.

The configuration tool normally respects the comments and most of the formatting inside the configuration file, so that it is quite possible to use it on hand-edited files, which you might nevertheless want to backup first...

2.4 Indexing WEB pages you wisit

With the help of a Firefox extension, Recoll can index the Internet pages that you visit. The extension was initially designed for the Beagle indexer, but it has recently be renamed and better adapted to Recoll.

The extension works by copying visited WEB pages to an indexing queue directory, which Recoll then processes, indexing the data, storing it into a local cache, then removing the file from the queue.

This feature can be enabled in the GUI Index configuration panel, or by editing the configuration file (set `processwebqueue` to 1).

A current pointer to the extension can be found, along with up-to-date instructions, on the Recoll wiki.

A copy of the indexed WEB pages is retained by Recoll in a local cache (from which previews can be fetched). The cache size can be adjusted from the Index configuration / Web history panel. Once the maximum size is reached, old pages are purged - both from the cache and the index - to make room for new ones, so you need to explicitly archive in some other place the pages that you want to keep indefinitely.

2.5 Extended attributes data

User extended attributes are named pieces of information that most modern file systems can attach to any file.

Recoll versions 1.19 and later process extended attributes as document fields by default. For older versions, this has to be activated at build time.

A freedesktop standard defines a few special attributes, which are handled as such by Recoll:

mime_type If set, this overrides any other determination of the file mime type.

charset If set, this defines the file character set (mostly useful for plain text files).

By default, other attributes are handled as Recoll fields. On Linux, the `user` prefix is removed from the name. This can be configured more precisely inside the fields configuration file.

2.6 Importing external tags

During indexing, it is possible to import metadata for each file by executing commands. For example, this could extract user tag data for the file and store it in a field for indexing.

See the section about the metadatacmds field in the main configuration chapter for more detail.

2.7 Periodic indexing

2.7.1 Running indexing

Indexing is always performed by the **recollindex** program, which can be started either from the command line or from the File menu in the **recoll** GUI program. When started from the GUI, the indexing will run on the same configuration **recoll** was started on. When started from the command line, **recollindex** will use the `RECOLL_CONFDIR` variable or accept a `-c` *`confdir`* option to specify a non-default configuration directory.

If the **recoll** program finds no index when it starts, it will automatically start indexing (except if canceled).

The **recollindex** indexing process can be interrupted by sending an interrupt (Ctrl-C, SIGINT) or terminate (SIGTERM) signal. Some time may elapse before the process exits, because it needs to properly flush and close the index. This can also be done from the **recoll** GUI File → Stop Indexing menu entry.

After such an interruption, the index will be somewhat inconsistent because some operations which are normally performed at the end of the indexing pass will have been skipped (for example, the stemming and spelling databases will be inexistant or out of date). You just need to restart indexing at a later time to restore consistency. The indexing will restart at the interruption point (the full file tree will be traversed, but files that were indexed up to the interruption and for which the index is still up to date will not need to be reindexed).

recollindex has a number of other options which are described in its man page. Only a few will be described here.

Option `-z` will reset the index when starting. This is almost the same as destroying the index files (the nuance is that the Xapian format version will not be changed).

Option `-Z` will force the update of all documents without resetting the index first. This will not have the "clean start" aspect of `-z`, but the advantage is that the index will remain available for querying while it is rebuilt, which can be a significant advantage if it is very big (some installations need days for a full index rebuild).

Of special interest also, maybe, are the `-i` and `-f` options. `-i` allows indexing an explicit list of files (given as command line parameters or read on `stdin`). `-f` tells **recollindex** to ignore file selection parameters from the configuration. Together, these options allow building a custom file selection process for some area of the file system, by adding the top directory to the `skippedPaths` list and using an appropriate file selection method to build the file list to be fed to **recollindex** `-if`. Trivial example:

```
find . -name indexable.txt -print | recollindex -if
```

recollindex `-i` will not descend into subdirectories specified as parameters, but just add them as index entries. It is up to the external file selection method to build the complete file list.

2.7.2 Using cron to automate indexing

The most common way to set up indexing is to have a cron task execute it every night. For example the following `crontab` entry would do it every day at 3:30AM (supposing **recollindex** is in your PATH):

```
30 3 * * * recollindex > /some/tmp/dir/recolltrace 2>&1
```

Or, using **anacron**:

```
1  15  su mylogin -c "recollindex recollindex > /tmp/rcltraceme 2>&1"
```

As of version 1.17 the Recoll GUI has dialogs to manage `crontab` entries for **recollindex**. You can reach them from the Preferences → Indexing Schedule menu. They only work with the good old **cron**, and do not give access to all features of **cron** scheduling.

The usual command to edit your `crontab` is **crontab** `-e` (which will usually start the **vi** editor to edit the file). You may have more sophisticated tools available on your system.

Please be aware that there may be differences between your usual interactive command line environment and the one seen by crontab commands. Especially the PATH variable may be of concern. Please check the crontab manual pages about possible issues.

2.8 Real time indexing

Real time monitoring/indexing is performed by starting the **recollindex** `-m` command. With this option, **recollindex** will detach from the terminal and become a daemon, permanently monitoring file changes and updating the index.

Under KDE, Gnome and some other desktop environments, the daemon can automatically started when you log in, by creating a desktop file inside the `~/.config/autostart` directory. This can be done for you by the Recoll GUI. Use the Preferences->Indexing Schedule menu.

With older X11 setups, starting the daemon is normally performed as part of the user session script.

The `rclmon.sh` script can be used to easily start and stop the daemon. It can be found in the `examples` directory (typically `/usr/local/[share/]recoll/examples`).

For example, my out of fashion xdm-based session has a `.xsession` script with the following lines at the end:

```
recollconf=$HOME/.recoll-home
recolldata=/usr/local/share/recoll
RECOLL_CONFDIR=$recollconf $recolldata/examples/rclmon.sh start

fvwm
```

The indexing daemon gets started, then the window manager, for which the session waits.

By default the indexing daemon will monitor the state of the X11 session, and exit when it finishes, it is not necessary to kill it explicitly. (The X11 server monitoring can be disabled with option `-x` to **recollindex**).

If you use the daemon completely out of an X11 session, you need to add option `-x` to disable X11 session monitoring (else the daemon will not start).

By default, the messages from the indexing daemon will be discarded. You may want to change this by setting the `daemlogfilename` and `daemloglevel` configuration parameters. Also the log file will only be truncated when the daemon starts. If the daemon runs permanently, the log file may grow quite big, depending on the log level.

When building Recoll, the real time indexing support can be customised during package configuration with the `--with[out]-fam` or `--with[out]-inotify` options. The default is currently to include inotify monitoring on systems that support it, and, as of Recoll 1.17, gamin support on FreeBSD.

While it is convenient that data is indexed in real time, repeated indexing can generate a significant load on the system when files such as email folders change. Also, monitoring large file trees by itself significantly taxes system resources. You probably do not want to enable it if your system is short on resources. Periodic indexing is adequate in most cases.

2.8.1 Slowing down the reindexing rate for fast changing files

When using the real time monitor, it may happen that some files need to be indexed, but change so often that they impose an excessive load for the system.

Recoll provides a configuration option to specify the minimum time before which a file, specified by a wildcard pattern, cannot be reindexed. See the `mondelaypatterns` parameter in the configuration section.

Chapter 3

Searching

3.1 Searching with the Qt graphical user interface

The **recoll** program provides the main user interface for searching. It is based on the Qt library.

recoll has two search modes:

- Simple search (the default, on the main screen) has a single entry field where you can enter multiple words.
- Advanced search (a panel accessed through the Tools menu or the toolbox bar icon) has multiple entry fields, which you may use to build a logical condition, with additional filtering on file type, location in the file system, modification date, and size.

In most cases, you can enter the terms as you think them, even if they contain embedded punctuation or other non-textual characters. For example, Recoll can handle things like email addresses, or arbitrary cut and paste from another text window, punctation and all.

The main case where you should enter text differently from how it is printed is for east-asian languages (Chinese, Japanese, Korean). Words composed of single or multiple characters should be entered separated by white space in this case (they would typically be printed without white space).

3.1.1 Simple search

1. Start the **recoll** program.
2. Possibly choose a search mode: Any term, All terms, File name or Query language.
3. Enter search term(s) in the text field at the top of the window.
4. Click the Search button or hit the **Enter** key to start the search.

The initial default search mode is Query language. Without special directives, this will look for documents containing all of the search terms (the ones with more terms will get better scores), just like the All terms mode which will ignore such directives. Any term will search for documents where at least one of the terms appear.

The Query Language features are described in a separate section.

All search modes allow wildcards inside terms (`*`, `?`, `[]`). You may want to have a look at the section about wildcards for more information about this.

File name will specifically look for file names. The point of having a separate file name search is that wild card expansion can be performed more efficiently on a small subset of the index (allowing wild cards on the left of terms without excessive penality). Things to know:

- White space in the entry should match white space in the file name, and is not treated specially.
- The search is insensitive to character case and accents, independantly of the type of index.
- An entry without any wild card character and not capitalized will be prepended and appended with '*' (ie: *etc* -> **etc**, but *Etc* -> *etc*).
- If you have a big index (many files), excessively generic fragments may result in inefficient searches.

You can search for exact phrases (adjacent words in a given order) by enclosing the input inside double quotes. Ex: "virtual reality".

When using a stripped index, character case has no influence on search, except that you can disable stem expansion for any term by capitalizing it. Ie: a search for floor will also normally look for flooring, floored, etc., but a search for Floor will only look for floor, in any character case. Stemming can also be disabled globally in the preferences. When using a raw index, the rules are a bit more complicated.

Recoll remembers the last few searches that you performed. You can use the simple search text entry widget (a combobox) to recall them (click on the thing at the right of the text field). Please note, however, that only the search texts are remembered, not the mode (all/any/file name).

Typing **Esc Space** while entering a word in the simple search entry will open a window with possible completions for the word. The completions are extracted from the database.

Double-clicking on a word in the result list or a preview window will insert it into the simple search entry field.

You can cut and paste any text into an All terms or Any term search field, punctuation, newlines and all - except for wildcard characters (single ? characters are ok). Recoll will process it and produce a meaningful search. This is what most differentiates this mode from the Query Language mode, where you have to care about the syntax.

You can use the ToolsAdvanced search dialog for more complex searches.

3.1.2 The default result list

After starting a search, a list of results will instantly be displayed in the main list window.

By default, the document list is presented in order of relevance (how well the system estimates that the document matches the query). You can sort the result by ascending or descending date by using the vertical arrows in the toolbar.

Clicking on the Preview link for an entry will open an internal preview window for the document. Further Preview clicks for the same search will open tabs in the existing preview window. You can use **Shift**+Click to force the creation of another preview window, which may be useful to view the documents side by side. (You can also browse successive results in a single preview window by typing **Shift+ArrowUp/Down** in the window).

Clicking the Open link will start an external viewer for the document. By default, Recoll lets the desktop choose the appropriate application for most document types (there is a short list of exceptions, see further). If you prefer to completely customize the choice of applications, you can uncheck the Use desktop preferences option in the GUI preferences dialog, and click the Choose editor applications button to adjust the predefined Recoll choices. The tool accepts multiple selections of mime types (e.g. to set up the editor for the dozens of office file types).

Even when Use desktop preferences is checked, there is a small list of exceptions, for mime types where the Recoll choice should override the desktop one. These are applications which are well integrated with Recoll, especially evince for viewing PDF and Postscript files because of its support for opening the document at a specific page and passing a search string as an argument. Of course, you can edit the list (in the GUI preferences) if you would prefer to lose the functionality and use the standard desktop tool.

You may also change the choice of applications by editing the mimeview configuration file if you find this more convenient.

The Preview and Open edit links may not be present for all entries, meaning that Recoll has no configured way to preview a given file type (which was indexed by name only), or no configured external editor for the file type. This can sometimes be adjusted simply by tweaking the mimemap and mimeview configuration files (the latter can be modified with the user preferences dialog).

The format of the result list entries is entirely configurable by using the preference dialog to edit an HTML fragment.

You can click on the `Query details` link at the top of the results page to see the query actually performed, after stem expansion and other processing.

Double-clicking on any word inside the result list or a preview window will insert it into the simple search text.

The result list is divided into pages (the size of which you can change in the preferences). Use the arrow buttons in the toolbar or the links at the bottom of the page to browse the results.

3.1.2.1 No results: the spelling suggestions

When a search yields no result, and if the aspell dictionary is configured, Recoll will try to check for misspellings among the query terms, and will propose lists of replacements. Clicking on one of the suggestions will replace the word and restart the search. You can hold any of the modifier keys (Ctrl, Shift, etc.) while clicking if you would rather stay on the suggestion screen because several terms need replacement.

3.1.2.2 The result list right-click menu

Apart from the preview and edit links, you can display a pop-up menu by right-clicking over a paragraph in the result list. This menu has the following entries:

- Preview
- Open
- Copy File Name
- Copy Url
- Save to File
- Find similar
- Preview Parent document
- Open Parent document
- Open Snippets Window

The Preview and Open entries do the same thing as the corresponding links.

The Copy File Name and Copy Url copy the relevant data to the clipboard, for later pasting.

Save to File allows saving the contents of a result document to a chosen file. This entry will only appear if the document does not correspond to an existing file, but is a subdocument inside such a file (ie: an email attachment). It is especially useful to extract attachments with no associated editor.

The Find similar entry will select a number of relevant term from the current document and enter them into the simple search field. You can then start a simple search, with a good chance of finding documents related to the current result.

The Parent document entries will appear for documents which are not actually files but are part of, or attached to, a higher level document. This entry is mainly useful for email attachments and permits viewing the message to which the document is attached. Note that the entry will also appear for an email which is part of an mbox folder file, but that you can't actually visualize the folder (there will be an error dialog if you try). Recoll is unfortunately not yet smart enough to disable the entry in this case. In other cases, the Open option makes sense, for example to start a chm viewer on the parent document for a help page.

The Open Snippets Window entry will only appear for documents which support page breaks (typically PDF, Postscript, DVI). The snippets window lists extracts from the document, taken around search terms occurrences, along with the corresponding page number, as links which can be used to start the native viewer on the appropriate page. If the viewer supports it, its search function will also be primed with one of the search terms.

3.1.3 The result table

In Recoll 1.15 and newer, the results can be displayed in spreadsheet-like fashion. You can switch to this presentation by clicking the table-like icon in the toolbar (this is a toggle, click again to restore the list).

Clicking on the column headers will allow sorting by the values in the column. You can click again to invert the order, and use the header right-click menu to reset sorting to the default relevance order (you can also use the sort-by-date arrows to do this).

Both the list and the table display the same underlying results. The sort order set from the table is still active if you switch back to the list mode. You can click twice on a date sort arrow to reset it from there.

The header right-click menu allows adding or deleting columns. The columns can be resized, and their order can be changed (by dragging). All the changes are recorded when you quit **recoll**

Hovering over a table row will update the detail area at the bottom of the window with the corresponding values. You can click the row to freeze the display. The bottom area is equivalent to a result list paragraph, with links for starting a preview or a native application, and an equivalent right-click menu. Typing **Esc** (the Escape key) will unfreeze the display.

3.1.4 Displaying thumbnails

The default format for the result list entries and the detail area of the result table display an icon for each result document. The icon is either a generic one determined from the MIME type, or a thumbnail of the document appearance. Thumbnails are only displayed if found in the standard freedesktop location, where they would typically have been created by a file manager.

Recoll has no capability to create thumbnails. A relatively simple trick is to use the Open parent document/folder entry in the result list popup menu. This should open a file manager window on the containing directory, which should in turn create the thumbnails (depending on your settings). Restarting the search should then display the thumbnails.

There are also some pointers about thumbnail generation on the Recoll wiki.

3.1.5 The preview window

The preview window opens when you first click a `Preview` link inside the result list.

Subsequent preview requests for a given search open new tabs in the existing window (except if you hold the **Shift** key while clicking which will open a new window for side by side viewing).

Starting another search and requesting a preview will create a new preview window. The old one stays open until you close it.

You can close a preview tab by typing **Ctrl-W** (**Ctrl** + **W**) in the window. Closing the last tab for a window will also close the window.

Of course you can also close a preview window by using the window manager button in the top of the frame.

You can display successive or previous documents from the result list inside a preview tab by typing **Shift+Down** or **Shift+Up** (**Down** and **Up** are the arrow keys).

A right-click menu in the text area allows switching between displaying the main text or the contents of fields associated to the document (ie: author, abtract, etc.). This is especially useful in cases where the term match did not occur in the main text but in one of the fields. In the case of images, you can switch between three displays: the image itself, the image metadata as extracted by **exiftool** and the fields, which is the metadata stored in the index.

You can print the current preview window contents by typing **Ctrl-P** (**Ctrl** + **P**) in the window text.

3.1.5.1 Searching inside the preview

The preview window has an internal search capability, mostly controlled by the panel at the bottom of the window, which works in two modes: as a classical editor incremental search, where we look for the text entered in the entry zone, or as a way to walk the matches between the document and the Recoll query that found it.

Incremental text search The preview tabs have an internal incremental search function. You initiate the search either by typing a **/** (slash) or **CTL-F** inside the text area or by clicking into the Search for: text field and entering the search string. You can then use the Next and Previous buttons to find the next/previous occurrence. You can also type **F3** inside the text area to get to the next occurrence.

If you have a search string entered and you use Ctrl-Up/Ctrl-Down to browse the results, the search is initiated for each successive document. If the string is found, the cursor will be positioned at the first occurrence of the search string.

Walking the match lists If the entry area is empty when you click the Next or Previous buttons, the editor will be scrolled to show the next match to any search term (the next highlighted zone). If you select a search group from the dropdown list and click Next or Previous, the match list for this group will be walked. This is not the same as a text search, because the occurences will include non-exact matches (as caused by stemming or wildcards). The search will revert to the text mode as soon as you edit the entry area.

3.1.6 Complex/advanced search

The advanced search dialog helps you build more complex queries without memorizing the search language constructs. It can be opened through the Tools menu or through the main toolbar.

The dialog has two tabs:

1. The first tab lets you specify terms to search for, and permits specifying multiple clauses which are combined to build the search.
2. The second tab lets filter the results according to file size, date of modification, mime type, or location.

Click on the Start Search button in the advanced search dialog, or type **Enter** in any text field to start the search. The button in the main window always performs a simple search.

Click on the `Show query details` link at the top of the result page to see the query expansion.

3.1.6.1 Avanced search: the "find" tab

This part of the dialog lets you constructc a query by combining multiple clauses of different types. Each entry field is configurable for the following modes:

- All terms.
- Any term.
- None of the terms.
- Phrase (exact terms in order within an adjustable window).
- Proximity (terms in any order within an adjustable window).
- Filename search.

Additional entry fields can be created by clicking the Add clause button.

When searching, the non-empty clauses will be combined either with an AND or an OR conjunction, depending on the choice made on the left (All clauses or Any clause).

Entries of all types except "Phrase" and "Near" accept a mix of single words and phrases enclosed in double quotes. Stemming and wildcard expansion will be performed as for simple search.

Phrases and Proximity searches These two clauses work in similar ways, with the difference that proximity searches do not impose an order on the words. In both cases, an adjustable number (slack) of non-matched words may be accepted between the searched ones (use the counter on the left to adjust this count). For phrases, the default count is zero (exact match). For proximity it is ten (meaning that two search terms, would be matched if found within a window of twelve words). Examples: a phrase search for `quick fox` with a slack of 0 will match `quick fox` but not `quick brown fox`. With a slack of 1 it will match the latter, but not `fox quick`. A proximity search for `quick fox` with the default slack will match the latter, and also `a fox is a cunning and quick animal`.

3.1.6.2 Avanced search: the "filter" tab

This part of the dialog has several sections which allow filtering the results of a search according to a number of criteria

- The first section allows filtering by dates of last modification. You can specify both a minimum and a maximum date. The initial values are set according to the oldest and newest documents found in the index.
- The next section allows filtering the results by file size. There are two entries for minimum and maximum size. Enter decimal numbers. You can use suffix multipliers: `k/K`, `m/M`, `g/G`, `t/T` for 1E3, 1E6, 1E9, 1E12 respectively.
- The next section allows filtering the results by their mime types, or mime categories (ie: media/text/message/etc.).

  You can transfer the types between two boxes, to define which will be included or excluded by the search.

  The state of the file type selection can be saved as the default (the file type filter will not be activated at program start-up, but the lists will be in the restored state).
- The bottom section allows restricting the search results to a sub-tree of the indexed area. You can use the Invert checkbox to search for files not in the sub-tree instead. If you use directory filtering often and on big subsets of the file system, you may think of setting up multiple indexes instead, as the performance may be better.

  You can use relative/partial paths for filtering. Ie, entering `dirA/dirB` would match either `/dir1/dirA/dirB/myfile1` or `/dir2/dirA/dirB/someother/myfile2`.

3.1.6.3 Avanced search history

The advanced search tool memorizes the last 100 searches performed. You can walk the saved searches by using the up and down arrow keys while the keyboard focus belongs to the advanced search dialog.

The complex search history can be erased, along with the one for simple search, by selecting the File → Erase Search History menu entry.

3.1.7 The term explorer tool

Recoll automatically manages the expansion of search terms to their derivatives (ie: plural/singular, verb inflections). But there are other cases where the exact search term is not known. For example, you may not remember the exact spelling, or only know the beginning of the name.

The term explorer tool (started from the toolbar icon or from the Term explorer entry of the Tools menu) can be used to search the full index terms list. It has three modes of operations:

Wildcard In this mode of operation, you can enter a search string with shell-like wildcards (*, ?, []). ie: *xapi** would display all index terms beginning with *xapi*. (More about wildcards here).

Regular expression This mode will accept a regular expression as input. Example: *word[0-9]+*. The expression is implicitely anchored at the beginning. Ie: *press* will match *pression* but not *expression*. You can use *.*press* to match the latter, but be aware that this will cause a full index term list scan, which can be quite long.

Stem expansion This mode will perform the usual stem expansion normally done as part user input processing. As such it is probably mostly useful to demonstrate the process.

Spelling/Phonetic In this mode, you enter the term as you think it is spelled, and Recoll will do its best to find index terms that sound like your entry. This mode uses the Aspell spelling application, which must be installed on your system for things to work (if your documents contain non-ascii characters, Recoll needs an aspell version newer than 0.60 for UTF-8 support). The language which is used to build the dictionary out of the index terms (which is done at the end of an indexing pass) is the one defined by your NLS environment. Weird things will probably happen if languages are mixed up.

Note that in cases where Recoll does not know the beginning of the string to search for (ie a wildcard expression like **coll*), the expansion can take quite a long time because the full index term list will have to be processed. The expansion is currently limited at 10000 results for wildcards and regular expressions. It is possible to change the limit in the configuration file.

Double-clicking on a term in the result list will insert it into the simple search entry field. You can also cut/paste between the result list and any entry field (the end of lines will be taken care of).

3.1.8 Multiple indexes

See the section describing the use of multiple indexes for generalities. Only the aspects concerning the **recoll** GUI are described here.

A **recoll** program instance is always associated with a specific index, which is the one to be updated when requested from the File menu, but it can use any number of Recoll indexes for searching. The external indexes can be selected through the external indexes tab in the preferences dialog.

Index selection is performed in two phases. A set of all usable indexes must first be defined, and then the subset of indexes to be used for searching. These parameters are retained across program executions (there are kept separately for each Recoll configuration). The set of all indexes is usually quite stable, while the active ones might typically be adjusted quite frequently.

The main index (defined by `RECOLL_CONFDIR`) is always active. If this is undesirable, you can set up your base configuration to index an empty directory.

When adding a new index to the set, you can select either a Recoll configuration directory, or directly a Xapian index directory. In the first case, the Xapian index directory will be obtained from the selected configuration.

As building the set of all indexes can be a little tedious when done through the user interface, you can use the `RECOLL_EXTRA_DBS` environment variable to provide an initial set. This might typically be set up by a system administrator so that every user does not have to do it. The variable should define a colon-separated list of index directories, ie:

```
export RECOLL_EXTRA_DBS=/some/place/xapiandb:/some/other/db
```

Another environment variable, `RECOLL_ACTIVE_EXTRA_DBS` allows adding to the active list of indexes. This variable was suggested and implemented by a Recoll user. It is mostly useful if you use scripts to mount external volumes with Recoll indexes. By using `RECOLL_EXTRA_DBS` and `RECOLL_ACTIVE_EXTRA_DBS`, you can add and activate the index for the mounted volume when starting **recoll**.

`RECOLL_ACTIVE_EXTRA_DBS` is available for Recoll versions 1.17.2 and later. A change was made in the same update so that **recoll** will automatically deactivate unreachable indexes when starting up.

3.1.9 Document history

Documents that you actually view (with the internal preview or an external tool) are entered into the document history, which is remembered.

You can display the history list by using the Tools/Doc History menu entry.

You can erase the document history by using the Erase document history entry in the File menu.

3.1.10 Sorting search results and collapsing duplicates

The documents in a result list are normally sorted in order of relevance. It is possible to specify a different sort order, either by using the vertical arrows in the GUI toolbox to sort by date, or switching to the result table display and clicking on any header. The sort order chosen inside the result table remains active if you switch back to the result list, until you click one of the vertical arrows, until both are unchecked (you are back to sort by relevance).

Sort parameters are remembered between program invocations, but result sorting is normally always inactive when the program starts. It is possible to keep the sorting activation state between program invocations by checking the Remember sort activation state option in the preferences.

It is also possible to hide duplicate entries inside the result list (documents with the exact same contents as the displayed one). The test of identity is based on an MD5 hash of the document container, not only of the text contents (so that ie, a text document with an image added will not be a duplicate of the text only). Duplicates hiding is controlled by an entry in the GUI configuration dialog, and is off by default.

As of release 1.19, when a result document does have undisplayed duplicates, a `Dups` link will be shown with the result list entry. Clicking the link will display the paths (URLs + ipaths) for the duplicate entries.

3.1.11 Search tips, shortcuts

3.1.11.1 Terms and search expansion

Term completion Typing **Esc Space** in the simple search entry field while entering a word will either complete the current word if its beginning matches a unique term in the index, or open a window to propose a list of completions.

Picking up new terms from result or preview text Double-clicking on a word in the result list or in a preview window will copy it to the simple search entry field.

Wildcards Wildcards can be used inside search terms in all forms of searches. More about wildcards.

Automatic suffixes Words like `odt` or `ods` can be automatically turned into query language `ext:xxx` clauses. This can be enabled in the Search preferences panel in the GUI.

Disabling stem expansion Entering a capitalized word in any search field will prevent stem expansion (no search for `garden ing` if you enter `Garden` instead of `garden`). This is the only case where character case should make a difference for a Recoll search. You can also disable stem expansion or change the stemming language in the preferences.

Finding related documents Selecting the Find similar documents entry in the result list paragraph right-click menu will select a set of "interesting" terms from the current result, and insert them into the simple search entry field. You can then possibly edit the list and start a search to find documents which may be apparented to the current result.

File names File names are added as terms during indexing, and you can specify them as ordinary terms in normal search fields (Recoll used to index all directories in the file path as terms. This has been abandoned as it did not seem really useful). Alternatively, you can use the specific file name search which will *only* look for file names, and may be faster than the generic search especially when using wildcards.

3.1.11.2 Working with phrases and proximity

Phrases and Proximity searches A phrase can be looked for by enclosing it in double quotes. Example: `"user manual"` will look only for occurrences of `user` immediately followed by `manual`. You can use the This phrase field of the advanced search dialog to the same effect. Phrases can be entered along simple terms in all simple or advanced search entry fields (except This exact phrase).

AutoPhrases This option can be set in the preferences dialog. If it is set, a phrase will be automatically built and added to simple searches when looking for `Any terms`. This will not change radically the results, but will give a relevance boost to the results where the search terms appear as a phrase. Ie: searching for `virtual reality` will still find all documents where either `virtual` or `reality` or both appear, but those which contain `virtual reality` should appear sooner in the list.

Phrase searches can strongly slow down a query if most of the terms in the phrase are common. This is why the `autophrase` option is off by default for Recoll versions before 1.17. As of version 1.17, `autophrase` is on by default, but very common terms will be removed from the constructed phrase. The removal threshold can be adjusted from the search preferences.

Phrases and abbreviations As of Recoll version 1.17, dotted abbreviations like `I.B.M.` are also automatically indexed as a word without the dots: `IBM`. Searching for the word inside a phrase (ie: `"the IBM company"`) will only match the dotted abrreviation if you increase the phrase slack (using the advanced search panel control, or the `o` query language modifier). Literal occurences of the word will be matched normally.

3.1.11.3 Others

Using fields You can use the query language and field specifications to only search certain parts of documents. This can be especially helpful with email, for example only searching emails from a specific originator: `search tips from:helpful gui`

Ajusting the result table columns When displaying results in table mode, you can use a right click on the table headers to activate a pop-up menu which will let you adjust what columns are displayed. You can drag the column headers to adjust their order. You can click them to sort by the field displayed in the column. You can also save the result list in CSV format.

Query explanation You can get an exact description of what the query looked for, including stem expansion, and Boolean operators used, by clicking on the result list header.

Advanced search history As of Recoll 1.18, you can display any of the last 100 complex searches performed by using the up and down arrow keys while the advanced search panel is active.

Browsing the result list inside a preview window Entering **Shift-Down** or **Shift-Up** (**Shift +** an arrow key) in a preview window will display the next or the previous document from the result list. Any secondary search currently active will be executed on the new document.

Scrolling the result list from the keyboard You can use **PageUp** and **PageDown** to scroll the result list, **Shift+Home** to go back to the first page. These work even while the focus is in the search entry.

Editing a new search while the focus is not in the search entry You can use the **Ctrl-Shift-S** shortcut to return the cursor to the search entry (and select the current search text), while the focus is anywhere in the main window.

Forced opening of a preview window You can use **Shift+**Click on a result list `Preview` link to force the creation of a preview window instead of a new tab in the existing one.

Closing previews Entering **Ctrl-W** in a tab will close it (and, for the last tab, close the preview window). Entering **Esc** will close the preview window and all its tabs.

Printing previews Entering **Ctrl-P** in a preview window will print the currently displayed text.

Quitting Entering **Ctrl-Q** almost anywhere will close the application.

3.1.12 Customizing the search interface

You can customize some aspects of the search interface by using the GUI configuration entry in the Preferences menu.

There are several tabs in the dialog, dealing with the interface itself, the parameters used for searching and returning results, and what indexes are searched.

User interface parameters:

- Highlight color for query terms: Terms from the user query are highlighted in the result list samples and the preview window. The color can be chosen here. Any Qt color string should work (ie `red`, `#ff0000`). The default is `blue`.
- Style sheet: The name of a Qt style sheet text file which is applied to the whole Recoll application on startup. The default value is empty, but there is a skeleton style sheet (`recoll.qss`) inside the `/usr/share/recoll/examples` directory. Using a style sheet, you can change most **recoll** graphical parameters: colors, fonts, etc. See the sample file for a few simple examples.

  You should be aware that parameters (e.g.: the background color) set inside the Recoll GUI style sheet will override global system preferences, with possible strange side effects: for example if you set the foreground to a light color and the background to a dark one in the desktop preferences, but only the background is set inside the Recoll style sheet, and it is light too, then text will appear light-on-light inside the Recoll GUI.
- Maximum text size highlighted for preview Inserting highlights on search term inside the text before inserting it in the preview window involves quite a lot of processing, and can be disabled over the given text size to speed up loading.
- Prefer HTML to plain text for preview if set, Recoll will display HTML as such inside the preview window. If this causes problems with the Qt HTML display, you can uncheck it to display the plain text version instead.
- Plain text to HTML line style: when displaying plain text inside the preview window, Recoll tries to preserve some of the original text line breaks and indentation. It can either use PRE HTML tags, which will well preserve the indentation but will force horizontal scrolling for long lines, or use BR tags to break at the original line breaks, which will let the editor introduce other line breaks according to the window width, but will lose some of the original indentation. The third option has been available in recent releases and is probably now the best one: use PRE tags with line wrapping.
- Use desktop preferences to choose document editor: if this is checked, the **xdg-open** utility will be used to open files when you click the Open link in the result list, instead of the application defined in `mimeview`. **xdg-open** will in term use your desktop preferences to choose an appropriate application.
- Exceptions: when using the desktop preferences for opening documents, these are mime types that will still be opened according to Recoll preferences. This is useful for passing parameters like page numbers or search strings to applications that support them (e.g. evince). This cannot be done with **xdg-open** which only supports passing one parameter.

- Choose editor applications this will let you choose the command started by the Open links inside the result list, for specific document types.
- Display category filter as toolbar... this will let you choose if the document categories are displayed as a list or a set of buttons.
- Auto-start simple search on white space entry: if this is checked, a search will be executed each time you enter a space in the simple search input field. This lets you look at the result list as you enter new terms. This is off by default, you may like it or not...
- Start with advanced search dialog open : If you use this dialog frequently, checking the entries will get it to open when recoll starts.
- Remember sort activation state if set, Recoll will remember the sort tool stat between invocations. It normally starts with sorting disabled.

Result list parameters:

- Number of results in a result page
- Result list font: There is quite a lot of information shown in the result list, and you may want to customize the font and/or font size. The rest of the fonts used by Recoll are determined by your generic Qt config (try the **qtconfig** command).
- Edit result list paragraph format string: allows you to change the presentation of each result list entry. See the result list customisation section.
- Edit result page HTML header insert: allows you to define text inserted at the end of the result page HTML header. More detail in the result list customisation section.
- Date format: allows specifying the format used for displaying dates inside the result list. This should be specified as an strftime() string (man strftime).
- Abstract snippet separator: for synthetic abstracts built from index data, which are usually made of several snippets from different parts of the document, this defines the snippet separator, an ellipsis by default.

Search parameters:

- Hide duplicate results: decides if result list entries are shown for identical documents found in different places.
- Stemming language: stemming obviously depends on the document's language. This listbox will let you chose among the stemming databases which were built during indexing (this is set in the main configuration file), or later added with **recollindex -s** (See the recollindex manual). Stemming languages which are dynamically added will be deleted at the next indexing pass unless they are also added in the configuration file.
- Automatically add phrase to simple searches: a phrase will be automatically built and added to simple searches when looking for `Any terms`. This will give a relevance boost to the results where the search terms appear as a phrase (consecutive and in order).
- Autophrase term frequency threshold percentage: very frequent terms should not be included in automatic phrase searches for performance reasons. The parameter defines the cutoff percentage (percentage of the documents where the term appears).
- Replace abstracts from documents: this decides if we should synthesize and display an abstract in place of an explicit abstract found within the document itself.
- Dynamically build abstracts: this decides if Recoll tries to build document abstracts (lists of *snippets*) when displaying the result list. Abstracts are constructed by taking context from the document information, around the search terms.
- Synthetic abstract size: adjust to taste...
- Synthetic abstract context words: how many words should be displayed around each term occurrence.
- Query language magic file name suffixes: a list of words which automatically get turned into `ext:xxx` file name suffix clauses when starting a query language query (ie: `doc xls xlsx...`). This will save some typing for people who use file types a lot when querying.

External indexes:

This panel will let you browse for additional indexes that you may want to search. External indexes are designated by their database directory (ie: `/home/someothergui/.recoll/xapiandb`, `/usr/local/recollglobal/xapiandb`).

Once entered, the indexes will appear in the External indexes list, and you can chose which ones you want to use at any moment by checking or unchecking their entries.

Your main database (the one the current configuration indexes to), is always implicitly active. If this is not desirable, you can set up your configuration so that it indexes, for example, an empty directory. An alternative indexer may also need to implement a way of purging the index from stale data,

3.1.12.1 The result list format

The result list presentation can be exhaustively customized by adjusting two elements:

- The paragraph format
- HTML code inside the header section

These can be edited from the Result list tab of the GUI configuration.

Newer versions of Recoll (from 1.17) use a WebKit HTML object by default (this may be disabled at build time), and total customisation is possible with full support for CSS and Javascript. Conversely, there are limits to what you can do with the older Qt QTextBrowser, but still, it is possible to decide what data each result will contain, and how it will be displayed.

No more detail will be given about the header part (only useful with the WebKit build), if there are restrictions to what you can do, they are beyond this author's HTML/CSS/Javascript abilities... There are a few examples on the page about customising the result list on the Recoll web site.

3.1.12.1.1 The paragraph format

This is an arbitrary HTML string where the following printf-like `%` substitutions will be performed:

- **%A** Abstract
- **%D** Date
- **%I** Icon image name. This is normally determined from the mime type. The associations are defined inside the mimeconf configuration file. If a thumbnail for the file is found at the standard Freedesktop location, this will be displayed instead.
- **%K** Keywords (if any)
- **%L** Precooked Preview, Edit, and possibly Snippets links
- **%M** Mime type
- **%N** result Number inside the result page
- **%R** Relevance percentage
- **%S** Size information
- **%T** Title or Filename if not set.
- **%t** Title or Filename if not set.
- **%U** Url

The format of the Preview, Edit, and Snippets links is `<a href="P%N">`, `<a href="E%N">` and `<a href="A%N">` where *docnum* (%N) expands to the document number inside the result page).

In addition to the predefined values above, all strings like `%(fieldname)` will be replaced by the value of the field named `fieldname` for this document. Only stored fields can be accessed in this way, the value of indexed but not stored fields is not known at this point in the search process (see field configuration). There are currently very few fields stored by default, apart from the values above (only `author` and `filename`), so this feature will need some custom local configuration to be useful. An example candidate would be the `recipient` field which is generated by the message filters.

The default value for the paragraph format string is:

```
<img src="%I" align="left">%R %S %L   <b>%T</b><br>
%M %D   <i>%U</i> %i<br>
%A %K
```

You may, for example, try the following for a more web-like experience:

```
<u><b><a href="P%N">%T</a></b></u><br>
%A<font color=#008000>%U - %S</font> - %L
```

Note that the P%N link in the above paragraph makes the title a preview link. Or the clean looking:

```
<img src="%I" align="left">%L <font color="#900000">%R</font>
  <b>%T&</b><br>%S 
<font color="#808080"><i>%U</i></font>
<table bgcolor="#e0e0e0">
<tr><td><div>%A</div></td></tr>
</table>%K
```

These samples, and some others are on the web site, with pictures to show how they look.

It is also possible to define the value of the snippet separator inside the abstract section.

3.2 Searching with the KDE KIO slave

3.2.1 What's this

The Recoll KIO slave allows performing a Recoll search by entering an appropriate URL in a KDE open dialog, or with an HTML-based interface displayed in **Konqueror**.

The HTML-based interface is similar to the Qt-based interface, but slightly less powerful for now. Its advantage is that you can perform your search while staying fully within the KDE framework: drag and drop from the result list works normally and you have your normal choice of applications for opening files.

The alternative interface uses a directory view of search results. Due to limitations in the current KIO slave interface, it is currently not obviously useful (to me).

The interface is described in more detail inside a help file which you can access by entering `recoll:/` inside the **konqueror** URL line (this works only if the recoll KIO slave has been previously installed).

The instructions for building this module are located in the source tree. See: `kde/kio/recoll/00README.txt`. Some Linux distributions do package the kio-recoll module, so check before diving into the build process, maybe it's already out there ready for one-click installation.

3.2.2 Searchable documents

As a sample application, the Recoll KIO slave could allow preparing a set of HTML documents (for example a manual) so that they become their own search interface inside **konqueror**.

This can be done by either explicitly inserting `<a href="recoll://...">` links around some document areas, or automatically by adding a very small javascript program to the documents, like the following example, which would initiate a search by double-clicking any term:

```
<script language="JavaScript">
    function recollsearch() {
        var t = document.getSelection();
        window.location.href = 'recoll://search/query?qtp=a&p=0&q=' +
            encodeURIComponent(t);
    }
</script>
 ....
<body ondblclick="recollsearch()">
```

3.3 Searching on the command line

There are several ways to obtain search results as a text stream, without a graphical interface:

- By passing option `-t` to the **recoll** program.
- By using the **recollq** program.
- By writing a custom Python program, using the Recoll Python API.

The first two methods work in the same way and accept/need the same arguments (except for the additional `-t` to **recoll**). The query to be executed is specified as command line arguments.

recollq is not built by default. You can use the `Makefile` in the `query` directory to build it. This is a very simple program, and if you can program a little c++, you may find it useful to taylor its output format to your needs.

recollq has a man page (not installed by default, look in the `doc/man` directory). The Usage string is as follows:

```
recollq: usage:
 -P: Show the date span for all the documents present in the index
 [-o|-a|-f] [-q] <query string>
 Runs a recoll query and displays result lines.
  Default: will interpret the argument(s) as a xesam query string
    query may be like:
    implicit AND, Exclusion, field spec:    t1 -t2 title:t3
    OR has priority: t1 OR t2 t3 OR t4 means (t1 OR t2) AND (t3 OR t4)
    Phrase: "t1 t2" (needs additional quoting on cmd line)
  -o Emulate the GUI simple search in ANY TERM mode
  -a Emulate the GUI simple search in ALL TERMS mode
  -f Emulate the GUI simple search in filename mode
  -q is just ignored (compatibility with the recoll GUI command line)
Common options:
    -c <configdir> : specify config directory, overriding $RECOLL_CONFDIR
    -d also dump file contents
    -n [first-]<cnt> define the result slice. The default value for [first]
       is 0. Without the option, the default max count is 2000.
       Use n=0 for no limit
    -b : basic. Just output urls, no mime types or titles
    -Q : no result lines, just the processed query and result count
    -m : dump the whole document meta[] array for each result
    -A : output the document abstracts
    -S fld : sort by field <fld>
    -D : sort descending
    -i <dbdir> : additional index, several can be given
    -e use url encoding (%xx) for urls
    -F <field name list> : output exactly these fields for each result.
       The field values are encoded in base64, output in one line and
       separated by one space character. This is the recommended format
```

```
       for use by other programs. Use a normal query with option -m to
       see the field names.
```

Sample execution:

```
recollq 'ilur -nautique mime:text/html'
Recoll query: ((((ilur:(wqf=11) OR ilurs) AND_NOT (nautique:(wqf=11)
  OR nautiques OR nautiqu OR nautiquement)) FILTER Ttext/html))
4 results
text/html       [file:///Users/uncrypted-dockes/projets/bateaux/ilur/comptes.html]      [ ↩
    comptes.html]  18593   bytes
text/html       [file:///Users/uncrypted-dockes/projets/nautique/webnautique/articles/ilur1 ↩
    /index.html] [Constructio...
text/html       [file:///Users/uncrypted-dockes/projets/pagepers/index.html]    [psxtcl/ ↩
    writemime/recoll]...
text/html       [file:///Users/uncrypted-dockes/projets/bateaux/ilur/factEtCie/recu-chasse- ↩
    maree....
```

3.4 Path translations

In some cases, the document paths stored inside the index do not match the actual ones, so that document previews and accesses will fail. This can occur in a number of circumstances:

- When using multiple indexes it is a relatively common occurrence that some will actually reside on a remote volume, for exemple mounted via NFS. In this case, the paths used to access the documents on the local machine are not necessarily the same than the ones used while indexing on the remote machine. For example, `/home/me` may have been used as a `topdirs` elements while indexing, but the directory might be mounted as `/net/server/home/me` on the local machine.
- The case may also occur with removable disks. It is perfectly possible to configure an index to live with the documents on the removable disk, but it may happen that the disk is not mounted at the same place so that the documents paths from the index are invalid.
- As a last exemple, one could imagine that a big directory has been moved, but that it is currently inconvenient to run the indexer.

More generally, the path translation facility may be useful whenever the documents paths seen by the indexer are not the same as the ones which should be used at query time.

Recoll has a facility for rewriting access paths when extracting the data from the index. The translations can be defined for the main index and for any additional query index.

In the above NFS example, Recoll could be instructed to rewrite any `file:///home/me` URL from the index to `file:///net/server/home/me`, allowing accesses from the client.

The translations are defined in the ptrans configuration file, which can be edited by hand or from the GUI external indexes configuration dialog.

3.5 The query language

The query language processor is activated in the GUI simple search entry when the search mode selector is set to Query Language. It can also be used with the KIO slave or the command line search. It broadly has the same capabilities as the complex search interface in the GUI.

The language is based on the (seemingly defunct) Xesam user search language specification.

If the results of a query language search puzzle you and you doubt what has been actually searched for, you can use the GUI `Show Query` link at the top of the result list to check the exact query which was finally executed by Xapian.

Here follows a sample request that we are going to explain:

```
author:"john doe" Beatles OR Lennon Live OR Unplugged -potatoes
```

This would search for all documents with *John Doe* appearing as a phrase in the author field (exactly what this is would depend on the document type, ie: the `From:` header, for an email message), and containing either *beatles* or *lennon* and either *live* or *unplugged* but not *potatoes* (in any part of the document).

An element is composed of an optional field specification, and a value, separated by a colon (the field separator is the last colon in the element). Example: *Eugenie*, *author:balzac*, *dc:title:grandet*

The colon, if present, means "contains". Xesam defines other relations, which are mostly unsupported for now (except in special cases, described further down).

All elements in the search entry are normally combined with an implicit AND. It is possible to specify that elements be OR'ed instead, as in *Beatles* OR *Lennon*. The OR must be entered literally (capitals), and it has priority over the AND associations: *word1 word2* OR *word3* means *word1* AND (*word2* OR *word3*) not (*word1* AND *word2*) OR *word3*. Explicit parenthesis are *not* supported.

An element preceded by a − specifies a term that should *not* appear. Pure negative queries are forbidden.

As usual, words inside quotes define a phrase (the order of words is significant), so that *title:"prejudice pride"* is not the same as *title:prejudice title:pride*, and is unlikely to find a result.

Modifiers can be set on a phrase clause, for example to specify a proximity search (unordered). See the modifier section.

Recoll currently manages the following default fields:

- `title`, `subject` or `caption` are synonyms which specify data to be searched for in the document title or subject.
- `author` or `from` for searching the documents originators.
- `recipient` or `to` for searching the documents recipients.
- `keyword` for searching the document-specified keywords (few documents actually have any).
- `filename` for the document's file name.
- `ext` specifies the file name extension (Ex: `ext:html`)

The field syntax also supports a few field-like, but special, criteria:

- `dir` for filtering the results on file location (Ex: `dir:/home/me/somedir`). `-dir` also works to find results not in the specified directory (release >= 1.15.8). A tilde inside the value will be expanded to the home directory. Wildcards will be expanded, but please have a look at an important limitation of wildcards in path filters.

  Relative paths also make sense, for example, `dir:share/doc` would match either `/usr/share/doc` or `/usr/local/share/doc`

  Several `dir` clauses can be specified, both positive and negative. For example the following makes sense:

  ```
  dir:recoll dir:src -dir:utils -dir:common
  ```

  This would select results which have both `recoll` and `src` in the path (in any order), and which have not either `utils` or `common`.

  You can also use `OR` conjunctions with `dir:` clauses.

  A special aspect of `dir` clauses is that the values in the index are not transcoded to UTF-8, and never lower-cased or unaccented, but stored as binary. This means that you need to enter the values in the exact lower or upper case, and that searches for names with diacritics may sometimes be impossible because of character set conversion issues. Non-ASCII UNIX file paths are an unending source of trouble and are best avoided.

  You need to use double-quotes around the path value if it contains space characters.

- `size` for filtering the results on file size. Example: `size<10000`. You can use <, > or = as operators. You can specify a range like the following: `size>100 size<1000`. The usual `k/K, m/M, g/G, t/T` can be used as (decimal) multipliers. Ex: `size>1k` to search for files bigger than 1000 bytes.

- date for searching or filtering on dates. The syntax for the argument is based on the ISO8601 standard for dates and time intervals. Only dates are supported, no times. The general syntax is 2 elements separated by a / character. Each element can be a date or a period of time. Periods are specified as PnYnMnD. The n numbers are the respective numbers of years, months or days, any of which may be missing. Dates are specified as YYYY-MM-DD. The days and months parts may be missing. If the / is present but an element is missing, the missing element is interpreted as the lowest or highest date in the index. Examples:
  - 2001-03-01/2002-05-01 the basic syntax for an interval of dates.
  - 2001-03-01/P1Y2M the same specified with a period.
  - 2001/ from the beginning of 2001 to the latest date in the index.
  - 2001 the whole year of 2001
  - P2D/ means 2 days ago up to now if there are no documents with dates in the future.
  - /2003 all documents from 2003 or older.

  Periods can also be specified with small letters (ie: p2y).
- mime or format for specifying the mime type. This one is quite special because you can specify several values which will be OR'ed (the normal default for the language is AND). Ex: mime:text/plain mime:text/html. Specifying an explicit boolean operator before a mime specification is not supported and will produce strange results. You can filter out certain types by using negation (-mime:some/type), and you can use wildcards in the value (mime:text/*). Note that mime is the ONLY field with an OR default. You do need to use OR with ext terms for example.
- type or rclcat for specifying the category (as in text/media/presentation/etc.). The classification of mime types in categories is defined in the Recoll configuration (mimeconf), and can be modified or extended. The default category names are those which permit filtering results in the main GUI screen. Categories are OR'ed like mime types above. This can't be negated with - either.

Words inside phrases and capitalized words are not stem-expanded. Wildcards may be used anywhere inside a term. Specifying a wild-card on the left of a term can produce a very slow search (or even an incorrect one if the expansion is truncated because of excessive size). Also see More about wildcards.

The document filters used while indexing have the possibility to create other fields with arbitrary names, and aliases may be defined in the configuration, so that the exact field search possibilities may be different for you if someone took care of the customisation.

3.5.1 Modifiers

Some characters are recognized as search modifiers when found immediately after the closing double quote of a phrase, as in "some term"modifierchars. The actual "phrase" can be a single term of course. Supported modifiers:

- l can be used to turn off stemming (mostly makes sense with p because stemming is off by default for phrases).
- o can be used to specify a "slack" for phrase and proximity searches: the number of additional terms that may be found between the specified ones. If o is followed by an integer number, this is the slack, else the default is 10.
- p can be used to turn the default phrase search into a proximity one (unordered). Example:"order any in"p
- C will turn on case sensitivity (if the index supports it).
- D will turn on diacritics sensitivity (if the index supports it).
- A weight can be specified for a query element by specifying a decimal value at the start of the modifiers. Example: "Import ant"2.5.

3.6 Search case and diacritics sensitivity

For Recoll versions 1.18 and later, and *when working with a raw index* (not the default), searches can be made sensitive to character case and diacritics. How this happens is controlled by configuration variables and what search data is entered.

The general default is that searches are insensitive to case and diacritics. An entry of `resume` will match any of `Resume`, `RESUME`, `résumé`, `Résumé` etc.

Two configuration variables can automate switching on sensitivity:

autodiacsens If this is set, search sensitivity to diacritics will be turned on as soon as an accented character exists in a search term. When the variable is set to true, `resume` will start a diacritics-unsensitive search, but `résumé` will be matched exactly. The default value is *false*.

autocasesens If this is set, search sensitivity to character case will be turned on as soon as an upper-case character exists in a search term *except for the first one*. When the variable is set to true, `us` or `Us` will start a diacritics-unsensitive search, but `US` will be matched exactly. The default value is *true* (contrary to `autodiacsens`).

As in the past, capitalizing the first letter of a word will turn off its stem expansion and have no effect on case-sensitivity.

You can also explicitely activate case and diacritics sensitivity by using modifiers with the query language. `C` will make the term case-sensitive, and `D` will make it diacritics-sensitive. Examples:

```
"us"C
```

will search for the term `us` exactly (`Us` will not be a match).

```
"resume"D
```

will search for the term `resume` exactly (`résumé` will not be a match).

When either case or diacritics sensitivity is activated, stem expansion is turned off. Having both does not make much sense.

3.7 Anchored searches and wildcards

Some special characters are interpreted by Recoll in search strings to expand or specialize the search. Wildcards expand a root term in controlled ways. Anchor characters can restrict a search to succeed only if the match is found at or near the beginning of the document or one of its fields.

3.7.1 More about wildcards

All words entered in Recoll search fields will be processed for wildcard expansion before the request is finally executed.

The wildcard characters are:

- `*` which matches 0 or more characters.
- `?` which matches a single character.
- `[]` which allow defining sets of characters to be matched (ex: `[abc]` matches a single character which may be 'a' or 'b' or 'c', `[0-9]` matches any number.

You should be aware of a few things when using wildcards.

- Using a wildcard character at the beginning of a word can make for a slow search because Recoll will have to scan the whole index term list to find the matches. However, this is much less a problem for field searches, and queries like *`author:*@domain.com`* can sometimes be very useful.

- For Recoll version 18 only, when working with a raw index (preserving character case and diacritics), the literal part of a wildcard expression will be matched exactly for case and diacritics. This is not true any more for versions 19 and later.
- Using a * at the end of a word can produce more matches than you would think, and strange search results. You can use the term explorer tool to check what completions exist for a given term. You can also see exactly what search was performed by clicking on the link at the top of the result list. In general, for natural language terms, stem expansion will produce better results than an ending * (stem expansion is turned off when any wildcard character appears in the term).

3.7.1.1 Wildcards and path filtering

Due to the way that Recoll processes wildcards inside `dir` path filtering clauses, they will have a multiplicative effect on the query size. A clause containg wildcards in several paths elements, like, for example, `dir:` *`/home/me/*/*/docdir`*, will almost certainly fail if your indexed tree is of any realistic size.

Depending on the case, you may be able to work around the issue by specifying the paths elements more narrowly, with a constant prefix, or by using 2 separate `dir:` clauses instead of multiple wildcards, as in `dir:` *`/home/me`* `dir:` *`docdir`*. The latter query is not equivalent to the initial one because it does not specify a number of directory levels, but that's the best we can do (and it may be actually more useful in some cases).

3.7.2 Anchored searches

Two characters are used to specify that a search hit should occur at the beginning or at the end of the text. `^` at the beginning of a term or phrase constrains the search to happen at the start, `$` at the end force it to happen at the end.

As this function is implemented as a phrase search it is possible to specify a maximum distance at which the hit should occur, either through the controls of the advanced search panel, or using the query language, for example, as in:

```
"^someterm"o10
```

which would force `someterm` to be found within 10 terms of the start of the text. This can be combined with a field search as in `somefield:"^someterm"o10` or `somefield:someterm$`.

This feature can also be used with an actual phrase search, but in this case, the distance applies to the whole phrase and anchor, so that, for example, `bla bla my unexpected term` at the beginning of the text would be a match for `"^my term"o5`.

Anchored searches can be very useful for searches inside somewhat structured documents like scientific articles, in case explicit metadata has not been supplied (a most frequent case), for example for looking for matches inside the abstract or the list of authors (which occur at the top of the document).

3.8 Desktop integration

Being independant of the desktop type has its drawbacks: Recoll desktop integration is minimal. However there are a few tools available:

- The KDE KIO Slave was described in a previous section.
- If you use a recent version of Ubuntu Linux, you may find the Ubuntu Unity Lens module useful.
- There is also an independantly developed Krunner plugin.

Here follow a few other things that may help.

3.8.1 Hotkeying recoll

It is surprisingly convenient to be able to show or hide the Recoll GUI with a single keystroke. Recoll comes with a small Python script, based on the libwnck window manager interface library, which will allow you to do just this. The detailed instructions are on this wiki page.

3.8.2 The KDE Kicker Recoll applet

This is probably obsolete now. Anyway:

The Recoll source tree contains the source code to the recoll_applet, a small application derived from the find_applet. This can be used to add a small Recoll launcher to the KDE panel.

The applet is not automatically built with the main Recoll programs, nor is it included with the main source distribution (because the KDE build boilerplate makes it relatively big). You can download its source from the recoll.org download page. Use the omnipotent **`configure;make;make install`** incantation to build and install.

You can then add the applet to the panel by right-clicking the panel and choosing the Add applet entry.

The recoll_applet has a small text window where you can type a Recoll query (in query language form), and an icon which can be used to restrict the search to certain types of files. It is quite primitive, and launches a new recoll GUI instance every time (even if it is already running). You may find it useful anyway.

Chapter 4

Programming interface

Recoll has an Application Programming Interface, usable both for indexing and searching, currently accessible from the Python language.

Another less radical way to extend the application is to write filters for new types of documents.

The processing of metadata attributes for documents (`fields`) is highly configurable.

4.1 Writing a document filter

Recoll filters cooperate to translate from the multitude of input document formats, simple ones as opendocument, acrobat), or compound ones such as Zip or Email, into the final Recoll indexing input format, which may be `text/plain` or `text/html`. Most filters are executable programs or scripts. A few filters are coded in C++ and live inside **recollindex**. This latter kind will not be described here.

There are currently (1.18 and since 1.13) two kinds of external executable filters:

- Simple filters (`exec` filters) run once and exit. They can be bare programs like antiword, or scripts using other programs. They are very simple to write, because they just need to print the converted document to the standard output. Their output can be `text/plain` or `text/html`.
- Multiple filters (`execm` filters), run as long as their master process (**recollindex**) is active. They can process multiple files (sparing the process startup time which can be very significant), or multiple documents per file (e.g.: for zip or chm files). They communicate with the indexer through a simple protocol, but are nevertheless a bit more complicated than the older kind. Most of new filters are written in Python, using a common module to handle the protocol. There is an exception, **rclimg** which is written in Perl. The subdocuments output by these filters can be directly indexable (text or HTML), or they can be other simple or compound documents that will need to be processed by another filter.

In both cases, filters deal with regular file system files, and can process either a single document, or a linear list of documents in each file. Recoll is responsible for performing up to date checks, deal with more complex embedding and other upper level issues.

In the extreme case of a simple filter returning a document in `text/plain` format, no metadata can be transferred from the filter to the indexer. Generic metadata, like document size or modification date, will be gathered and stored by the indexer.

Filters that produce `text/html` format can return an arbitrary amount of metadata inside HTML `meta` tags. These will be processed according to the directives found in the fields configuration file.

The filters that can handle multiple documents per file return a single piece of data to identify each document inside the file. This piece of data, called an `ipath element` will be sent back by Recoll to extract the document at query time, for previewing, or for creating a temporary file to be opened by a viewer.

The following section describes the simple filters, and the next one gives a few explanations about the `execm` ones. You could conceivably write a simple filter with only the elements in the manual. This will not be the case for the other ones, for which you will have to look at the code.

4.1.1 Simple filters

Recoll simple filters are usually shell-scripts, but this is in no way necessary. Extracting the text from the native format is the difficult part. Outputting the format expected by Recoll is trivial. Happily enough, most document formats have translators or text extractors which can be called from the filter. In some cases the output of the translating program is completely appropriate, and no intermediate shell-script is needed.

Filters are called with a single argument which is the source file name. They should output the result to stdout.

When writing a filter, you should decide if it will output plain text or HTML. Plain text is simpler, but you will not be able to add metadata or vary the output character encoding (this will be defined in a configuration file). Additionally, some formatting may be easier to preserve when previewing HTML. Actually the deciding factor is metadata: Recoll has a way to extract metadata from the HTML header and use it for field searches..

The `RECOLL_FILTER_FORPREVIEW` environment variable (values `yes`, `no`) tells the filter if the operation is for indexing or previewing. Some filters use this to output a slightly different format, for example stripping uninteresting repeated keywords (ie: `Subject:` for email) when indexing. This is not essential.

You should look at one of the simple filters, for example **rclps** for a starting point.

Don't forget to make your filter executable before testing !

4.1.2 "Multiple" filters

If you can program and want to write an `execm` filter, it should not be too difficult to make sense of one of the existing modules. For example, look at **rclzip** which uses Zip file paths as identifiers (`ipath`), and **rclics**, which uses an integer index. Also have a look at the comments inside the `internfile/mh_execm.h` file and possibly at the corresponding module.

`execm` filters sometimes need to make a choice for the nature of the `ipath` elements that they use in communication with the indexer. Here are a few guidelines:

- Use ASCII or UTF-8 (if the identifier is an integer print it, for example, like printf %d would do).
- If at all possible, the data should make some kind of sense when printed to a log file to help with debugging.
- Recoll uses a colon (`:`) as a separator to store a complex path internally (for deeper embedding). Colons inside the `ipath` elements output by a filter will be escaped, but would be a bad choice as a filter-specific separator (mostly, again, for debugging issues).

In any case, the main goal is that it should be easy for the filter to extract the target document, given the file name and the `ipath` element.

`execm` filters will also produce a document with a null `ipath` element. Depending on the type of document, this may have some associated data (e.g. the body of an email message), or none (typical for an archive file). If it is empty, this document will be useful anyway for some operations, as the parent of the actual data documents.

4.1.3 Telling Recoll about the filter

There are two elements that link a file to the filter which should process it: the association of file to mime type and the association of a mime type with a filter.

The association of files to mime types is mostly based on name suffixes. The types are defined inside the mimemap file. Example:

```
.doc = application/msword
```

If no suffix association is found for the file name, Recoll will try to execute the **file -i** command to determine a mime type.

The association of file types to filters is performed in the mimeconf file. A sample will probably be of better help than a long explanation:

```
[index]
application/msword = exec antiword -t -i 1 -m UTF-8;\
     mimetype = text/plain ; charset=utf-8

application/ogg = exec rclogg

text/rtf = exec unrtf --nopict --html; charset=iso-8859-1; mimetype=text/html

application/x-chm = execm rclchm
```

The fragment specifies that:

- `application/msword` files are processed by executing the **antiword** program, which outputs `text/plain` encoded in `utf-8`.
- `application/ogg` files are processed by the **rclogg** script, with default output type (`text/html`, with encoding specified in the header, or `utf-8` by default).
- `text/rtf` is processed by **unrtf**, which outputs `text/html`. The `iso-8859-1` encoding is specified because it is not the `utf-8` default, and not output by **unrtf** in the HTML header section.
- `application/x-chm` is processed by a persistant filter. This is determined by the `execm` keyword.

4.1.4 Filter HTML output

The output HTML could be very minimal like the following example:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
  </head>
  <body>
   Some text content
  </body>
</html>
```

You should take care to escape some characters inside the text by transforming them into appropriate entities. At the very minimum, "`&`" should be transformed into "`&`", "`<`" should be transformed into "`<`". This is not always properly done by translating programs which output HTML, and of course never by those which output plain text.

When encapsulating plain text in an HTML body, the display of a preview may be improved by enclosing the text inside `<pre>` tags.

The character set needs to be specified in the header. It does not need to be UTF-8 (Recoll will take care of translating it), but it must be accurate for good results.

Recoll will process `meta` tags inside the header as possible document fields candidates. Documents fields can be processed by the indexer in different ways, for searching or displaying inside query results. This is described in a following section.

By default, the indexer will process the standard header fields if they are present: `title`, `meta/description`, and `meta/keywords` are both indexed and stored for query-time display.

A predefined non-standard `meta` tag will also be processed by Recoll without further configuration: if a `date` tag is present and has the right format, it will be used as the document date (for display and sorting), in preference to the file modification date. The date format should be as follows:

```
<meta name="date" content="YYYY-mm-dd HH:MM:SS">
or
<meta name="date" content="YYYY-mm-ddTHH:MM:SS">
```

Example:

```
<meta name="date" content="2013-02-24 17:50:00">
```

Filters also have the possibility to "invent" field names. This should also be output as meta tags:

```
<meta name="somefield" content="Some textual data" />
```

You can embed HTML markup inside the content of custom fields, for improving the display inside result lists. In this case, add a (wildly non-standard) `markup` attribute to tell Recoll that the value is HTML and should not be escaped for display.

```
<meta name="somefield" markup="html" content="Some <i>textual</i> data" />
```

As written above, the processing of fields is described in a further section.

4.1.5 Page numbers

The indexer will interpret `^L` characters in the filter output as indicating page breaks, and will record them. At query time, this allows starting a viewer on the right page for a hit or a snippet. Currently, only the PDF, Postscript and DVI filters generate page breaks.

4.2 Field data processing

`Fields` are named pieces of information in or about documents, like `title`, `author`, `abstract`.

The field values for documents can appear in several ways during indexing: either output by filters as `meta` fields in the HTML header section, or extracted from file extended attributes, or added as attributes of the `Doc` object when using the API, or again synthetized internally by Recoll.

The Recoll query language allows searching for text in a specific field.

Recoll defines a number of default fields. Additional ones can be output by filters, and described in the `fields` configuration file.

Fields can be:

- `indexed`, meaning that their terms are separately stored in inverted lists (with a specific prefix), and that a field-specific search is possible.
- `stored`, meaning that their value is recorded in the index data record for the document, and can be returned and displayed with search results.

A field can be either or both indexed and stored. This and other aspects of fields handling is defined inside the `fields` configuration file.

The sequence of events for field processing is as follows:

- During indexing, **recollindex** scans all `meta` fields in HTML documents (most document types are transformed into HTML at some point). It compares the name for each element to the configuration defining what should be done with fields (the `fields` file)
- If the name for the `meta` element matches one for a field that should be indexed, the contents are processed and the terms are entered into the index with the prefix defined in the `fields` file.
- If the name for the `meta` element matches one for a field that should be stored, the content of the element is stored with the document data record, from which it can be extracted and displayed at query time.
- At query time, if a field search is performed, the index prefix is computed and the match is only performed against appropriately prefixed terms in the index.

- At query time, the field can be displayed inside the result list by using the appropriate directive in the definition of the result list paragraph format. All fields are displayed on the fields screen of the preview window (which you can reach through the right-click menu). This is independant of the fact that the search which produced the results used the field or not.

You can find more information in the section about the fields file, or in comments inside the file.

You can also have a look at the example on the Wiki, detailing how one could add a *page count* field to pdf documents for displaying inside result lists.

4.3 API

4.3.1 Interface elements

A few elements in the interface are specific and and need an explanation.

udi An udi (unique document identifier) identifies a document. Because of limitations inside the index engine, it is restricted in length (to 200 bytes), which is why a regular URI cannot be used. The structure and contents of the udi is defined by the application and opaque to the index engine. For example, the internal file system indexer uses the complete document path (file path + internal path), truncated to length, the suppressed part being replaced by a hash value.

ipath This data value (set as a field in the Doc object) is stored, along with the URL, but not indexed by Recoll. Its contents are not interpreted, and its use is up to the application. For example, the Recoll internal file system indexer stores the part of the document access path internal to the container file (`ipath` in this case is a list of subdocument sequential numbers). url and ipath are returned in every search result and permit access to the original document.

Stored and indexed fields The `fields` file inside the Recoll configuration defines which document fields are either "indexed" (searchable), "stored" (retrievable with search results), or both.

Data for an external indexer, should be stored in a separate index, not the one for the Recoll internal file system indexer, except if the latter is not used at all). The reason is that the main document indexer purge pass would remove all the other indexer's documents, as they were not seen during indexing. The main indexer documents would also probably be a problem for the external indexer purge operation.

4.3.2 Python interface

4.3.2.1 Introduction

Recoll versions after 1.11 define a Python programming interface, both for searching and indexing. The indexing portion has seen little use, but the searching one is used in the Recoll Ubuntu Unity Lens and Recoll Web UI.

The API is inspired by the Python database API specification. There were two major changes in recent Recoll versions:

- The basis for the Recoll API changed from Python database API version 1.0 (Recoll versions up to 1.18.1), to version 2.0 (Recoll 1.18.2 and later).
- The `recoll` module became a package (with an internal `recoll` module) as of Recoll version 1.19, in order to add more functions. For existing code, this only changes the way the interface must be imported.

We will mostly describe the new API and package structure here. A paragraph at the end of this section will explain a few differences and ways to write code compatible with both versions.

The Python interface can be found in the source package, under `python/recoll`.

The `python/recoll/` directory contains the usual `setup.py`. After configuring the main Recoll code, you can use the script to build and install the Python module:

```
cd recoll-xxx/python/recoll
python setup.py build
python setup.py install
```

The normal Recoll installer installs the Python API along with the main code.

When installing from a repository, and depending on the distribution, the Python API can sometimes be found in a separate package.

4.3.2.2 Recoll package

The `recoll` package contains two modules:

- The `recoll` module contains functions and classes used to query (or update) the index.
- The `rclextract` module contains functions and classes used to access document data.

4.3.2.3 The recoll module

4.3.2.3.1 Functions

connect(confdir=None, extra_dbs=None, writable = False) The `connect()` function connects to one or several Recoll index(es) and returns a `Db` object.

- `confdir` may specify a configuration directory. The usual defaults apply.
- `extra_dbs` is a list of additional indexes (Xapian directories).
- `writable` decides if we can index new data through this connection.

This call initializes the recoll module, and it should always be performed before any other call or object creation.

4.3.2.3.2 Classes

4.3.2.3.2.1 The Db class

A Db object is created by a `connect()` function and holds a connection to a Recoll index.

METHODS

Db.close() Closes the connection. You can't do anything with the `Db` object after this.

Db.query(), Db.cursor() These aliases return a blank `Query` object for this index.

Db.setAbstractParams(maxchars, contextwords) Set the parameters used to build snippets (sets of keywords in context text fragments). `maxchars` defines the maximum total size of the abstract. `contextwords` defines how many terms are shown around the keyword.

Db.termMatch(match_type, expr, field='', maxlen=-1, casesens=False, diacsens=False, lang='english') Expand an expression against the index term list. Performs the basic function from the GUI term explorer tool. `match_type` can be either of `wildcard`, `regexp` or `stem`. Returns a list of terms expanded from the input expression.

4.3.2.3.2.2 The Query class

A `Query` object (equivalent to a cursor in the Python DB API) is created by a `Db.query()` call. It is used to execute index searches.

METHODS

Query.sortby(fieldname, ascending=True) Sort results by *`fieldname`*, in ascending or descending order. Must be called before executing the search.

Query.execute(query_string, stemming=1, stemlang="english") Starts a search for *`query_string`*, a Recoll search language string.

Query.executesd(SearchData) Starts a search for the query defined by the SearchData object.

Query.fetchmany(size=query.arraysize) Fetches the next `Doc` objects in the current search results, and returns them as an array of the required size, which is by default the value of the `arraysize` data member.

Query.fetchone() Fetches the next `Doc` object from the current search results.

Query.close() Closes the query. The object is unusable after the call.

Query.scroll(value, mode='relative') Adjusts the position in the current result set. `mode` can be `relative` or `absolute`.

Query.getgroups() Retrieves the expanded query terms as a list of pairs. Meaningful only after executexx In each pair, the first entry is a list of user terms (of size one for simple terms, or more for group and phrase clauses), the second a list of query terms as derived from the user terms and used in the Xapian Query.

Query.getxquery() Return the Xapian query description as a Unicode string. Meaningful only after executexx.

Query.highlight(text, ishtml = 0, methods = object) Will insert <span "class=rclmatch">, tags around the match areas in the input text and return the modified text. `ishtml` can be set to indicate that the input text is HTML and that HTML special characters should not be escaped. `methods` if set should be an object with methods startMatch(i) and endMatch() which will be called for each match and should return a begin and end tag

Query.makedocabstract(doc, methods = object)) Create a snippets abstract for `doc` (a `Doc` object) by selecting text around the match terms. If methods is set, will also perform highlighting. See the highlight method.

Query.__iter__() and Query.next() So that things like `for doc in query:` will work.

DATA DESCRIPTORS

Query.arraysize Default number of records processed by fetchmany (r/w).

Query.rowcount Number of records returned by the last execute.

Query.rownumber Next index to be fetched from results. Normally increments after each fetchone() call, but can be set/reset before the call to effect seeking (equivalent to using `scroll()`). Starts at 0.

4.3.2.3.2.3 The Doc class

A `Doc` object contains index data for a given document. The data is extracted from the index when searching, or set by the indexer program when updating. The Doc object has many attributes to be read or set by its user. It matches exactly the Rcl::Doc C++ object. Some of the attributes are predefined, but, especially when indexing, others can be set, the name of which will be processed as field names by the indexing configuration. Inputs can be specified as Unicode or strings. Outputs are Unicode objects. All dates are specified as Unix timestamps, printed as strings. Please refer to the `rcldb/rcldoc.h` C++ file for a description of the predefined attributes.

At query time, only the fields that are defined as `stored` either by default or in the `fields` configuration file will be meaningful in the `Doc` object. Especially this will not be the case for the document text. See the `rclextract` module for accessing document contents.

METHODS

get(key), [] operator Retrieve the named doc attribute

getbinurl() Retrieve the URL in byte array format (no transcoding), for use as parameter to a system call.

items() Return a dictionary of doc object keys/values

keys() list of doc object keys (attribute names).

4.3.2.3.2.4 The SearchData class

A `SearchData` object allows building a query by combining clauses, for execution by `Query.executesd()`. It can be used in replacement of the query language approach. The interface is going to change a little, so no detailed doc for now...

METHODS

addclause(type='and'|'or'|'excl'|'phrase'|'near'|'sub', qstring=string, slack=0, field='', stemming=1, subSearch=SearchData)

4.3.2.4 The rclextract module

Index queries do not provide document content (only a partial and unprecise reconstruction is performed to show the snippets text). In order to access the actual document data, the data extraction part of the indexing process must be performed (subdocument access and format translation). This is not trivial in general. The `rclextract` module currently provides a single class which can be used to access the data content for result documents.

4.3.2.4.1 Classes

4.3.2.4.1.1 The Extractor class

METHODS

Extractor(doc) An `Extractor` object is built from a `Doc` object, output from a query.

Extractor.textextract(ipath) Extract document defined by *`ipath`* and return a `Doc` object. The doc.text field has the document text as either text/plain or text/html according to doc.mimetype. The typical use would be as follows:

```
qdoc = query.fetchone()
extractor = recoll.Extractor(qdoc)
text = extractor.textextract(qdoc.ipath)
```

Extractor.idoctofile(ipath, targetmtype, outfile='') Extracts document into an output file, which can be given explicitly or will be created as a temporary file to be deleted by the caller. Typical use:

```
qdoc = query.fetchone()
extractor = recoll.Extractor(qdoc)
filename = extractor.idoctofile(qdoc.ipath, qdoc.mimetype)
```

4.3.2.5 Example code

The following sample would query the index with a user language string. See the `python/samples` directory inside the Recoll source for other examples. The `recollgui` subdirectory has a very embryonic GUI which demonstrates the highlighting and data extraction functions.

```
#!/usr/bin/env python

from recoll import recoll

db = recoll.connect()
db.setAbstractParams(maxchars=80, contextwords=4)

query = db.query()
nres = query.execute("some user question")
print "Result count: ", nres
if nres > 5:
    nres = 5
for i in range(nres):
    doc = query.fetchone()
    print "Result #%d" % (query.rownumber,)
    for k in ("title", "size"):
        print k, ":", getattr(doc, k).encode('utf-8')
    abs = db.makeDocAbstract(doc, query).encode('utf-8')
    print abs
    print
```

4.3.2.6 Compatibility with the previous version

The following code fragments can be used to ensure that code can run with both the old and the new API (as long as it does not use the new abilities of the new API of course).

Adapting to the new package structure:

```
try:
    from recoll import recoll
    from recoll import rclextract
    hasextract = True
except:
    import recoll
    hasextract = False
```

Adapting to the change of nature of the `next` `Query` member. The same test can be used to choose to use the `scroll()` method (new) or set the `next` value (old).

```
       rownum = query.next if type(query.next) == int else \
                query.rownumber
```

Chapter 5

Installation and configuration

5.1 Installing a binary copy

There are three types of binary Recoll installations:

- Through your system normal software distribution framework (ie, Debian/Ubuntu apt, FreeBSD ports, etc.).
- From a package downloaded from the Recoll web site.
- From a prebuilt tree downloaded from the Recoll web site.

In all cases, the strict software dependancies (ie on Xapian or iconv) will be automatically satisfied, you should not have to worry about them.

You will only have to check or install supporting applications for the file types that you want to index beyond those that are natively processed by Recoll (text, HTML, email files, and a few others).

You should also maybe have a look at the configuration section (but this may not be necessary for a quick test with default parameters). Most parameters can be more conveniently set from the GUI interface.

5.1.1 Installing through a package system

If you use a BSD-type port system or a prebuilt package (DEB, RPM, manually or through the system software configuration utility), just follow the usual procedure for your system.

5.1.2 Installing a prebuilt Recoll

The unpackaged binary versions on the Recoll web site are just compressed tar files of a build tree, where only the useful parts were kept (executables and sample configuration).

The executable binary files are built with a static link to libxapian and libiconv, to make installation easier (no dependencies).

After extracting the tar file, you can proceed with installation as if you had built the package from source (that is, just type `make install`). The binary trees are built for installation to `/usr/local`.

5.2 Supporting packages

Recoll uses external applications to index some file types. You need to install them for the file types that you wish to have indexed (these are run-time optional dependencies. None is needed for building or running Recoll except for indexing their specific file type).

After an indexing pass, the commands that were found missing can be displayed from the **recoll** File menu. The list is stored in the `missing` text file inside the configuration directory.

A list of common file types which need external commands follows. Many of the filters need the **iconv** command, which is not always listed as a dependancy.

Please note that, due to the relatively dynamic nature of this information, the most up to date version is now kept on the Recoll helper applications page along with links to the home pages or best source/patches pages, and misc tips. The list below is not updated often and may be quite stale.

For many Linux distributions, most of the commands listed can be installed from the package repositories. However, the packages are sometimes outdated, or not the best version for Recoll, so you should take a look at the Recoll helper applications page if a file type is important to you.

As of Recoll release 1.14, a number of XML-based formats that were handled by ad hoc filter code now use the **xsltproc** command, which usually comes with libxslt. These are: abiword, fb2 (ebooks), kword, openoffice, svg.

Now for the list:

- Openoffice files need **unzip** and **xsltproc**.
- PDF files need **pdftotext** which is part of the Xpdf or Poppler packages.
- Postscript files need **pstotext**. The original version has an issue with shell character in file names, which is corrected in recent packages. See the the Recoll helper applications page for more detail.
- MS Word needs **antiword**. It is also useful to have **wvWare** installed as it may be be used as a fallback for some files which **antiword** does not handle.
- MS Excel and PowerPoint need **catdoc**.
- MS Open XML (docx) needs **xsltproc**.
- Wordperfect files need **wpd2html** from the libwpd (or libwpd-tools on Ubuntu) package.
- RTF files need **unrtf**, which, in its standard version, has much trouble with non-western character sets. Check the Recoll helper applications page.
- TeX files need **untex** or **detex**. Check the Recoll helper applications page for sources if it's not packaged for your distribution.
- dvi files need **dvips**.
- djvu files need **djvutxt** and **djvused** from the DjVuLibre package.
- Audio files: Recoll releases before 1.13 used the **id3info** command from the id3lib package to extract mp3 tag information, **metaflac** (standard flac tools) for flac files, and **ogginfo** (vorbis tools) for ogg files. Releases 1.14 and later use a single Python filter based on mutagen for all audio file types.
- Pictures: Recoll uses the Exiftool Perl package to extract tag information. Most image file formats are supported. Note that there may not be much interest in indexing the technical tags (image size, aperture, etc.). This is only of interest if you store personal tags or textual descriptions inside the image files.
- chm: files in microsoft help format need Python and the pychm module (which needs chmlib).
- ICS: up to Recoll 1.13, iCalendar files need Python and the icalendar module. icalendar is not needed for newer versions, which use internal code.
- Zip archives need Python (and the standard zipfile module).
- Rar archives need Python, the rarfile Python module and the **unrar** utility.
- Midi karaoke files need Python and the Midi module
- Konqueror webarchive format with Python (uses the Tarfile module).
- mimehtml web archive format (support based on the email filter, which introduces some mild weirdness, but still usable).

Text, HTML, email folders, and Scribus files are processed internally. Lyx is used to index Lyx files. Many filters need **iconv** and the standard **sed** and **awk**.

5.3 Building from source

5.3.1 Prerequisites

If you can install any or all of the following through the package manager for your system, all the better. Especially Qt is a very big piece of software, but you will most probably be able to find a binary package.

You may have to compile Xapian but this is easy.

The shopping list:

- C++ compiler. Up to Recoll version 1.13.04, its absence can manifest itself by strange messages about a missing iconv_open.
- Development files for Xapian core.

Important
If you are building Xapian for an older CPU (before Pentium 4 or Athlon 64), you need to add the `--disable-sse` flag to the configure command. Else all Xapian application will crash with an `illegal instruction` error.

- Development files for Qt 4 . Recoll has not been tested with Qt 5 yet. Recoll 1.15.9 was the last version to support Qt 3. If you do not want to install or build the Qt Webkit module, Recoll has a configuration option to disable its use (see further).
- Development files for X11 and zlib.
- You may also need libiconv. On Linux systems, the iconv interface is part of libc and you should not need to do anything special.

Check the Recoll download page for up to date version information.

5.3.2 Building

Recoll has been built on Linux, FreeBSD, Mac OS X, and Solaris, most versions after 2005 should be ok, maybe some older ones too (Solaris 8 is ok). If you build on another system, and need to modify things, I would very much welcome patches.

Configure options:

- `--without-aspell` will disable the code for phonetic matching of search terms.
- `--with-fam` or `--with-inotify` will enable the code for real time indexing. Inotify support is enabled by default on recent Linux systems.
- `--with-qzeitgeist` will enable sending Zeitgeist events about the visited search results, and needs the qzeitgeist package.
- `--disable-webkit` is available from version 1.17 to implement the result list with a Qt QTextBrowser instead of a WebKit widget if you do not or can't depend on the latter.
- `--disable-idxthreads` is available from version 1.19 to suppress multithreading inside the indexing process. You can also use the run-time configuration to restrict **recollindex** to using a single thread, but the compile-time option may disable a few more unused locks. This only applies to the use of multithreading for the core index processing (data input). The Recoll monitor mode always uses at least two threads of execution.
- `--disable-python-module` will avoid building the Python module.
- `--disable-xattr` will prevent fetching data from file extended attributes. Beyond a few standard attributes, fetching extended attributes data can only be useful is some application stores data in there, and also needs some simple configuration (see comments in the `fields` configuration file).

- `--enable-camelcase` will enable splitting *camelCase* words. This is not enabled by default as it has the unfortunate side-effect of making some phrase searches quite confusing: ie, `"MySQL manual"` would be matched by `"MySQL manual"` and `"my sql manual"` but not `"mysql manual"` (only inside phrase searches).
- `--with-file-command` Specify the version of the 'file' command to use (ie: --with-file-command=/usr/local/bin/file). Can be useful to enable the gnu version on systems where the native one is bad.
- `--disable-qtgui` Disable the Qt interface. Will allow building the indexer and the command line search program in absence of a Qt environment.
- `--disable-x11mon` Disable X11 connection monitoring inside recollindex. Together with --disable-qtgui, this allows building recoll without Qt and X11.
- `--disable-pic` will compile Recoll with position-dependant code. This is incompatible with building the KIO or the Python or PHP extensions, but might yield very marginally faster code.
- Of course the usual autoconf **configure** options, like `--prefix` apply.

Normal procedure:

```
cd recoll-xxx
configure
make
(practices usual hardship-repelling invocations)
```

There is little auto-configuration. The **configure** script will mainly link one of the system-specific files in the `mk` directory to `mk/sysconf`. If your system is not known yet, it will tell you as much, and you may want to manually copy and modify one of the existing files (the new file name should be the output of **uname** `-s`).

5.3.2.1 Building on Solaris

We did not test building the GUI on Solaris for recent versions. You will need at least Qt 4.4. There are some hints on an old web site page, they may still be valid.

Someone did test the 1.19 indexer and Python module build, they do work, with a few minor glitches. Be sure to use GNU **make** and **install**.

5.3.3 Installation

Either type **`make install`** or execute **`recollinstall`** ***`prefix`***, in the root of the source tree. This will copy the commands to *`prefix`*`/bin` and the sample configuration files, scripts and other shared data to *`prefix`*`/share/recoll`.

If the installation prefix given to **recollinstall** is different from either the system default or the value which was specified when executing **configure** (as in **`configure --prefix /some/path`**), you will have to set the `RECOLL_DATADIR` environment variable to indicate where the shared data is to be found (ie for (ba)sh: **`export RECOLL_DATADIR=/some/path/share/recol`**

You can then proceed to configuration.

5.4 Configuration overview

Most of the parameters specific to the **recoll** GUI are set through the Preferences menu and stored in the standard Qt place (`$HOME/.config/Recoll.org/recoll.conf`). You probably do not want to edit this by hand.

Recoll indexing options are set inside text configuration files located in a configuration directory. There can be several such directories, each of which defines the parameters for one index.

The configuration files can be edited by hand or through the Index configuration dialog (Preferences menu). The GUI tool will try to respect your formatting and comments as much as possible, so it is quite possible to use both ways.

The most accurate documentation for the configuration parameters is given by comments inside the default files, and we will just give a general overview here.

By default, for each index, there are two sets of configuration files. System-wide configuration files are kept in a directory named like `/usr/[local/]share/recoll/examples`, and define default values, shared by all indexes. For each index, a parallel set of files defines the customized parameters.

In addition (as of Recoll version 1.19.7), it is possible to specify two additional configuration directories which will be stacked before and after the user configuration directory. These are defined by the `RECOLL_CONFTOP` and `RECOLL_CONFMID` environment variables. Values from configuration files inside the top directory will override user ones, values from configuration files inside the middle directory will override system ones and be overriden by user ones. These two variables may be of use to applications which augment Recoll functionality, and need to add configuration data without disturbing the user's files. Please note that the two, currently single, values will probably be interpreted as colon-separated lists in the future: do not use colon characters inside the directory paths.

The default location of the configuration is the `.recoll` directory in your home. Most people will only use this directory.

This location can be changed, or others can be added with the `RECOLL_CONFDIR` environment variable or the `-c` option parameter to **recoll** and **recollindex**.

If the `.recoll` directory does not exist when **recoll** or **recollindex** are started, it will be created with a set of empty configuration files. **recoll** will give you a chance to edit the configuration file before starting indexing. **recollindex** will proceed immediately. To avoid mistakes, the automatic directory creation will only occur for the default location, not if `-c` or `RECOLL_CONFDIR` were used (in the latter cases, you will have to create the directory).

All configuration files share the same format. For example, a short extract of the main configuration file might look as follows:

```
# Space-separated list of directories to index.
topdirs =  ~/docs /usr/share/doc

[~/somedirectory-with-utf8-txt-files]
defaultcharset = utf-8
```

There are three kinds of lines:

- Comment (starts with #) or empty.
- Parameter affectation (*name* = *value*).
- Section definition ([*somedirname*]).

Depending on the type of configuration file, section definitions either separate groups of parameters or allow redefining some parameters for a directory sub-tree. They stay in effect until another section definition, or the end of file, is encountered. Some of the parameters used for indexing are looked up hierarchically from the current directory location upwards. Not all parameters can be meaningfully redefined, this is specified for each in the next section.

When found at the beginning of a file path, the tilde character (~) is expanded to the name of the user's home directory, as a shell would do.

White space is used for separation inside lists. List elements with embedded spaces can be quoted using double-quotes.

Encoding issues Most of the configuration parameters are plain ASCII. Two particular sets of values may cause encoding issues:

- File path parameters may contain non-ascii characters and should use the exact same byte values as found in the file system directory. Usually, this means that the configuration file should use the system default locale encoding.
- The `unac_except_trans` parameter should be encoded in UTF-8. If your system locale is not UTF-8, and you need to also specify non-ascii file paths, this poses a difficulty because common text editors cannot handle multiple encodings in a single file. In this relatively unlikely case, you can edit the configuration file as two separate text files with appropriate encodings, and concatenate them to create the complete configuration.

5.4.1 The main configuration file, recoll.conf

`recoll.conf` is the main configuration file. It defines things like what to index (top directories and things to ignore), and the default character set to use for document types which do not specify it internally.

The default configuration will index your home directory. If this is not appropriate, start **recoll** to create a blank configuration, click Cancel, and edit the configuration file before restarting the command. This will start the initial indexing, which may take some time.

Most of the following parameters can be changed from the Index Configuration menu in the **recoll** interface. Some can only be set by editing the configuration file.

5.4.1.1 Parameters affecting what documents we index:

`topdirs` Specifies the list of directories or files to index (recursively for directories). You can use symbolic links as elements of this list. See the `followLinks` option about following symbolic links found under the top elements (not followed by default).

`skippedNames` A space-separated list of patterns for names of files or directories that should be completely ignored. The list defined in the default file is:

```
skippedNames = #* bin CVS  Cache cache* caughtspam  tmp .thumbnails .svn \
         *~ .beagle .git .hg .bzr loop.ps .xsession-errors \
         .recoll* xapiandb recollrc recoll.conf
```

The list can be redefined at any sub-directory in the indexed area.

The top-level directories are not affected by this list (that is, a directory in `topdirs` might match and would still be indexed).

The list in the default configuration does not exclude hidden directories (names beginning with a dot), which means that it may index quite a few things that you do not want. On the other hand, email user agents like thunderbird usually store messages in hidden directories, and you probably want this indexed. One possible solution is to have `.*` in `skippedNames`, and add things like `~/.thunderbird` or `~/.evolution` in `topdirs`.

Not even the file names are indexed for patterns in this list. See the `recoll_noindex` variable in `mimemap` for an alternative approach which indexes the file names.

`skippedPaths` and `daemSkippedPaths` A space-separated list of patterns for *paths* of files or directories that should be skipped. There is no default in the sample configuration file, but the code always adds the configuration and database directories in there.

`skippedPaths` is used both by batch and real time indexing. `daemSkippedPaths` can be used to specify things that should be indexed at startup, but not monitored.

Example of use for skipping text files only in a specific directory:

```
skippedPaths = ~/somedir/*.txt
```

`skippedPathsFnmPathname` The values in the `*skippedPaths` variables are matched by default with `fnmatch(3)`, with the FNM_PATHNAME and FNM_LEADING_DIR flags. This means that '/' characters must be matched explicitely. You can set `skippedPathsFnmPathname` to 0 to disable the use of FNM_PATHNAME (meaning that /*/dir3 will match /dir1/dir2/dir3).

`zipSkippedNames` A space-separated list of patterns for names of files or directories that should be ignored inside zip archives. This is used directly by the zip filter, and has a function similar to skippedNames, but works independantly. Can be redefined for filesystem subdirectories. For versions up to 1.19, you will need to update the Zip filter and install a supplementary Python module. The details are described on the Recoll wiki.

`followLinks` Specifies if the indexer should follow symbolic links while walking the file tree. The default is to ignore symbolic links to avoid multiple indexing of linked files. No effort is made to avoid duplication when this option is set to true. This option can be set individually for each of the `topdirs` members by using sections. It can not be changed below the `topdirs` level.

indexedmimetypes Recoll normally indexes any file which it knows how to read. This list lets you restrict the indexed mime types to what you specify. If the variable is unspecified or the list empty (the default), all supported types are processed. Can be redefined for subdirectories.

compressedfilemaxkbs Size limit for compressed (.gz or .bz2) files. These need to be decompressed in a temporary directory for identification, which can be very wasteful if 'uninteresting' big compressed files are present. Negative means no limit, 0 means no processing of any compressed file. Defaults to -1.

textfilemaxmbs Maximum size for text files. Very big text files are often uninteresting logs. Set to -1 to disable (default 20MB).

textfilepagekbs If set to other than -1, text files will be indexed as multiple documents of the given page size. This may be useful if you do want to index very big text files as it will both reduce memory usage at index time and help with loading data to the preview window. A size of a few megabytes would seem reasonable (default: 1MB).

membermaxkbs This defines the maximum size in kilobytes for an archive member (zip, tar or rar at the moment). Bigger entries will be skipped.

indexallfilenames Recoll indexes file names in a special section of the database to allow specific file names searches using wild cards. This parameter decides if file name indexing is performed only for files with mime types that would qualify them for full text indexing, or for all files inside the selected subtrees, independently of mime type.

usesystemfilecommand Decide if we use the **file** `-i` system command as a final step for determining the mime type for a file (the main procedure uses suffix associations as defined in the `mimemap` file). This can be useful for files with suffix-less names, but it will also cause the indexing of many bogus "text" files.

processwebqueue If this is set, process the directory where Web browser plugins copy visited pages for indexing.

webqueuedir The path to the web indexing queue. This is hard-coded in the Firefox plugin as `~/.recollweb/ToIndex` so there should be no need to change it.

5.4.1.2 Parameters affecting how we generate terms:

Changing some of these parameters will imply a full reindex. Also, when using multiple indexes, it may not make sense to search indexes that don't share the values for these parameters, because they usually affect both search and index operations.

indexStripChars Decide if we strip characters of diacritics and convert them to lower-case before terms are indexed. If we don't, searches sensitive to case and diacritics can be performed, but the index will be bigger, and some marginal weirdness may sometimes occur. The default is a stripped index (`indexStripChars =1`) for now. When using multiple indexes for a search, this parameter must be defined identically for all. Changing the value implies an index reset.

maxTermExpand Maximum expansion count for a single term (e.g.: when using wildcards). The default of 10000 is reasonable and will avoid queries that appear frozen while the engine is walking the term list.

maxXapianClauses Maximum number of elementary clauses we can add to a single Xapian query. In some cases, the result of term expansion can be multiplicative, and we want to avoid using excessive memory. The default of 100 000 should be both high enough in most cases and compatible with current typical hardware configurations.

nonumbers If this set to true, no terms will be generated for numbers. For example "123", "1.5e6", 192.168.1.4, would not be indexed ("value123" would still be). Numbers are often quite interesting to search for, and this should probably not be set except for special situations, ie, scientific documents with huge amounts of numbers in them. This can only be set for a whole index, not for a subtree.

nocjk If this set to true, specific east asian (Chinese Korean Japanese) characters/word splitting is turned off. This will save a small amount of cpu if you have no CJK documents. If your document base does include such text but you are not interested in searching it, setting `nocjk` may be a significant time and space saver.

cjkngramlen This lets you adjust the size of n-grams used for indexing CJK text. The default value of 2 is probably appropriate in most cases. A value of 3 would allow more precision and efficiency on longer words, but the index will be approximately twice as large.

indexstemminglanguages A list of languages for which the stem expansion databases will be built. See recollindex(1) or use the **recollindex** -l command for possible values. You can add a stem expansion database for a different language by using **recollindex** -s, but it will be deleted during the next indexing. Only languages listed in the configuration file are permanent.

defaultcharset The name of the character set used for files that do not contain a character set definition (ie: plain text files). This can be redefined for any sub-directory. If it is not set at all, the character set used is the one defined by the nls environment (LC_ALL, LC_CTYPE, LANG), or iso8859-1 if nothing is set.

unac_except_trans This is a list of characters, encoded in UTF-8, which should be handled specially when converting text to unaccented lowercase. For example, in Swedish, the letter a with diaeresis has full alphabet citizenship and should not be turned into an a. Each element in the space-separated list has the special character as first element and the translation following. The handling of both the lowercase and upper-case versions of a character should be specified, as appartenance to the list will turn-off both standard accent and case processing. Example for Swedish:

```
unac_except_trans =  åå Åå ää Ää öö Öö
```

Note that the translation is not limited to a single character, you could very well have something like üue in the list.

The default value set for unac_except_trans can't be listed here because I have trouble with SGML and UTF-8, but it only contains ligature decompositions: german ss, oe, ae, fi, fl.

This parameter can't be defined for subdirectories, it is global, because there is no way to do otherwise when querying. If you have document sets which would need different values, you will have to index and query them separately.

maildefcharset This can be used to define the default character set specifically for email messages which don't specify it. This is mainly useful for readpst (libpst) dumps, which are utf-8 but do not say so.

localfields This allows setting fields for all documents under a given directory. Typical usage would be to set an "rclaptg" field, to be used in mimeview to select a specific viewer. If several fields are to be set, they should be separated with a semi-colon (';') character, which there is currently no way to escape. Also note the initial semi-colon. Example: localfields=;rclaptg=gnus;other =val, then select specifier viewer with mimetype|tag=... in mimeview.

noxattrfields Recoll versions 1.19 and later automatically translate file extended attributes into document fields (to be processed according to the parameters from the fields file). Setting this variable to 1 will disable the behaviour.

metadatacmds This allows executing external commands for each file and storing the output in Recoll document fields. This could be used for example to index external tag data. The value is a list of field names and commands, don't forget an initial semi-colon. Example:

```
[/some/area/of/the/fs]
metadatacmds = ; tags = tmsu tags %f; otherfield = somecmd -xx %f
```

As a specially disgusting hack brought by Recoll 1.19.7, if a "field name" begins with rclmulti, the data returned by the command is expected to contain multiple field values, in configuration file format. This allows setting several fields by executing a single command. Example:

```
metadatacmds = ; rclmulti1 = somecmd %f
```

If somecmd returns data in the form of:

```
field1 = value1
field2 = "value for field2"
```

field1 and field2 will be set inside the document metadata.

5.4.1.3 Parameters affecting where and how we store things:

dbdir The name of the Xapian data directory. It will be created if needed when the index is initialized. If this is not an absolute path, it will be interpreted relative to the configuration directory. The value can have embedded spaces but starting or trailing spaces will be trimmed. You cannot use quotes here.

idxstatusfile The name of the scratch file where the indexer process updates its status. Default: `idxstatus.txt` inside the configuration directory.

maxfsoccuppc Maximum file system occupation before we stop indexing. The value is a percentage, corresponding to what the "Capacity" df output column shows. The default value is 0, meaning no checking.

mboxcachedir The directory where mbox message offsets cache files are held. This is normally $RECOLL_CONFDIR/mboxcache, but it may be useful to share a directory between different configurations.

mboxcacheminmbs The minimum mbox file size over which we cache the offsets. There is really no sense in caching offsets for small files. The default is 5 MB.

webcachedir This is only used by the web browser plugin indexing code, and defines where the cache for visited pages will live. Default: `$RECOLL_CONFDIR/webcache`

webcachemaxmbs This is only used by the web browser plugin indexing code, and defines the maximum size for the web page cache. Default: 40 MB.

idxflushmb Threshold (megabytes of new text data) where we flush from memory to disk index. Setting this can help control memory usage. A value of 0 means no explicit flushing, letting Xapian use its own default, which is flushing every 10000 (or XAPIAN_FLUSH_THRESHOLD) documents, which gives little memory usage control, as memory usage also depends on average document size. The default value is 10, and it is probably a bit low. If your system usually has free memory, you can try higher values between 20 and 80. In my experience, values beyond 100 are always counterproductive.

5.4.1.4 Parameters affecting multithread processing

The Recoll indexing process **recollindex** can use multiple threads to speed up indexing on multiprocessor systems. The work done to index files is divided in several stages and some of the stages can be executed by multiple threads. The stages are:

1. File system walking: this is always performed by the main thread.
2. File conversion and data extraction.
3. Text processing (splitting, stemming, etc.)
4. Xapian index update.

You can also read a longer document about the transformation of Recoll indexing to multithreading.

The threads configuration is controlled by two configuration file parameters.

thrQSizes This variable defines the job input queues configuration. There are three possible queues for stages 2, 3 and 4, and this parameter should give the queue depth for each stage (three integer values). If a value of -1 is used for a given stage, no queue is used, and the thread will go on performing the next stage. In practise, deep queues have not been shown to increase performance. A value of 0 for the first queue tells Recoll to perform autoconfiguration (no need for the two other values in this case) - this is the default configuration.

thrTCounts This defines the number of threads used for each stage. If a value of -1 is used for one of the queue depths, the corresponding thread count is ignored. It makes no sense to use a value other than 1 for the last stage because updating the Xapian index is necessarily single-threaded (and protected by a mutex).

The following example would use three queues (of depth 2), and 4 threads for converting source documents, 2 for processing their text, and one to update the index. This was tested to be the best configuration on the test system (quadri-processor with multiple disks).

```
thrQSizes = 2 2 2
thrTCounts =  4 2 1
```

The following example would use a single queue, and the complete processing for each document would be performed by a single thread (several documents will still be processed in parallel in most cases). The threads will use mutual exclusion when entering the index update stage. In practise the performance would be close to the precedent case in general, but worse in certain cases (e.g. a Zip archive would be performed purely sequentially), so the previous approach is preferred. YMMV... The 2 last values for thrTCounts are ignored.

```
thrQSizes = 2 -1 -1
thrTCounts =  6 1 1
```

The following example would disable multithreading. Indexing will be performed by a single thread.

```
thrQSizes = -1 -1 -1
```

5.4.1.5 Miscellaneous parameters:

autodiacsens IF the index is not stripped, decide if we automatically trigger diacritics sensitivity if the search term has accented characters (not in `unac_except_trans`). Else you need to use the query language and the `D` modifier to specify diacritics sensitivity. Default is no.

autocasesens IF the index is not stripped, decide if we automatically trigger character case sensitivity if the search term has upper-case characters in any but the first position. Else you need to use the query language and the `C` modifier to specify character-case sensitivity. Default is yes.

loglevel,daemloglevel Verbosity level for recoll and recollindex. A value of 4 lists quite a lot of debug/information messages. 2 only lists errors. The `daem`version is specific to the indexing monitor daemon.

logfilename, daemlogfilename Where the messages should go. 'stderr' can be used as a special value, and is the default. The `daem`version is specific to the indexing monitor daemon.

mondelaypatterns This allows specify wildcard path patterns (processed with fnmatch(3) with 0 flag), to match files which change too often and for which a delay should be observed before re-indexing. This is a space-separated list, each entry being a pattern and a time in seconds, separated by a colon. You can use double quotes if a path entry contains white space. Example:

```
mondelaypatterns = *.log:20 "this one has spaces*:10"
```

monixinterval Minimum interval (seconds) for processing the indexing queue. The real time monitor does not process each event when it comes in, but will wait this time for the queue to accumulate to diminish overhead and in order to aggregate multiple events to the same file. Default 30 S.

monauxinterval Period (in seconds) at which the real time monitor will regenerate the auxiliary databases (spelling, stemming) if needed. The default is one hour.

monioniceclass, monioniceclassdata These allow defining the ionice class and data used by the indexer (default class 3, no data).

filtermaxseconds Maximum filter execution time, after which it is aborted. Some postscript programs just loop...

filtersdir A directory to search for the external filter scripts used to index some types of files. The value should not be changed, except if you want to modify one of the default scripts. The value can be redefined for any sub-directory.

iconsdir The name of the directory where **recoll** result list icons are stored. You can change this if you want different images.

idxabsmlen Recoll stores an abstract for each indexed file inside the database. The text can come from an actual 'abstract' section in the document or will just be the beginning of the document. It is stored in the index so that it can be displayed inside the result lists without decoding the original file. The `idxabsmlen` parameter defines the size of the stored abstract. The default value is 250 bytes. The search interface gives you the choice to display this stored text or a synthetic abstract built by extracting text around the search terms. If you always prefer the synthetic abstract, you can reduce this value and save a little space.

aspellLanguage Language definitions to use when creating the aspell dictionary. The value must match a set of aspell language definition files. You can type "aspell config" to see where these are installed (look for data-dir). The default if the variable is not set is to use your desktop national language environment to guess the value.

noaspell If this is set, the aspell dictionary generation is turned off. Useful for cases where you don't need the functionality or when it is unusable because aspell crashes during dictionary generation.

mhmboxquirks This allows defininining location-related quirks for the mailbox handler. Currently only the `tbird` flag is defined, and it should be set for directories which hold Thunderbird data, as their folder format is weird.

5.4.2 The fields file

This file contains information about dynamic fields handling in Recoll. Some very basic fields have hard-wired behaviour, and, mostly, you should not change the original data inside the `fields` file. But you can create custom fields fitting your data and handle them just like they were native ones.

The `fields` file has several sections, which each define an aspect of fields processing. Quite often, you'll have to modify several sections to obtain the desired behaviour.

We will only give a short description here, you should refer to the comments inside the file for more detailed information.

Field names should be lowercase alphabetic ASCII.

[prefixes] A field becomes indexed (searchable) by having a prefix defined in this section.

[stored] A field becomes stored (displayable inside results) by having its name listed in this section (typically with an empty value).

[aliases] This section defines lists of synonyms for the canonical names used inside the `[prefixes]` and `[stored]` sections

filter-specific sections Some filters may need specific configuration for handling fields. Only the email message filter currently has such a section (named `[mail]`). It allows indexing arbitrary email headers in addition to the ones indexed by default. Other such sections may appear in the future.

Here follows a small example of a personal `fields` file. This would extract a specific email header and use it as a searchable field, with data displayable inside result lists. (Side note: as the email filter does no decoding on the values, only plain ascii headers can be indexed, and only the first occurrence will be used for headers that occur several times).

```
[prefixes]
# Index mailmytag contents (with the given prefix)
mailmytag = XMTAG

[stored]
# Store mailmytag inside the document data record (so that it can be
# displayed - as %(mailmytag) - in result lists).
mailmytag =

[mail]
# Extract the X-My-Tag mail header, and use it internally with the
# mailmytag field name
x-my-tag = mailmytag
```

5.4.2.1 Extended attributes in the fields file

Recoll versions 1.19 and later process user extended file attributes as documents fields by default.

Attributes are processed as fields of the same name, after removing the `user` prefix on Linux.

The `[xattrtofields]` section of the `fields` file allows specifying translations from extended attributes names to Recoll field names. An empty translation disables use of the corresponding attribute data.

5.4.3 The mimemap file

mimemap specifies the file name extension to mime type mappings.

For file names without an extension, or with an unknown one, the system's **file** -i command will be executed to determine the mime type (this can be switched off inside the main configuration file).

The mappings can be specified on a per-subtree basis, which may be useful in some cases. Example: gaim logs have a .txt extension but should be handled specially, which is possible because they are usually all located in one place.

mimemap also has a recoll_noindex variable which is a list of suffixes. Matching files will be skipped (which avoids unnecessary decompressions or **file** executions). This is partially redundant with skippedNames in the main configuration file, with a few differences: it will not affect directories, it cannot be made dependant on the file-system location (it is a configuration-wide parameter), and the file names will still be indexed (not even the file names are indexed for patterns in skippedNames. recoll_noindex is used mostly for things known to be unindexable by a given Recoll version. Having it there avoids cluttering the more user-oriented and locally customized skippedNames.

5.4.4 The mimeconf file

mimeconf specifies how the different mime types are handled for indexing, and which icons are displayed in the **recoll** result lists.

Changing the parameters in the [index] section is probably not a good idea except if you are a Recoll developer.

The [icons] section allows you to change the icons which are displayed by **recoll** in the result lists (the values are the basenames of the png images inside the iconsdir directory (specified in recoll.conf).

5.4.5 The mimeview file

mimeview specifies which programs are started when you click on an Open link in a result list. Ie: HTML is normally displayed using firefox, but you may prefer Konqueror, your openoffice.org program might be named **oofice** instead of **openoffice** etc.

Changes to this file can be done by direct editing, or through the **recoll** GUI preferences dialog.

If Use desktop preferences to choose document editor is checked in the Recoll GUI preferences, all mimeview entries will be ignored except the one labelled application/x-all (which is set to use **xdg-open** by default).

In this case, the xallexcepts top level variable defines a list of mime type exceptions which will be processed according to the local entries instead of being passed to the desktop. This is so that specific Recoll options such as a page number or a search string can be passed to applications that support them, such as the evince viewer.

As for the other configuration files, the normal usage is to have a mimeview inside your own configuration directory, with just the non-default entries, which will override those from the central configuration file.

All viewer definition entries must be placed under a [view] section.

The keys in the file are normally mime types. You can add an application tag to specialize the choice for an area of the filesystem (using a localfields specification in mimeconf). The syntax for the key is *mimetype|tag*

The nouncompforviewmts entry, (placed at the top level, outside of the [view] section), holds a list of mime types that should not be uncompressed before starting the viewer (if they are found compressed, ie: *mydoc.doc.gz*).

The right side of each assignment holds a command to be executed for opening the file. The following substitutions are performed:

- **%D** Document date
- **%f** File name. This may be the name of a temporary file if it was necessary to create one (ie: to extract a subdocument from a container).
- **%F** Original file name. Same as %f except if a temporary file is used.
- **%i** Internal path, for subdocuments of containers. The format depends on the container type. If this appears in the command line, Recoll will not create a temporary file to extract the subdocument, expecting the called application (possibly a script) to be able to handle it.

- **%M** Mime type
- **%p** Page index. Only significant for a subset of document types, currently only PDF, Postscript and DVI files. Can be used to start the editor at the right page for a match or snippet.
- **%s** Search term. The value will only be set for documents with indexed page numbers (ie: PDF). The value will be one of the matched search terms. It would allow pre-setting the value in the "Find" entry inside Evince for example, for easy highlighting of the term.
- **%U, %u** Url.

In addition to the predefined values above, all strings like `%(fieldname)` will be replaced by the value of the field named `fieldname` for the document. This could be used in combination with field customisation to help with opening the document.

5.4.6 The `ptrans` file

`ptrans` specifies query-time path translations. These can be useful in multiple cases.

The file has a section for any index which needs translations, either the main one or additional query indexes. The sections are named with the Xapian index directory names. No slash character should exist at the end of the paths (all comparisons are textual). An exemple should make things sufficiently clear

```
[/home/me/.recoll/xapiandb]
/this/directory/moved = /to/this/place

[/path/to/additional/xapiandb]
/server/volume1/docdir = /net/server/volume1/docdir
/server/volume2/docdir = /net/server/volume2/docdir
```

5.4.7 Examples of configuration adjustments

5.4.7.1 Adding an external viewer for an non-indexed type

Imagine that you have some kind of file which does not have indexable content, but for which you would like to have a functional Open link in the result list (when found by file name). The file names end in *`.blob`* and can be displayed by application *`blobviewer`*.

You need two entries in the configuration files for this to work:

- In `$RECOLL_CONFDIR/mimemap` (typically `~/.recoll/mimemap`), add the following line:

  ```
  .blob = application/x-blobapp
  ```

  Note that the mime type is made up here, and you could call it *`diesel/oil`* just the same.

- In `$RECOLL_CONFDIR/mimeview` under the `[view]` section, add:

  ```
  application/x-blobapp = blobviewer %f
  ```

  We are supposing that *`blobviewer`* wants a file name parameter here, you would use `%u` if it liked URLs better.

If you just wanted to change the application used by Recoll to display a mime type which it already knows, you would just need to edit `mimeview`. The entries you add in your personal file override those in the central configuration, which you do not need to alter. `mimeview` can also be modified from the Gui.

5.4.7.2 Adding indexing support for a new file type

Let us now imagine that the above *.blob* files actually contain indexable text and that you know how to extract it with a command line program. Getting Recoll to index the files is easy. You need to perform the above alteration, and also to add data to the `mimeconf` file (typically in `~/.recoll/mimeconf`):

- Under the `[index]` section, add the following line (more about the *rclblob* indexing script later):

```
application/x-blobapp = exec rclblob
```

- Under the `[icons]` section, you should choose an icon to be displayed for the files inside the result lists. Icons are normally 64x64 pixels PNG files which live in `/usr/[local/]share/recoll/images`.
- Under the `[categories]` section, you should add the mime type where it makes sense (you can also create a category). Categories may be used for filtering in advanced search.

The *rclblob* filter should be an executable program or script which exists inside `/usr/[local/]share/recoll/filters`. It will be given a file name as argument and should output the text or html contents on the standard output.

The filter programming section describes in more detail how to write a filter.